REPÚBLICA DEL ECUADOR

CARRERA EN
DESARROLLO DE SOFTWARE

MANUAL TÉCNICO

Desarrollo de una aplicación móvil para la administración de pedidos, productos, ventas y usuarios en la promoción de la gastronomía mexicana: Azteca Fest.

Instituto Superior Tecnológico Tena
Documentación Técnica Avanzada
Aplicación Móvil Funcional

Desarrollado por:
Andrés Andy, Winston Alvarado

Año 2025

ÍNDICE

1 INTRODUCCION Y OBJETIVOS

1.1 Propósito del Documento

Este manual proporciona una documentación completa para desarrolladores, administradores de sistemas y equipos técnicos sobre la aplicación móvil Azteca Fest. Su objetivo es servir como guía de referencia para el desarrollo, despliegue, mantenimiento y escalamiento del sistema.

1.2 Alcance del Proyecto

Azteca Fest es una solución integral para la gestión de negocios de gastronomía mexicana que incluye:

- **Aplicación Móvil: Flutter para Android, iOS y Web**
- **Backend: Supabase (PostgreSQL, Authentication, Storage)**
- **Módulos Principales:**
  - **Gestión de productos e inventario**
  - **Sistema de pedidos en tiempo real**
  - **POS (Point of Sale) para ventas directas**
  - **Dashboard administrativo multi-rol**
  - **Reportes y analíticas**
  - **Sistema de notificaciones push**

1.3 Tecnologías Principales

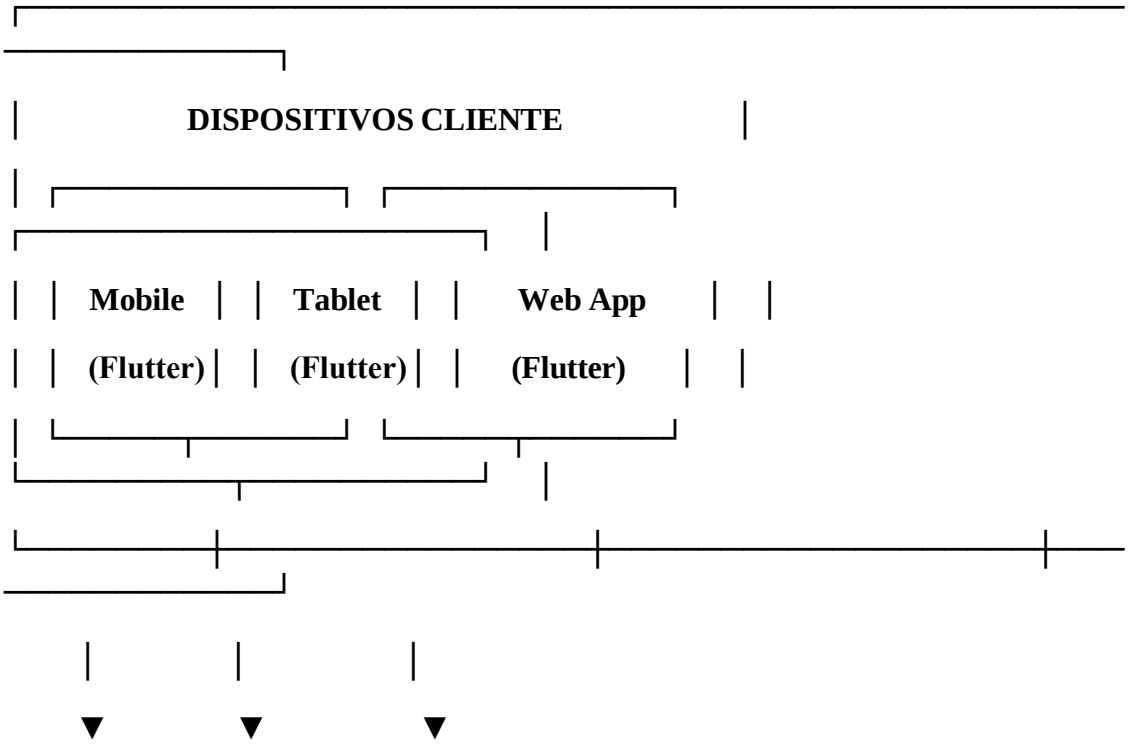
Tecnologías	Versión	Propósitos
Flutter	3.38.3+	Framework UI multiplataforma
Dart	3.10.1+	Lenguaje de programación
Supabase	2.15.2+	Backend como servicio
Supabase	2.15.2+	Base de datos relacional
Supabase	2.15.2+	Notificaciones y Analytics
GitHub Actions		CI/CD Pipeline

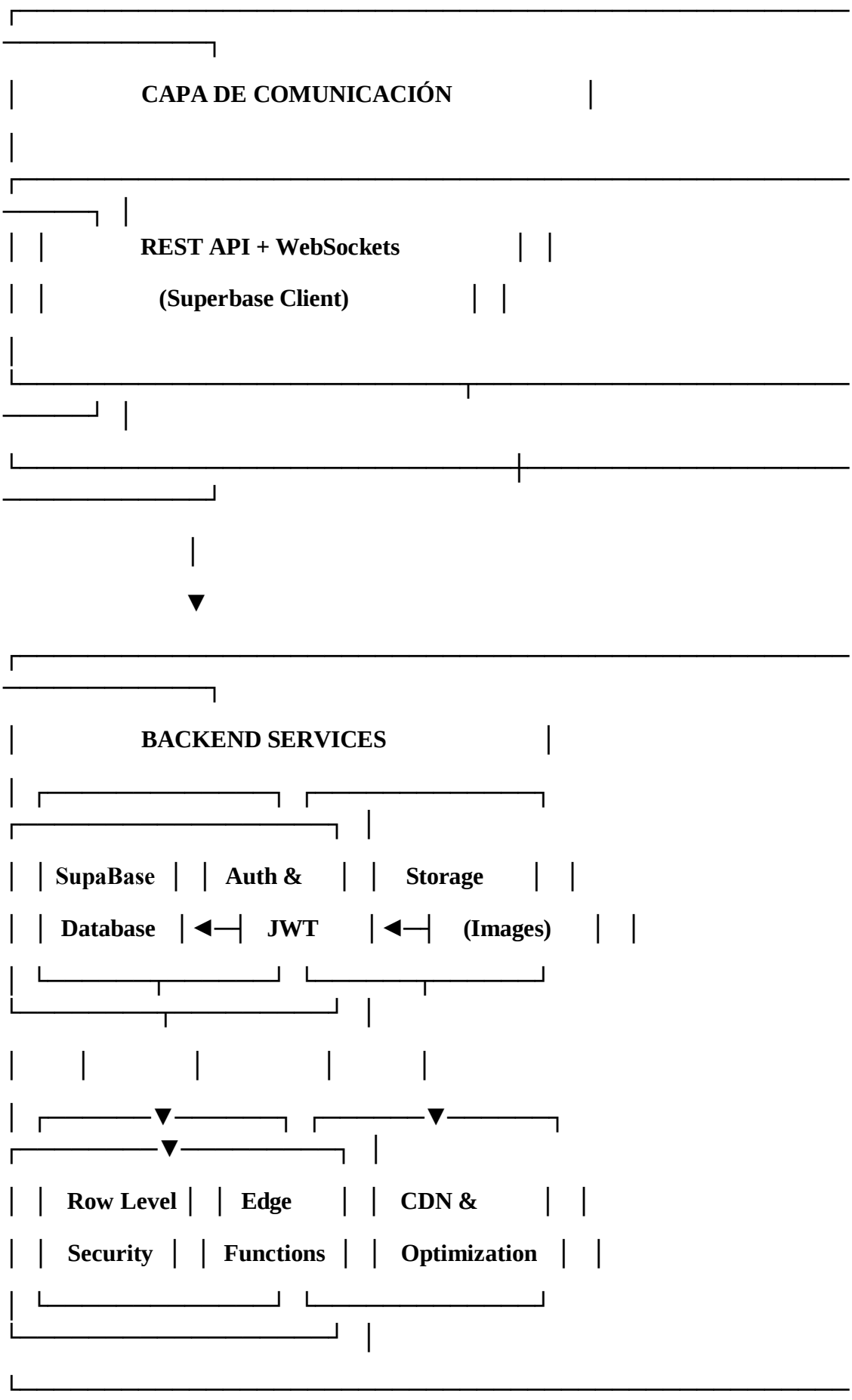
1.4 Roles del Sistema

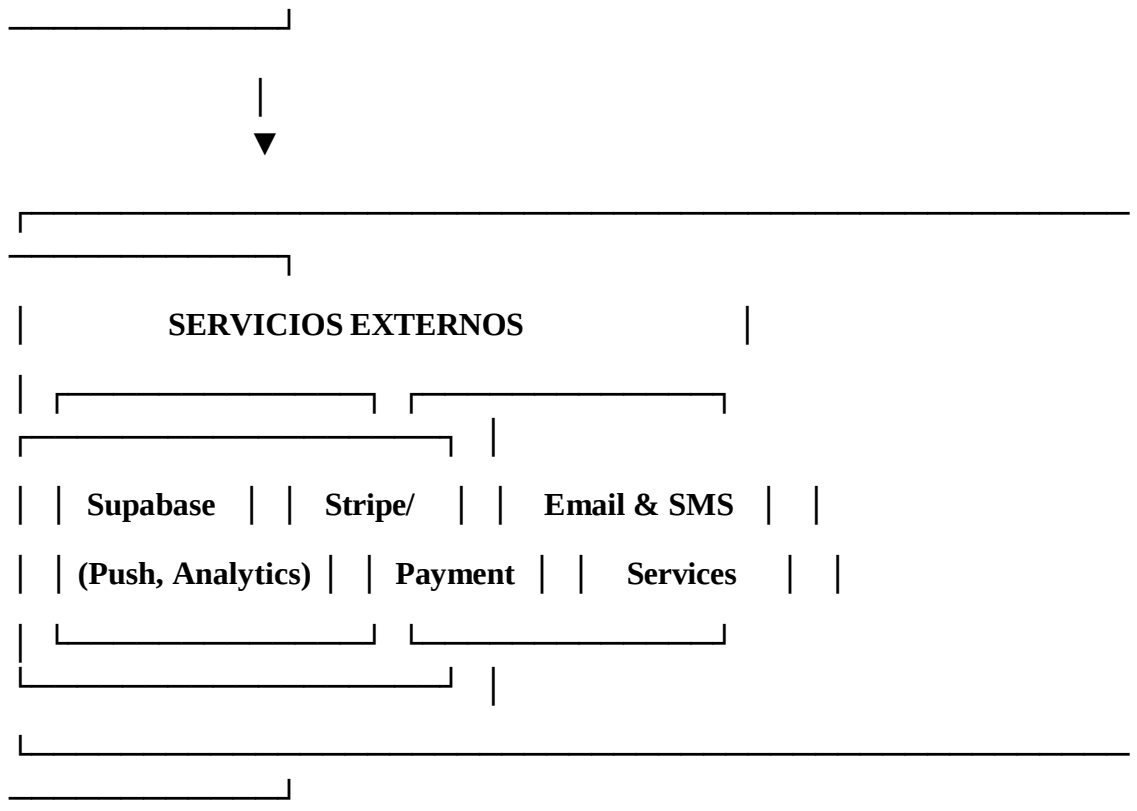
1. **Administrador: Acceso completo, gestión de usuarios, reportes**
2. **Vendedor: Ventas directas, atención de pedidos, gestión de clientes**
3. **Cliente: Catálogo, pedidos, seguimiento, perfil personal**

2 ARQUITECTURA DEL SISTEMA

2.1 Diagrama de Arquitectura




CAPA DE COMUNICACIÓN
REST API + WebSockets
(Superbase Client)
BACKEND SERVICES
SupaBase
Database
Auth &
JWT
Storage
(Images)
Row Level
Security
Edge
Functions
CDN &
Optimization




```

## 2.2 Patrones de Diseño Implementados

1. **MVVM (Model-View-ViewModel): Separación clara entre lógica y presentación**
2. **Provider Pattern: Gestión de estado reactiva**
3. **Repository/DAO Pattern: Abstracción del acceso a datos**
4. **Dependency Injection: Inyección de dependencias vía constructores**
5. **Strategy Pattern: Para diferentes implementaciones de DAOs**
6. **Observer Pattern: Para notificaciones y actualizaciones en tiempo real**
```

2.3 Flujo de Datos

VIEW → VIEWMODEL → REPOSITORY/DAO → SUPABASE API

↑ ↓ ↓ ↓

└──────────┘└──────────┘└──────────┘

Notify Listeners / Streams

3 CONFIGURACIÓN DEL ENTORNO

3.1 Requisitos del Sistema

```
 requisitos_sistema.yaml
desarrollo:
  sistema_operativo: "Windows 11"
  ram_minima: "8 GB (recomendado 16 GB)"
  almacenamiento: "10 GB libres"
  herramientas:
    - "Git 2.30+"
    - "Flutter SDK 3.38.3+"
    - "Dart SDK 3.10.1+"
    - "Android Studio 2022+ o VS Code"
    - "Android SDK / Xcode (para compilación)"
    - "Node.js 18+ (opcional para scripts)"
```

3.2 Instalación Paso a Paso

```
1. Instalar Flutter
Descargar desde: https://flutter.dev/docs/get-started/install

2. Verificar
instalación flutter
doctor

3. Clonar repositorio
git clone https://github.com/tu-usuario/azteca-
fest.git cd azteca-fest

4. Instalar
dependencias flutter
pub get

5. Configurar variables de
entorno cp .env.example .env
Editar .env con tus credenciales

6. Ejecutar en modo
desarrollo flutter run
```

3.3 Archivo. env de Configuración

3.3.1 Guarda credenciales sensibles fuera del código

```
CONFIGURACIÓN SUPABASE
SUPABASE_URL=https://tu-proyecto.superbase.co
SUPABASE_ANON_KEY=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9...
SUPERBASE_SERVICE_KEY=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9...

CONFIGURACIÓN APP
APP_NAME="Azteca Fest"
APP_VERSION="1.0.0"
ENVIRONMENT="development"  development, staging, production
DEBUG_MODE=true
LOG_LEVEL="verbose"
```

3.3.2 CONFIGURACIÓN EMAIL/SMS

3.3.2.1 Servicio de email con Gmail SMTP para recuperación de contraseñas.

```
import 'package:mailer/mailer.dart';

import 'package:mailer/smtp_server.dart';

import 'package:flutter/foundation.dart';


/// 📧 EmailService - Servicio para enviar emails de recuperación de contraseña

///

/// Este servicio utiliza SMTP de Gmail para enviar emails de reset de contraseña.

///

/// CONFIGURACIÓN NECESARIA:

/// 1. Crear cuenta de Gmail

/// 2. Habilitar "Contraseñas de aplicación" en Google (https://myaccount.google.com/apppasswords)

/// 3. Configurar las credenciales abajo

///

/// ⚠️ IMPORTANTE: Configurar estas credenciales ANTES de usar

class EmailService {

  // 🔧 CONFIGURAR AQUÍ TUS CREDENCIALES DE GMAIL

  static const String _senderEmail = 'winstonalvarado94@gmail.com'; // ➤ Cambiar por tu email

  static const String _senderPassword = 'phfm hrlt mccp axzt'; // ➤ Contraseña de aplicación (16 caracteres)

  static const String _senderName = 'AztecaFest Sistema';

  /// Enviar email de recuperación de contraseña

  ///

  /// Parámetros:

  /// - [recipientEmail]: Email del usuario
```

```
  /// - [resetLink]: Link para resetear contraseña (incluye
token)
  /// - [userName]: Nombre del usuario
  ///
  /// Retorna: true si se envió correctamente, false si hubo
error
  Future<bool> enviarEmailRecuperacion({
    required String recipientEmail,
    required String resetLink,
    required String userName,
  }) async {
    try {
      debugPrint('📧 Intentando enviar email de recuperación
a: $recipientEmail');


      // Validar configuración
      // Solo simular si las credenciales son claramente
placeholders
      if (_senderEmail.isEmpty || _senderPassword.isEmpty ||
          _senderEmail == 'tu_email@gmail.com' ||
          _senderPassword == 'xxxx xxxx xxxx xxxx') {
        debugPrint('⚠️ ADVERTENCIA: Credenciales de email NO
configuradas');
        debugPrint('📧 Simulando envío de email a:
$recipientEmail');
        debugPrint('🔗 Link de reset: $resetLink');
        debugPrint('⚠️ Configura _senderEmail y
_senderPassword en email_service.dart');
        return true; // Retornar true para permitir flujo de
desarrollo
      }
```

```
      // Si las credenciales están configuradas pero son las mismas que los valores por defecto,
      // asumir que el usuario las configuró intencionalmente
      debugPrint('✅ Credenciales configuradas. Procediendo con envío real.');

      // Crear mensaje
      final message = Message()
        ..from = Address(_senderEmail, _senderName)
        ..recipients.add(recipientEmail)
        ..subject = '🔐 Recupera tu contraseña - AztecaFest'
        ..html = _buildHtmlEmail(userName, resetLink);

      // Configurar servidor SMTP de Gmail explícitamente
      final smtpServer = SmtpServer(
        'smtp.gmail.com',
        port: 587,
        username: _senderEmail,
        password: _senderPassword,
        ssl: false,
        allowInsecure: true,
      );

      debugPrint('🔄 Conectando a servidor SMTP de Gmail...');
      debugPrint('📤 Enviando email a: $recipientEmail');

      // Enviar email REAL
      final sendReport = await send(message, smtpServer);
```

```
      debugPrint('✅ Email enviado correctamente');
      debugPrint('📊 Reporte: ${sendReport.toString()}');
      return true;
    } catch (e, stackTrace) {
      debugPrint('❌ Error al enviar email: $e');
      debugPrint('📄 Stack trace: $stackTrace');
      debugPrint('💡 Posibles causas y soluciones:');
      debugPrint('   1. Verifica que el email y contraseña sean correctos');
      debugPrint('   2. Usa una "Contraseña de aplicación" de Google (NO tu contraseña normal)');
      debugPrint('   3. Asegúrate de que la verificación en dos pasos está activada en tu cuenta de Google');
      debugPrint('   4. Verifica tu conexión a Internet');
      debugPrint('   5. Comprueba que el puerto 587 no está bloqueado por tu firewall');

      // Verificar credenciales específicamente
      if (_senderEmail.contains('@') == false || _senderEmail.contains('.') == false) {
        debugPrint('⚠️ El email del remitente no parece válido: $_senderEmail');
      }

      if (_senderPassword.length != 19 || _senderPassword.contains(' ') == false) {
        debugPrint('⚠️ La contraseña no tiene el formato esperado de contraseña de aplicación (debe ser xxxx xxxx xxxx xxxx)');
      }

      return false;
```

```
    }
  }

  /// Construir HTML del email con estilo profesional
  static String _buildHtmlEmail(String userName, String resetLink) {
    return '''
    <!DOCTYPE html>
    <html>
      <head>
        <meta charset="UTF-8">
        <style>
          body {
            font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
            line-height: 1.6;
            color: #333;
          }
          .container {
            max-width: 600px;
            margin: 0 auto;
            background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
            padding: 20px;
            border-radius: 10px;
          }
          .content {
            background: white;
            padding: 30px;
            border-radius: 8px;
```

```
            box-shadow: 0 4px 6px rgba(0,0,0,0.1);
        }
        .header {
            text-align: center;
            margin-bottom: 30px;
        }
        .logo {
            font-size: 28px;
            font-weight: bold;
            color: #E74C3C;
            margin-bottom: 10px;
        }
        .title {
            color: #2C3E50;
            margin-bottom: 20px;
        }
        .button {
            display: inline-block;
            padding: 15px 30px;
            background: linear-gradient(135deg, #E74C3C,
#FF6B6B);
            color: white !important;
            text-decoration: none;
            border-radius: 25px;
            font-weight: bold;
            margin: 20px 0;
            box-shadow: 0 4px 15px rgba(231, 76, 60, 0.3);
        }
        .warning {
            background: #FFF3CD;
```

```
            border-left: 4px solid #FFC107;
            padding: 15px;
            margin: 20px 0;
            border-radius: 4px;
          }
          .footer {
            text-align: center;
            margin-top: 30px;
            color: #6c757d;
            font-size: 12px;
          }
        </style>
      </head>
      <body>
        <div class="container">
          <div class="content">
            <div class="header">
              <div class="logo">🧢 AztecaFest</div>
              <h1 class="title">Recupera tu contraseña</h1>
            </div>

            <p>¡Hola <strong>$userName</strong>!</p>

            <p>Hemos recibido una solicitud para restablecer
tu contraseña en AztecaFest.</p>

            <p>Para crear una nueva contraseña, haz clic en
el siguiente botón:</p>

            <div style="text-align: center;">
```

```
              <a href="$resetLink" class="button">Restablecer
Contraseña</a>
            </div>

            <p>Si no puedes hacer clic en el botón, copia y
pega este enlace en tu navegador:</p>
            <p style="word-break: break-all; color:
#007bff;">$resetLink</p>

            <div class="warning">
              <p><strong>Importante:</strong></p>
              <ul>
                <li>Este enlace expira en 24 horas</li>
                <li>No compartas este email con nadie</li>
                <li>Si no solicitaste este cambio, ignora
este email</li>
              </ul>
            </div>

            <p>Gracias por usar AztecaFest. ¡Disfruta de
nuestros deliciosos tacos!</p>

            <div class="footer">
              <p>© 2025 AztecaFest. Todos los derechos
reservados.</p>
              <p>Este email fue enviado automáticamente. No
responda a este correo.</p>
            </div>
          </div>
        </div>
      </body>
    </html>
```

```
    ''';
  }

  /// Validar email - versión simplificada
  static bool validarEmail(String email) {
    if (email.isEmpty) return false;
    // Verificar que tenga @ y . y que no estén al inicio o final
    return email.contains('@') &&
           email.contains('.') &&
           email.indexOf('@') > 0 &&
           email.lastIndexOf('.') > email.indexOf('@') + 1 &&
           email.lastIndexOf('.') < email.length - 1;
  }

  /// Enviar email genérico con asunto y cuerpo HTML personalizado
  ///
  /// Parámetros:
  /// - [destinatario]: Email del destinatario
  /// - [asunto]: Asunto del email
  /// - [cuerpo]: Cuerpo en HTML del email
  ///
  /// Retorna: true si se envió correctamente
  Future<bool> enviarEmail({
    required String destinatario,
    required String asunto,
    required String cuerpo,
  }) async {
    try {
```

```
      debugPrint('📧 Intentando enviar email a: $destinatario');
      debugPrint('Asunto: $asunto');

      // Validar configuración
      // Solo simular si las credenciales son claramente placeholders
      if (_senderEmail.isEmpty || _senderPassword.isEmpty ||
          _senderEmail == 'tu_email@gmail.com' ||
          _senderPassword == 'xxxx xxxx xxxx xxxx') {
        debugPrint('⚠️ ADVERTENCIA: Credenciales de email NO configuradas');
        debugPrint('📧 Simulando envío de email a: $destinatario');
        debugPrint('⚠️ Configura _senderEmail y _senderPassword en email_service.dart');
        return true; // Retornar true para permitir flujo de desarrollo
      }

      // Si las credenciales están configuradas pero son las mismas que los valores por defecto,
      // asumir que el usuario las configuró intencionalmente
      debugPrint('✅ Credenciales configuradas. Procediendo con envío real.');

      // Crear mensaje
      final message = Message()
        ..from = Address(_senderEmail, _senderName)
        ..recipients.add(destinatario)
        ..subject = asunto
        ..html = cuerpo;
```

```
      // Configurar servidor SMTP de Gmail explícitamente
      final smtpServer = SmtpServer(
        'smtp.gmail.com',
        port: 587,
        username: _senderEmail,
        password: _senderPassword,
        ssl: false,
        allowInsecure: true,
      );

      // Enviar email
      await send(message, smtpServer);

      debugPrint('✅ Email enviado correctamente');
      return true;
    } catch (e, stackTrace) {
      debugPrint('❌ Error al enviar email: $e');
      debugPrint('📄 Stack trace: $stackTrace');
      debugPrint('💡 Posibles causas:');
      debugPrint('   1. Verifica tu conexión a Internet');
      debugPrint('   2. Comprueba que el puerto 587 no está bloqueado por tu firewall');
      debugPrint('   3. Verifica que las credenciales sean correctas');

      // Verificar credenciales específicamente
      if (_senderEmail.contains('@') == false || _senderEmail.contains('.') == false) {
        debugPrint('⚠️ El email del remitente no parece
```

```
válido: $_senderEmail');
      }


      if (_senderPassword.length != 19 ||
_senderPassword.contains(' ') == false) {
        debugPrint('⚠ La contraseña no tiene el formato
esperado de contraseña de aplicación');
      }


      return false;
    }
  }
}
```

3.4 CONFIGURACIÓN PAGO SIMULADO

3.4.1 DAO para gestionar ventas y detalles en Supabase.

```
// lib/data/dao/venta_dao_supabase.dart

import '../../config/supabase_config.dart';
import '../models/venta.dart';
import '../models/detalle_venta.dart';

class VentaDaoSupabase {
  static final _tableVentas = SupabaseConfig.ventas;
  static final _tableDetalles =
SupabaseConfig.detalleVentas;
  static final _tableProductos = SupabaseConfig.productos;
  static final _tableUsuarios = SupabaseConfig.usuarios;

  /// Convertir datos de Supabase a Venta
  Venta _mapToVenta(Map<String, dynamic> data) {
    final venta = Venta(
      id: data['id'] as int?,
      numeroVenta: (data['numero_venta'] as String?) ?? 'V-
${data['id'] ?? 0}',
      fecha: data['fecha'] != null
          ? DateTime.parse(data['fecha'] as String)
          : DateTime.now(),
```

```
      clienteId: data['cliente_id'] as int?,
      vendedorId: (data['vendedor_id'] as int?) ?? 0,
      subtotal: (data['subtotal'] as num?)?.toDouble() ??
0.0,
      descuento: (data['descuento'] as num?)?.toDouble() ??
0.0,
      impuesto: (data['impuesto'] as num?)?.toDouble() ??
0.0,
      total: (data['total'] as num?)?.toDouble() ?? 0.0,
      metodoPago: (data['metodo_pago'] as String?) ?? '',
      estado: (data['estado'] as String?) ?? 'completada',
      observaciones: data['observaciones'] as String?,
      nombreCliente: data['nombre_cliente'] as String?,
      nombreVendedor: data['nombre_vendedor'] as String?,
    );

    return venta;
  }

  /// Convertir datos de Supabase a DetalleVenta
  DetalleVenta _mapToDetalleVenta(Map<String, dynamic> data)
{
    final subtotal = (data['subtotal'] as num?)?.toDouble()
?? 0.0;
    final descuento = (data['descuento'] as
num?)?.toDouble() ?? 0.0;
    final total = subtotal - descuento;

    final detalle = DetalleVenta(
      id: data['id'] as int?,
      ventaId: (data['venta_id'] as int?) ?? 0,
      productoId: (data['producto_id'] as int?) ?? 0,
      cantidad: (data['cantidad'] as int?) ?? 0,
      precioUnitario: (data['precio_unitario'] as
num?)?.toDouble() ?? 0.0,
      subtotal: subtotal,
      descuento: descuento,
      total: total,
    );

    return detalle;
  }
```

```
  /// Obtener todas las ventas
  Future<List<Venta>> obtenerTodas() async {
    try {
      final response = await _tableVentas
          .select('*')
          .order('fecha', ascending: false);

      final ventas = (response as List).map((json) =>
_mapToVenta(json)).toList();

      // Cargar nombres de usuarios por separado
      for (var venta in ventas) {
        await _cargarNombresUsuarios(venta);
      }

      return ventas;
    } catch (e) {
      print('[v0] Error al obtener ventas: $e');
      return [];
    }
  }

  Future<void> _cargarNombresUsuarios(Venta venta) async {
    try {
      // Solo cargar si no tiene nombre ya
      if (venta.clienteId != null && (venta.nombreCliente ==
null || venta.nombreCliente!.isEmpty)) {
        try {
          final clienteData = await _tableUsuarios
              .select('nombre')
              .eq('id', venta.clienteId!)
              .maybeSingle();
          if (clienteData != null) {
            venta.nombreCliente = clienteData['nombre'] as
String?;
          }
        } catch (e) {
          // Ignorar error
        }
      }

      if (venta.vendedorId != null && venta.vendedorId! > 0
&&
```

```
          (venta.nombreVendedor == null ||
venta.nombreVendedor!.isEmpty)) {
        try {
          final vendedorData = await _tableUsuarios
              .select('nombre')
              .eq('id', venta.vendedorId!)
              .maybeSingle();
          if (vendedorData != null) {
            venta.nombreVendedor = vendedorData['nombre'] as
String?;
          }
        } catch (e) {
          // Ignorar error
        }
      }
    } catch (e) {
      // Ignorar errores de carga de nombres
    }
  }

  /// Obtener venta por ID con detalles
  Future<Venta?> obtenerPorId(int id) async {
    try {
      final response = await _tableVentas
          .select('*')
          .eq('id', id)
          .maybeSingle();

      if (response == null) return null;

      final venta = _mapToVenta(response);
      await _cargarNombresUsuarios(venta);
      venta.detalles = await obtenerDetalles(id);

      return venta;
    } catch (e) {
      print('[v0] Error al obtener venta por ID: $e');
      return null;
    }
  }

  /// Obtener detalles de una venta
  Future<List<DetalleVenta>> obtenerDetalles(int ventaId)
```

```
async {
    try {
      final response = await _tableDetalles
          .select('*')
          .eq('venta_id', ventaId)
          .order('id', ascending: true);

      final detalles = (response as List).map((json) =>
_mapToDetalleVenta(json)).toList();

      // Cargar nombres de productos
      for (var detalle in detalles) {
        try {
          final productoData = await _tableProductos
              .select('nombre, codigo')
              .eq('id', detalle.productoId)
              .maybeSingle();
          if (productoData != null) {
            detalle.nombreProducto = productoData['nombre']
as String?;
            detalle.codigoProducto = productoData['codigo']
as String?;
          }
        } catch (e) {
          // Ignorar error
        }
      }

      return detalles;
    } catch (e) {
      print('[v0] Error al obtener detalles de venta: $e');
      return [];
    }
  }

  /// Generar número de venta único
  Future<String> generarNumeroVenta() async {
    try {
      final ahora = DateTime.now();
      final fecha =
'${ahora.year}${ahora.month.toString().padLeft(2,
'0')}${ahora.day.toString().padLeft(2, '0')}';
```

```
      final inicioHoy = DateTime(ahora.year, ahora.month,
ahora.day);
      final finHoy = DateTime(ahora.year, ahora.month,
ahora.day, 23, 59, 59);
      final ventasHoy = await obtenerPorFechas(inicioHoy,
finHoy);

      final numeroSecuencial = (ventasHoy.length +
1).toString().padLeft(4, '0');

      return 'V-$fecha-$numeroSecuencial';
    } catch (e) {
      return 'V-${DateTime.now().millisecondsSinceEpoch}';
    }
  }

  /// Insertar nueva venta con sus detalles
  /// Simplified insert without foreign key checks
  Future<int?> insertar(Venta venta) async {
    try {
      final numeroVenta = venta.numeroVenta.isNotEmpty
          ? venta.numeroVenta
          : await generarNumeroVenta();

      print('[v0] Insertando venta: $numeroVenta');
      print('[v0] Total: ${venta.total}, Detalles:
${venta.detalles.length}');

      final Map<String, dynamic> ventaData = {
        'numero_venta': numeroVenta,
        'fecha': venta.fecha.toIso8601String(),
        'subtotal': venta.subtotal,
        'descuento': venta.descuento,
        'impuesto': venta.impuesto,
        'total': venta.total,
        'metodo_pago': venta.metodoPago,
        'estado': venta.estado,
        'observaciones': venta.observaciones,
        'nombre_cliente': venta.nombreCliente ?? 'Cliente',
        'nombre_vendedor': venta.nombreVendedor ??
'Autoservicio',
      };
```

```
      if (venta.clienteId != null && venta.clienteId! > 0) {
        try {
          final clienteExiste = await _tableUsuarios
              .select('id')
              .eq('id', venta.clienteId!)
              .maybeSingle();
          if (clienteExiste != null) {
            ventaData['cliente_id'] = venta.clienteId;
            print('[v0] Cliente verificado:
${venta.clienteId}');
          } else {
            print('[v0] Cliente ID ${venta.clienteId} no
existe en tabla usuarios');
            // No agregar cliente_id, solo guardar el nombre
          }
        } catch (e) {
          print('[v0] Error verificando cliente: $e');
          // Continuar sin cliente_id
        }
      }

      if (venta.vendedorId != null && venta.vendedorId! > 0)
{
        try {
          final vendedorExiste = await _tableUsuarios
              .select('id')
              .eq('id', venta.vendedorId!)
              .maybeSingle();
          if (vendedorExiste != null) {
            ventaData['vendedor_id'] = venta.vendedorId;
            print('[v0] Vendedor verificado:
${venta.vendedorId}');
          } else {
            print('[v0] Vendedor ID ${venta.vendedorId} no
existe en tabla usuarios');
          }
        } catch (e) {
          print('[v0] Error verificando vendedor: $e');
        }
      }

      print('[v0] Datos de venta a insertar: $ventaData');
```

```
      final responseVenta = await _tableVentas
          .insert(ventaData)
          .select('id')
          .single();

      final ventaId = responseVenta['id'] as int;
      print('[v0] Venta insertada con ID: $ventaId');

      // Insertar los detalles
      for (var detalle in venta.detalles) {
        print('[v0] Insertando detalle: producto
${detalle.productoId}, cantidad ${detalle.cantidad}');

        try {
          final productoExiste = await _tableProductos
              .select('id, stock')
              .eq('id', detalle.productoId)
              .maybeSingle();

          if (productoExiste != null) {
            await _tableDetalles.insert({
              'venta_id': ventaId,
              'producto_id': detalle.productoId,
              'cantidad': detalle.cantidad,
              'precio_unitario': detalle.precioUnitario,
              'subtotal': detalle.subtotal,
              'descuento': detalle.descuento,
              'total': detalle.total, // Campo requerido que
faltaba
            });

            // Actualizar stock
            final stockActual = productoExiste['stock'] as
int;
            final nuevoStock = stockActual -
detalle.cantidad;

            await _tableProductos
                .update({'stock': nuevoStock > 0 ?
nuevoStock : 0})
                .eq('id', detalle.productoId);

            print('[v0] Stock actualizado: $stockActual ->
```

```
$nuevoStock');
          } else {
            print('[v0] ADVERTENCIA: Producto
${detalle.productoId} no encontrado');
          }
        } catch (e) {
          print('[v0] Error procesando detalle: $e');
          // Continuar con los demás detalles
        }
      }

      print('[v0] Venta creada exitosamente con ID:
$ventaId');
      return ventaId;
    } catch (e) {
      print('[v0] ERROR al insertar venta: $e');
      return null;
    }
  }

  /// Actualizar venta existente
  Future<bool> actualizar(Venta venta) async {
    try {
      await _tableVentas
          .update({
            'numero_venta': venta.numeroVenta,
            'fecha': venta.fecha.toIso8601String(),
            'subtotal': venta.subtotal,
            'descuento': venta.descuento,
            'impuesto': venta.impuesto,
            'total': venta.total,
            'metodo_pago': venta.metodoPago,
            'estado': venta.estado,
            'observaciones': venta.observaciones,
            'nombre_cliente': venta.nombreCliente,
            'nombre_vendedor': venta.nombreVendedor,
          })
          .eq('id', venta.id!);

      return true;
    } catch (e) {
      print('[v0] Error al actualizar venta: $e');
      return false;
```

```
    }
  }

  /// Eliminar venta y sus detalles
  Future<bool> eliminar(int id) async {
    try {
      await _tableDetalles.delete().eq('venta_id', id);
      await _tableVentas.delete().eq('id', id);
      return true;
    } catch (e) {
      print('[v0] Error al eliminar venta: $e');
      return false;
    }
  }

  /// Obtener ventas por cliente
  Future<List<Venta>> obtenerPorCliente(int clienteId) async
{
    try {
      final response = await _tableVentas
          .select('*')
          .eq('cliente_id', clienteId)
          .order('fecha', ascending: false);

      return (response as List).map((json) =>
_mapToVenta(json)).toList();
    } catch (e) {
      print('[v0] Error al obtener ventas por cliente: $e');
      return [];
    }
  }

  /// Obtener ventas por vendedor
  Future<List<Venta>> obtenerPorVendedor(int vendedorId)
async {
    try {
      final response = await _tableVentas
          .select('*')
          .eq('vendedor_id', vendedorId)
          .order('fecha', ascending: false);

      return (response as List).map((json) =>
_mapToVenta(json)).toList();
```

```
    } catch (e) {
      print('[v0] Error al obtener ventas por vendedor:
$e');
      return [];
    }
  }

  /// Obtener ventas por rango de fechas
  Future<List<Venta>> obtenerPorFechas(DateTime inicio,
DateTime fin) async {
    try {
      final response = await _tableVentas
          .select('*')
          .gte('fecha', inicio.toIso8601String())
          .lte('fecha', fin.toIso8601String())
          .order('fecha', ascending: false);

      return (response as List).map((json) =>
_mapToVenta(json)).toList();
    } catch (e) {
      print('[v0] Error al obtener ventas por fechas: $e');
      return [];
    }
  }

  /// Obtener total de ventas
  Future<double> obtenerTotalVentas() async {
    try {
      final response = await _tableVentas.select('total');

      double total = 0;
      for (var venta in response as List) {
        total += (venta['total'] as num).toDouble();
      }

      return total;
    } catch (e) {
      return 0;
    }
  }

  /// Obtener estadísticas de ventas
  Future<Map<String, dynamic>> obtenerEstadisticas() async {
```

```
    try {
      final response = await _tableVentas.select('total,
fecha');

      double totalGeneral = 0;
      int cantidadVentas = 0;

      for (var venta in response as List) {
        totalGeneral += (venta['total'] as num).toDouble();
        cantidadVentas++;
      }

      final promedio = cantidadVentas > 0 ? totalGeneral /
cantidadVentas : 0;

      return {
        'total': totalGeneral,
        'cantidad': cantidadVentas,
        'promedio': promedio,
      };
    } catch (e) {
      return {'total': 0.0, 'cantidad': 0, 'promedio': 0.0};
    }
  }
}
```

4 ESTRUCTURA DEL PROYECTO

En este apartado se utilizo la arquitectura MVVM , con el cual esta hecho esta app móvil "Azteca Fest" basándonos con el clean architecture que nos indica la ubicación exacta de las subcarpetas de lib.

4.1 Descripción Completa de Directorios

main.dart # Punto de entrada

│ **provider_setup.dart** # Configuración de providers

│

├──── **assets** # Recursos estáticos

│ └──── **images**

favicon.ico
logo.png
config # Configuración de Supabase

```
│       supabase_config.dart
│
├──────core    # Núcleo (rutas, temas, constantes)
│       app_routes.dart
│       app_theme.dart
│       constants.dart
│       responsive_helper.dart
│
├──────data    # Capa de datos (DAOs, modelos)
│   │   database_helper.dart
│   │
│   ├──────dao
│   │       cliente_dao.dart
│   │       detalle_pedido.dart
│   │       pedido_dao.dart
│   │       pedido_dao_supabase.dart
│   │       producto_dao.dart
│   │       producto_dao_supabase.dart
│   │       usuario_dao.dart
│   │       usuario_dao_supabase.dart
│   │       venta_dao.dart
│   │       venta_dao_supabase.dart
│   │
```

```
│   └──────models
│       cliente.dart
│       detalle_pedido.dart
│       detalle_venta.dart
│       pedido.dart
│       pedido_estado.dart
│       persona_base.dart
│       producto.dart
│       usuario.dart
│       venta.dart
│
├──────examples
│   migration_example.dart
│
├──────screens # Pantallas individuales
│   crear_pedido_screen.dart
│   pedido_detalle_screen.dart
│   producto_detalle_screen.dart
│   usuario_detalle_screen.dart
│   usuario_form_screen.dart
│
├──────services # Servicios (auth, carrito, email, storage)
│   auth_service.dart
```

carrito_service.dart

email_service.dart

image_storage_service.dart

migration_service.dart

password_recovery_service.dart

permission_service.dart

report_service.dart

session_timeout_service.dart

supabase_storage_service.dart

utils # Utilidades (validadores, formateadores)

formatters.dart

responsive_helper.dart

responsive_util.dart

validators.dart

viewmodels # ViewModels (MVVM)

cliente_viewmodel.dart

pedido_viewmodel.dart

producto_viewmodel.dart

usuario_viewmodel.dart

venta_viewmodel.dart

```
└──────views  # Vistas organizadas por rol
  ├──────admin
  │       admin_dashboard.dart
  │       admin_migration_screen.dart
  │       admin_pedidos_tab.dart
  │       admin_productos_tab.dart
  │       admin_reportes_tab.dart
  │       admin_usuarios_tab.dart
  │       admin_ventas_tab.dart
  │
  ├──────auth
  │       forgot_password_screen.dart
  │       home.dart
  │       login_screen.dart
  │       register_screen.dart
  │       reset_password_screen.dart
  │
  ├──────cliente
  │       carrito_screen.dart
  │       catalogo_screen.dart
  │       cliente_dashboard.dart
  │       crear_pedido_cliente_screen.dart
  │       mis_pedidos_tab.dart
```

productos

producto_detalle_screen.dart

producto_form_dialog.dart

vendedor

vendedor_clientes_tab.dart

vendedor_crear_venta.dart

vendedor_dashboard.dart

vendedor_pedidos_tab.dart

vendedor_productos_tab.dart

vendedor_ventas_tab.dart

widgets

animated_add_button.dart

cart_badge.dart

custom_button.dart

custom_input.dart

list_item.dart

producto_card.dart

product_feedback_overlay.dart

responsive_widgets.dart

scrollable_list.dart

└──────shared

configuracion_clientescreen.dart

configuracion_usuarioscreen.dart

perfil_cliente_screen.dart

perfil_usuario_screen.dart

4.2 Archivos Principales Detallados

4.2.1 lib/main.dart` Punto de entrada de la app, configuración de rutas, temas y providers.

```
import 'package:flutter/material.dart';
import 'package:flutter_localizations/flutter_localizations.dart';
import 'package:provider/provider.dart';
import 'package:flutter/foundation.dart'
    show kIsWeb, defaultTargetPlatform,
TargetPlatform; import
'package:sqflite_common_ffi/sqflite_ffi.dart';
import 'package:sqflite_common_ffi_web/sqflite_ffi_web.dart';

// ======================================
// CONFIGURACIÓN Y MODELOS
// ======================================
import
'config/supabase_config.dart';
import 'data/models/usuario.dart';
import 'core/app_theme.dart';
import 'core/app_routes.dart';
import 'provider_setup.dart';

// ======================================
// VIEWMODELS
// ======================================
import 'viewmodels/usuario_viewmodel.dart';
import 'viewmodels/producto_viewmodel.dart';
import 'viewmodels/venta_viewmodel.dart';
import 'viewmodels/pedido_viewmodel.dart';
```

```
import 'viewmodels/cliente_viewmodel.dart';
```

```dart
// ======================================
// IMPORTS ESENCIALES (se cargan al inicio)
// ======================================
import 'views/auth/home.dart';
import 'views/auth/login_screen.dart';

// ======================================
// IMPORTS DIFERIDOS (se cargan cuando se necesitan)
// ======================================
import 'views/auth/register_screen.dart' deferred as
register; import 'views/auth/forgot_password_screen.dart'
deferred as forgot_password;
import 'views/auth/reset_password_screen.dart' deferred as reset_password;
import 'views/admin/admin_dashboard.dart' deferred as admin;
import 'views/vendedor/vendedor_dashboard.dart' deferred as
vendedor; import 'views/cliente/cliente_dashboard.dart' deferred
as cliente; import 'views/cliente/catalogo_screen.dart' deferred
as catalogo; import
'views/widgets/shared/perfil_usuario_screen.dart' deferred as
perfil;
import 'views/widgets/shared/perfil_cliente_screen.dart'
    deferred as perfil_cliente;
import
    'views/widgets/shared/configuracion_usuarioscreen.dart'
    deferred as config;

void main() async {
  // Asegurar inicialización de Flutter
  WidgetsFlutterBinding.ensureInitialized();

  print('🚀 Iniciando AztecaFest...');

  // ======================================
  // INICIALIZAR SUPABASE
  // ======================================
  print('🔄 Inicializando
  Supabase...'); try {
    await SupabaseConfig.initialize();
    print('✅ Supabase inicializado correctamente');

    // Verificar conexión
    final conexionExitosa = await
    SupabaseConfig.verificarConexion(); if (conexionExitosa) {
      print('✅ Conectado a Supabase exitosamente');
      print('🔑 URL: ${SupabaseConfig.supabaseUrl}');
    } else {
```

```
      print('⚠ Advertencia: No se pudo verificar la conexión a Supabase');
    }
  } catch (e) {
    print('❌ Error al inicializar Supabase: $e');
    print('⚠ La app continuará pero puede tener problemas de
conectividad');
  }

  // ======================================
  // INICIALIZAR BASE DE DATOS LOCAL
  // ======================================
  print('🔄 Configurando base de datos
  local...'); if (kIsWeb) {
    databaseFactory = databaseFactoryFfiWeb;
    print('✅ Base de datos web configurada');
  } else if (defaultTargetPlatform == TargetPlatform.windows ||
      defaultTargetPlatform == TargetPlatform.linux ||
      defaultTargetPlatform == TargetPlatform.macOS) {
    sqfliteFfiInit();
    databaseFactory = databaseFactoryFfi;
    print('✅ Base de datos desktop
    configurada');
  } else {
    print('✅ Base de datos móvil configurada');
  }

  print('🎉 AztecaFest iniciada correctamente');

  // Ejecutar la app
  runApp(const
  AztecaFestApp());
}

// ======================================
// APLICACIÓN PRINCIPAL
// ======================================
class AztecaFestApp extends StatelessWidget {
  const AztecaFestApp({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return MultiProvider(
      providers: [
        // Providers del sistema (ProviderSetup)
        ...ProviderSetup.allProviders,
```

```
    // ViewModels adicionales
    ChangeNotifierProvider(
      create: (_) => UsuarioViewModel(),
    ),
    ChangeNotifierProvider(
      create: (_) => ProductoViewModel(),
    ),
    ChangeNotifierProvider(
      create: (_) => VentaViewModel(),
    ),
    ChangeNotifierProvider(
      create: (_) => PedidoViewModel(),
    ),
    ChangeNotifierProvider(
      create: (_) => ClienteViewModel(),
    ),
  ],
  child: MaterialApp(
    title: 'AztecaFest',
    debugShowCheckedModeBanner: false,
    theme: AppTheme.temaOscuro, // Tema oscuro por defecto
    initialRoute: AppRoutes.home,

    // ========================================
    // LOCALIZACIONES (Soporte español)
    // ========================================
    localizationsDelegates: const [
      GlobalMaterialLocalizations.delegate,
      GlobalWidgetsLocalizations.delegate,
      GlobalCupertinoLocalizations.delegate,
    ],
    supportedLocales: const [
      Locale('es', 'ES'),
      Locale('en', 'US'),
    ],

    // ========================================
    // RUTAS ESTÁTICAS (solo las esenciales)
    // ========================================
    routes: {
      AppRoutes.home: (context) => const HomeScreen(),
      AppRoutes.login: (context) => const LoginScreen(),
    },

    // ========================================
```

```
      // RUTAS DINÁMICAS CON CARGA DIFERIDA
      // ======================================
      onGenerateRoute: (settings) {
        return _handleRoute(settings);
      },

      // ======================================
      // RUTA NO ENCONTRADA (404)
      // ======================================
      onUnknownRoute: (settings) {
        return MaterialPageRoute(
          builder: (context) => ErrorScreen(routeName: settings.name),
        );
      },
    ),
  );
}

// ======================================
// MANEJADOR DE RUTAS CON CARGA DIFERIDA
// ======================================
static Route<dynamic>? _handleRoute(RouteSettings settings)
  { final usuario = settings.arguments as Usuario?;

  // Pantalla de carga mientras se carga el módulo
  Widget buildLoadingScreen(String moduleName) {
    return Scaffold(
      backgroundColor: const Color(0xFF1A1A1A),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            const CircularProgressIndicator(
              color: Color(0xFFFF073A),
            ),
            const SizedBox(height:
            24), Text(
              'Cargando $moduleName...',
              style: const TextStyle(
                color: Colors.white70,
                fontSize: 16,
              ),
            ),
          ],
        ),
```

```
    ),
  );
}

try {
  // ======================================
  // REGISTRO
  // ======================================
  if (settings.name == AppRoutes.registro) {
    return MaterialPageRoute(
      builder: (context) => FutureBuilder(
        future: register.loadLibrary(),
        builder: (context, snapshot) {
          if (snapshot.connectionState == ConnectionState.done)
            { return register.RegisterScreen();
          }
          return buildLoadingScreen('Registro');
        },
      ),
    );
  }

  // ======================================
  // OLVIDÉ MI CONTRASEÑA
  // ======================================
  if (settings.name == AppRoutes.forgotPassword)
    { return MaterialPageRoute(
      builder: (context) => FutureBuilder(
        future:
        forgot_password.loadLibrary(),
        builder: (context, snapshot) {
          if (snapshot.connectionState == ConnectionState.done)
            { return forgot_password.ForgotPasswordScreen();
          }
          return buildLoadingScreen('Recuperación de Contraseña');
        },
      ),
    );
  }

  // ======================================
  // RESETEAR CONTRASEÑA
  // ======================================
  if (settings.name == AppRoutes.resetPassword) {
    String? token = settings.arguments as String?;
```

```
        // Extraer token de parámetros de URI para deep
        links if (token == null) {
          // Para web
          if (kIsWeb)
          {
            token = Uri.base.queryParameters['token'];
          }
          // Para móvil - intentar extraer de la
          ruta else {
            if (settings.name != null &&
settings.name!.contains('?token=')) {
              try {
                final uri = Uri.parse(settings.name!);
                token = uri.queryParameters['token'];
              } catch (e) {
                print('Error parsing URI: $e');
              }
            }
          }
        }

        if (token == null) {
          return
          MaterialPageRoute(
            builder: (context) => const Scaffold(
              body: Center(
                child: Text(
                  'Token no válido',
                  style: TextStyle(color: Colors.white),
                ),
              ),
            ),
          );
        }

        return MaterialPageRoute(
          builder: (context) => FutureBuilder(
            future:
            reset_password.loadLibrary(),
            builder: (context, snapshot) {
              if (snapshot.connectionState == ConnectionState.done) {
                return reset_password.ResetPasswordScreen(token:
                token!);
              }
              return buildLoadingScreen('Resetear Contraseña');
            },
          ),
        );
```

```
}

// ======================================
// CATÁLOGO
// ======================================
if (settings.name == AppRoutes.catalogo) {
  return MaterialPageRoute(
    builder: (context) => FutureBuilder(
      future: catalogo.loadLibrary(),
      builder: (context, snapshot) {
        if (snapshot.connectionState == ConnectionState.done)
          { return catalogo.CatalogoScreen();
        }
        return buildLoadingScreen('Catálogo');
      },
    ),
  );
}

// ======================================
// ADMIN DASHBOARD
// ======================================
if (settings.name == AppRoutes.adminDashboard)
  { if (usuario == null) return
  _routeToLogin();

  return MaterialPageRoute(
    builder: (context) =>
      FutureBuilder( future:
      admin.loadLibrary(), builder:
      (context, snapshot) {
        if (snapshot.connectionState == ConnectionState.done)
          { return admin.AdminDashboard(usuario: usuario);
        }
        return buildLoadingScreen('Panel de Administrador');
      },
    ),
  );
}

// ======================================
// VENDEDOR DASHBOARD
// ======================================
if (settings.name == AppRoutes.vendedorDashboard)
  { if (usuario == null) return _routeToLogin();

  return MaterialPageRoute(
    builder: (context) => FutureBuilder(
```

```
      future: vendedor.loadLibrary(),
      builder: (context, snapshot) {
        if (snapshot.connectionState == ConnectionState.done)
          { return vendedor.VendedorDashboard(vendedor:
          usuario);
        }
        return buildLoadingScreen('Panel de Vendedor');
      },
    ),
  );
}

// ======================================
// CLIENTE DASHBOARD
// ======================================
if (settings.name == AppRoutes.clienteDashboard)
  { if (usuario == null) return _routeToLogin();

  return MaterialPageRoute(
    builder: (context) => FutureBuilder(
      future: cliente.loadLibrary(),
      builder: (context, snapshot) {
        if (snapshot.connectionState == ConnectionState.done)
          { return cliente.ClienteDashboard(cliente:
          usuario);
        }
        return buildLoadingScreen('Panel de Cliente');
      },
    ),
  );
}

// ======================================
// PERFIL USUARIO
// ======================================
if (settings.name == AppRoutes.perfilUsuario)
  { if (usuario == null) return
  _routeToLogin();

  return MaterialPageRoute(
    builder: (context) => FutureBuilder(
      future: perfil.loadLibrary(),
      builder: (context, snapshot) {
        if (snapshot.connectionState == ConnectionState.done)
          { return perfil.PerfilUsuarioScreen(usuario:
          usuario);
        }
        return buildLoadingScreen('Perfil');
```

```
            },
          ),
        );
      }

      // ======================================
      // PERFIL CLIENTE
      // ======================================
      if (settings.name == AppRoutes.perfilCliente)
        { if (usuario == null) return
        _routeToLogin();

        return MaterialPageRoute(
          builder: (context) => FutureBuilder(
            future:
            perfil_cliente.loadLibrary(),
            builder: (context, snapshot) {
              if (snapshot.connectionState == ConnectionState.done)
                { return
                perfil_cliente.PerfilClienteScreen(cliente:
usuario)
;             }
              return buildLoadingScreen('Perfil');
            },
          ),
        );
      }

      // ======================================
      // CONFIGURACIÓN USUARIOS
      // ======================================
      if (settings.name == AppRoutes.configuracionUsuarios)
        { if (usuario == null) return _routeToLogin();

        return MaterialPageRoute(
          builder: (context) => FutureBuilder(
            future: config.loadLibrary(),
            builder: (context, snapshot) {
              if (snapshot.connectionState == ConnectionState.done)
                { return
                config.ConfiguracionUsuarioScreen(usuario:
usuario)
;             }
              return buildLoadingScreen('Configuración');
            },
          ),
```

```
      );
    }
    // ======================================

    // REPORTES (Placeholders)
    // ======================================
    if (settings.name == AppRoutes.reporteVentas)
      { return MaterialPageRoute(
        builder: (context) =>
            const PlaceholderScreen(titulo: 'Reporte de Ventas'),
      );
    }

    if (settings.name == AppRoutes.reporteInventario) {
      return MaterialPageRoute(
        builder: (context) =>
            const PlaceholderScreen(titulo: 'Reporte de Inventario'),
      );
    }

    if (settings.name == AppRoutes.configuracion)
      { return MaterialPageRoute(
        builder: (context) =>
            const PlaceholderScreen(titulo: 'Configuración'),
      );
    }
  } catch (e) {
    print('❌ Error al cargar módulo: $e');
    return MaterialPageRoute(
      builder: (context) =>
        ErrorScreen( routeName:
        settings.name, error:
        e.toString(),
      ),
    );
  }

  return null;
}

// ======================================
// HELPER - Redireccionar al login
// ======================================
static MaterialPageRoute _routeToLogin() {
  return MaterialPageRoute(
    builder: (context) => const LoginScreen(),
  );
}
```

```
}
// ========================================
// ERROR SCREEN
// ========================================
class ErrorScreen extends StatelessWidget {
  final String? routeName;
  final String? error;

  const ErrorScreen({
    Key? key,
    this.routeName,
    this.error,
  }) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: const
      Color(0xFF1A1A1A), appBar: AppBar(
        title: const Text('Error'),
        backgroundColor: const
        Color(0xFFFF073A), elevation: 0,
      ),
      body: Center(
        child:
        Padding(
          padding: const EdgeInsets.all(24.0),
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              Container(
                padding: const EdgeInsets.all(20),
                decoration: BoxDecoration(
                  color: const Color(0xFFFF073A).withOpacity(0.2),
                  shape: BoxShape.circle,
                  border: Border.all(
                    color: const Color(0xFFFF073A),
                    width: 2,
                  ),
                ),
                child: const Icon(
                  Icons.error_outline_rounded,
                  size: 64,
                  color: Color(0xFFFF073A),
                ),
              ),
              const SizedBox(height: 24),
```

```
const Text(
  'Ruta no encontrada',
  textAlign: TextAlign.center,
  style: TextStyle(
    fontSize: 24,
    fontWeight: FontWeight.bold,
    color: Colors.white,
  ),
),
if (routeName != null) ...[
  const SizedBox(height: 12),
  Text(
    routeName!,
    textAlign: TextAlign.center,
    style: TextStyle(
      fontSize: 16,
      color: Colors.grey[400],
    ),
  ),
],
if (error != null) ...[
  const SizedBox(height: 12),
  Container(
    padding: const EdgeInsets.all(16),
    decoration: BoxDecoration(
      color: const Color(0xFF2A2A2A),
      borderRadius: BorderRadius.circular(12),
      border: Border.all(
        color: Colors.white.withOpacity(0.1),
      ),
    ),
    child: Text(
      error!,
      textAlign:
      TextAlign.center, style:
      TextStyle(
        fontSize: 14,
        color: Colors.grey[500],
        fontFamily: 'monospace',
      ),
    ),
  ),
],
const SizedBox(height:
32), ElevatedButton.icon(
  onPressed: () {
```

```
                  Navigator.of(context).pushNamedAndRemoveUntil(
                    AppRoutes.home,
                    (route) => false,
                  );
                },
                icon: const
                Icon(Icons.home_rounded), label:
                const Text('Ir al Inicio'), style:
                ElevatedButton.styleFrom(
                  backgroundColor: const Color(0xFFFF073A),
                  foregroundColor: Colors.white,
                  padding: const EdgeInsets.symmetric(
                    horizontal: 32,
                    vertical: 16,
                  ),
                  shape: RoundedRectangleBorder(
                    borderRadius:
                    BorderRadius.circular(12),
                  ),
                ),
              ),
            ],
          ),
        ),
      ),
    );
  }
}

// ========================================
// PLACEHOLDER SCREEN
// ========================================
class PlaceholderScreen extends StatelessWidget {
  final String titulo;

  const PlaceholderScreen({Key? key, required this.titulo}) :
super(key: key);

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: const
      Color(0xFF1A1A1A), appBar: AppBar(
        title: Text(titulo),
        backgroundColor: const Color(0xFF2A2A2A),
        elevation: 0,
      ),
```

```
body: Center(
  child: Column(
    mainAxisAlignment: MainAxisAlignment.center,
    children: [
      Container(
        padding: const EdgeInsets.all(20),
        decoration: BoxDecoration(
          color: const Color(0xFFF39C12).withOpacity(0.2),
          shape: BoxShape.circle,
          border: Border.all(
            color: const Color(0xFFF39C12),
            width: 2,
          ),
        ),
        child: const Icon(
          Icons.construction_rounded,
          size: 64,
          color: Color(0xFFF39C12),
        ),
      ),
      const SizedBox(height: 24),
      const Text(
        'Pantalla en
        construcción', style:
        TextStyle(
          fontSize: 20,
          fontWeight: FontWeight.bold,
          color: Colors.white,
        ),
      ),
      const SizedBox(height: 8),
      Text(
        titulo,
        style: TextStyle(
          fontSize: 16,
          color: Colors.grey[400],
        ),
      ),
      const SizedBox(height: 32),
      ElevatedButton.icon(
        onPressed: () =>
        Navigator.of(context).pop(), icon: const
        Icon(Icons.arrow_back_rounded), label: const
        Text('Volver'),
        style: ElevatedButton.styleFrom(
          backgroundColor: const
          Color(0xFFF39C12), foregroundColor:
```

```
                Colors.white,
                padding: const EdgeInsets.symmetric(
                  horizontal: 24,
                  vertical: 12,
                ),
                shape: RoundedRectangleBorder(
                  borderRadius:
                  BorderRadius.circular(12),
                ),
              ),
            ),
          ),
        ],
      ),
    ),
  );
 }
}
```

4.2.2 lib/provider_setup.dart` - Configuración Providers: Configuración de providers para inyección de dependencias.

```
import 'package:provider/provider.dart';
import 'package:provider/single_child_widget.dart';
import 'viewmodels/usuario_viewmodel.dart';
import
'viewmodels/producto_viewmodel.dart';
import 'viewmodels/venta_viewmodel.dart';
import 'viewmodels/pedido_viewmodel.dart';
import
'viewmodels/cliente_viewmodel.dart';
import 'services/carrito_service.dart';
import 'services/auth_service.dart';
import 'services/report_service.dart';
import 'services/password_recovery_service.dart';

class ProviderSetup {
  // ✅ CAMBIAR el tipo a SingleChildWidget
  static List<SingleChildWidget> providers = [
    ChangeNotifierProvider<UsuarioViewModel>(
      create: (_) => UsuarioViewModel(),
    ),
    ChangeNotifierProvider<ProductoViewModel>(
      create: (_) => ProductoViewModel(),
    ),
    ChangeNotifierProvider<VentaViewModel>(
      create: (_) => VentaViewModel(),
    ),
```

```
    ChangeNotifierProvider<PedidoViewModel
       >( create: (_) => PedidoViewModel(),
    ),
    ChangeNotifierProvider<ClienteViewModel>(
      create: (_) => ClienteViewModel(),
    ),
    ChangeNotifierProvider<CarritoService>
      ( create: (_) => CarritoService(),
    ),
    ChangeNotifierProvider<AuthService>(
      create: (_) => AuthService(),
    ),
    ChangeNotifierProvider<PasswordRecoveryService>(
      create: (_) => PasswordRecoveryService(),
    ),
  ];

  // ✅ CAMBIAR el tipo a SingleChildWidget
  static List<SingleChildWidget> singletonProviders = [
    Provider<ReportService>(
      create: (_) => ReportService(),
    ),
  ];

  // Método para obtener TODOS los providers
  combinados static List<SingleChildWidget> get
  allProviders => [
    ...providers,
    ...singletonProviders,
  ];
```

4.2.3 lib/core/constants.dart: Constantes globales (colores, roles, rutas, mensajes).

```
import 'package:flutter/material.dart';

class AppConstants {
  // ======================================
  // COLORES PRINCIPALES (Estilo Admin Dashboard)
  // ======================================
  // Gradiente principal
  static const Color colorPrimario = Color(0xFFFF6B6B);
  static const Color colorSecundario =
  Color(0xFFFF8E53);
```

```
  // Colores de estado
  static const Color colorExito = Color(0xFF4CAF50);
  static const Color colorError = Color(0xFFF44336);
  static const Color colorAdvertencia =
  Color(0xFFFF9800); static const Color colorInfo =
  Color(0xFF2196F3);

  // ======================================
  // COLORES DE FONDO (Dark Theme)
  // ======================================
  static const Color fondoOscuro = Color(0xFF0F0F0F);      //
Fondo principal oscuro
  static const Color fondoMedio = Color(0xFF1A1A1A);       //
AppBar, cards principales
  static const Color fondoTarjeta = Color(0xFF2A2A2A);     // Tarjetas
y contenedores
  static const Color fondoInput = Color(0xFF1E1E1E);       // Campos
de texto

  // Compatibilidad (para pantallas que aún no se
  migraron) static const Color fondoClaro =
  Color(0xFFF5F5F5); static const Color fondoBlanco =
  Colors.white;

  // ======================================
  // COLORES DE TEXTO (Dark Theme)
  // ======================================
  static const Color textoBlanco = Colors.white;           //
Texto principal en dark
  static const Color textoGrisClaro = Color(0xFFE0E0E0);   //
Texto secundario
  static const Color textoGrisMedio = Color(0xFFB0B0B0);   //
Texto deshabilitado/hints
  static const Color textoGrisOscuro = Color(0xFF757575);  //
Texto terciario

  // Compatibilidad (para pantallas claras)
  static const Color textoOscuro = Color(0xFF212121);
  static const Color textoMedio = Color(0xFF757575);
  static const Color textoClaro = Color(0xFFBDBDBD);

  // ======================================
  // INFORMACIÓN DE LA APLICACIÓN
  // ======================================
  static const String nombreApp =
  'AztecaFest'; static const String versionApp
```

```
= '1.0.0';

// ======================================
// MENSAJES GENERALES
// ======================================
static const String mensajeCargando = 'Cargando...';
static const String mensajeError = 'Ha ocurrido un
error'; static const String mensajeExito = 'Operación
exitosa';
static const String mensajeSinDatos = 'No hay datos disponibles';
static const String mensajeConfirmar = '¿Está seguro?';

// ======================================
// VALIDACIONES
// ======================================
static const int longitudMinimaPassword = 6;
static const int longitudMaximaNombre = 100;
static const int longitudMaximaDescripcion =
500;

// ======================================
// CONFIGURACIÓN DE PAGINACIÓN
// ======================================
static const int elementosPorPagina = 20;

// ======================================
// ROLES DE USUARIO
// ✅ VERIFICADO: Consistente con base de datos real
// ======================================

// Roles principales (valores exactos de
BD) static const String rolAdmin =
'admin'; static const String rolVendedor =
'vendedor'; static const String rolCliente
= 'cliente';

// Alias para compatibilidad (opcional)
static const String rolAdministrador = 'admin';

// Lista de roles válidos
static const List<String> rolesValidos = [
  rolAdmin,
  rolVendedor,
  rolCliente,
];

// Lista de roles de usuario (admin y vendedor)
```

```
static const List<String> rolesUsuario = [
  rolAdmin,rolVendedor,
];

// ======================================
// MÉTODOS HELPER PARA ROLES
// ======================================

/// Verifica si un rol es de administrador
static bool esAdmin(String? rol) {
  if (rol == null) return false;
  final rolLower =
  rol.toLowerCase();
  return rolLower == 'admin' || rolLower == 'administrador';
}

/// Verifica si un rol es de vendedor
static bool esVendedor(String? rol) {
  if (rol == null) return false;
  return rol.toLowerCase() == 'vendedor';
}

/// Verifica si un rol es de cliente
static bool esCliente(String? rol) {
  if (rol == null) return false;
  return rol.toLowerCase() == 'cliente';
}

/// Verifica si un rol tiene permisos de gestión (admin o vendedor)
static bool tienePermisosGestion(String? rol) {
  return esAdmin(rol) || esVendedor(rol);
}

/// Normaliza el rol a su formato estándar
static String normalizarRol(String rol) {
  final rolLower = rol.toLowerCase();
  if (rolLower == 'admin' || rolLower == 'administrador')
    { return rolAdmin;
  }
  if (rolLower == 'vendedor') {
    return rolVendedor;
  }
  return rolCliente;
}

/// Obtiene el nombre de visualización del
rol static String getNombreRol(String? rol)
{
```

```
  if (rol == null) return 'Desconocido';

  if (esAdmin(rol)) return
  'Administrador'; if (esVendedor(rol))
  return 'Vendedor'; if (esCliente(rol))
  return 'Cliente';

  return rol; // Retorna el rol original si no coincide
}

// ======================================
// ESTADOS DE VENTA
// ======================================
static const String ventaCompletada =
'completada'; static const String ventaCancelada
= 'cancelada'; static const String ventaPendiente
= 'pendiente';

static const List<String> estadosVenta = [
  ventaPendiente,
  ventaCompletada,
  ventaCancelada,
];

// ======================================
// ESTADOS DE PEDIDO
// ======================================
static const String pedidoPendiente =
'pendiente'; static const String pedidoProcesando
= 'procesando'; static const String pedidoEnviado
= 'enviado'; static const String pedidoEntregado
= 'entregado'; static const String
pedidoCancelado = 'cancelado';

static const List<String> estadosPedido = [
  pedidoPendiente,
  pedidoProcesando,
  pedidoEnviado,
  pedidoEntregado,
  pedidoCancelado,
];

// ======================================
// MÉTODOS DE PAGO
// ======================================
static const String pagoEfectivo =
'efectivo'; static const String pagoTarjeta
```

```
= 'tarjeta';
static const String pagoTransferencia = 'transferencia';

static const List<String> metodosPago = [
  pagoEfectivo,
  pagoTarjeta,
  pagoTransferencia,
];

// ======================================
// ESPACIADO
// ======================================
static const double espaciadoPequeno = 8.0;
static const double espaciadoMedio = 16.0;
static const double espaciadoGrande = 24.0;
static const double espaciadoExtraGrande =
32.0;

// ======================================
// BORDES REDONDEADOS
// ======================================
static const double bordeRedondeadoPequeno = 4.0;
static const double bordeRedondeadoMedio = 8.0;
static const double bordeRedondeadoGrande = 12.0;
static const double bordeRedondeadoExtraGrande =
16.0;

// ======================================
// TAMAÑOS DE FUENTE
// ======================================
static const double fuentePequena = 12.0;
static const double fuenteNormal = 14.0;
static const double fuenteMediana = 16.0;
static const double fuenteGrande = 18.0;
static const double fuenteTitulo = 24.0;
static const double fuenteEncabezado =
32.0;

// ======================================
// ICONOS POR ROL
// ======================================
static IconData getIconoRol(String? rol) {
  if (esAdmin(rol)) return
  Icons.admin_panel_settings; if (esVendedor(rol))
  return Icons.point_of_sale;
  if (esCliente(rol)) return Icons.person;
  return Icons.help_outline;
}
```

```
  // ======================================
  // COLORES POR ROL
  // ======================================
  static Color getColorRol(String? rol) {
    if (esAdmin(rol)) return const Color(0xFFE74C3C); // Rojo
    if (esVendedor(rol)) return const Color(0xFF3498DB); //
    Azul if (esCliente(rol)) return const Color(0xFF2ECC71);
    // Verde return Colors.grey;
  }

  // ======================================
  // IMPUESTOS
  // ======================================
  static const double ivaEcuador = 0.12; // 12% IVA en
  Ecuador static const double ivaDefault = 0.16; // 16% IVA
  por defecto

  // ======================================
  // LÍMITES Y RESTRICCIONES
  // ======================================
  static const int stockMinimo = 0;
  static const int stockMaximo = 9999;
  static const double precioMinimo =
  0.01;
  static const double precioMaximo =
  999999.99; static const int
  cantidadMinimaCarrito = 1; static const int
  cantidadMaximaCarrito = 100;

  // ======================================
  // FORMATOS
  // ======================================
  static const String formatoFecha = 'dd/MM/yyyy';
  static const String formatoFechaHora = 'dd/MM/yyyy HH:mm';
  static const String formatoMoneda = '\$';

  // ======================================
  // RESPONSIVIDAD (Breakpoints)
  // ======================================
  static const double breakpointMobile = 600.0;   // < 600 = Móvil
  static const double breakpointTablet = 800.0;   // 600-800 = Tablet
  static const double breakpointDesktop = 1200.0; // > 800 =
  Escritorio

  /// Helper para obtener tamaños responsivos
  /// Retorna un Map con: titleSize, iconSize, fontSize, avatarSize,
padding
```

```
  static Map<String, dynamic> getResponsiveSizes(double width)
    { if (width > breakpointTablet) {
      // Desktop
      return {
        'titleSize': 22.0,
        'iconSize': 26.0,
        'fontSize': 16.0,
        'avatarSize': 44.0,
        'padding': 16.0,
        'cardPadding': 20.0,
        'buttonHeight': 50.0,
      };
    } else if (width > breakpointMobile) {
      // Tablet
      return {
        'titleSize': 20.0,
        'iconSize': 24.0,
        'fontSize': 15.0,
        'avatarSize': 40.0,
        'padding': 14.0,
        'cardPadding': 16.0,
        'buttonHeight': 46.0,
      };
    } else {
      // Móvil
      return {
        'titleSize': 18.0,
        'iconSize': 22.0,
        'fontSize': 14.0,
        'avatarSize': 36.0,
        'padding': 12.0,
        'cardPadding': 12.0,
        'buttonHeight': 44.0,
      };
    }
  }

  /// Verifica si es móvil
  static bool isMobile(double width) => width < breakpointMobile;

  /// Verifica si es tablet
  static bool isTablet(double width) => width >= breakpointMobile && width
< breakpointTablet;

  /// Verifica si es escritorio
  static bool isDesktop(double width) => width >= breakpointTablet;
}
```

4.2.4 lib/core/app_routes.dart: Definición de rutas y navegación por roles.

```
// core/app_routes.dart
// ✅ Rutas organizadas por roles y funcionalidades

class AppRoutes {
  // ======================================
  // RUTAS DE AUTENTICACIÓN
  // ======================================

  static const String login = '/login';
  static const String registro =
  '/registro';
  static const String forgotPassword = '/forgot-password';
  static const String resetPassword = '/reset-password';

  // ======================================
  // RUTAS PRINCIPALES - DASHBOARDS POR ROL
  // ======================================

  static const String home = '/'; // Redirige según rol
  static const String adminDashboard = '/admin-dashboard';
  static const String vendedorDashboard = '/vendedor-dashboard';
  static const String clienteDashboard = '/cliente-dashboard';

  // ======================================
  // RUTAS DE ADMIN
  // ======================================

  // Tabs del admin
  static const String adminPedidos = '/admin-pedidos';
  static const String adminProductos = '/admin-
  productos'; static const String adminReportes =
  '/admin-reportes'; static const String adminUsuarios =
  '/admin-usuarios'; static const String adminVentas =
  '/admin-ventas';

  // Gestión de productos (admin)
  static const String productoDetalle = '/producto-
  detalle'; static const String productoNuevo =
  '/producto-nuevo'; static const String productoEditar =
  '/producto-editar';

  // Gestión de ventas (admin)
  static const String ventaDetalle = '/venta-detalle';
  static const String ventaNueva = '/venta-nueva';
```

```
// Gestión de usuarios (admin)
static const String usuarioDetalle = '/usuario-
detalle'; static const String usuarioNuevo =
'/usuario-nuevo'; static const String usuarioEditar =
'/usuario-editar';

// Reportes
static const String reporteVentas = '/reporte-ventas';
static const String reporteInventario = '/reporte-inventario';

// ======================================
// RUTAS DE VENDEDOR
// ======================================

// Tabs del vendedor
static const String vendedorClientes = '/vendedor-clientes';
static const String vendedorCrearVenta = '/vendedor-crear-
venta'; static const String vendedorProductos = '/vendedor-
productos'; static const String vendedorVentas = '/vendedor-
ventas';
static const String vendedorPedidos = '/vendedor-pedidos';

// Gestión de clientes (vendedor)
static const String clienteDetalle = '/cliente-
detalle'; static const String clienteNuevo =
'/cliente-nuevo'; static const String clienteEditar =
'/cliente-editar';

// ======================================
// RUTAS DE CLIENTE
// ======================================

static const String catalogo = '/catalogo';
static const String carrito = '/carrito';
static const String misPedidos = '/mis-
pedidos';

// Detalle de pedido (compartido entre roles)
static const String pedidoDetalle = '/pedido-detalle';

// ======================================
// RUTAS DE PERFIL (Todos los roles)
// ======================================

static const String perfilCliente = '/perfil-
cliente'; static const String perfilUsuario =
'/perfil-usuario';
```

```
// ======================================
// RUTAS DE CONFIGURACIÓN
// ======================================

static const String configuracion = '/configuracion';
static const String configuracionUsuarios = '/configuracion-usuarios';

// ======================================
// HELPERS - Navegación según rol
// ======================================

/// Retorna el dashboard correspondiente según el
rol static String getDashboardByRole(String rol) {
  switch (rol.toLowerCase()) {
    case 'admin':
    case 'administrador':
      return adminDashboard;
    case 'vendedor':
      return vendedorDashboard;
    case 'cliente':
      return clienteDashboard;
    default:
      return clienteDashboard;
  }
}

/// Rutas exclusivas de admin
static bool requiresAdmin(String route) {
  return [
    adminDashboard,
    adminPedidos,
    adminProductos,
    adminReportes,
    adminUsuarios,
    adminVentas,
    usuarioNuevo,
    usuarioEditar,
    usuarioDetalle,
    reporteVentas,
    reporteInventario,
    configuracion,
    configuracionUsuarios,
  ].contains(route);
}

/// Rutas para admin o vendedor
static bool requiresAdminOrVendedor(String route) {
```

```
    return [
      vendedorClientes,
      vendedorCrearVenta,
      vendedorProductos,
      vendedorVentas,
      vendedorPedidos,
      clienteNuevo,
      clienteEditar,
      clienteDetalle,
      productoDetalle,
      productoNuevo,
      productoEditar,
      ventaNueva,
      ventaDetalle,
    ].contains(route);
  }

  /// Rutas públicas/cliente
  static bool isClienteRoute(String route) {
    return [
      clienteDashboard
      , catalogo,
      carrito,
      misPedidos,
      pedidoDetalle,
      perfilCliente,
    ].contains(route);
  }

  /// Obtiene la ruta de perfil según el rol
  static String getPerfilByRole(String rol) {
    if (rol.toLowerCase() == 'cliente')
      { return perfilCliente;
    }
    return perfilUsuario;
  }
}

// ======================================
// ENUMS PARA ROLES
// ======================================

enum UserRole {
  admin,
  vendedor,
```

```
    cliente;

    String get displayName
      { switch (this) {
        case UserRole.admin:
          return
          'Administrador';
                 case
          UserRole.vendedor:
          return 'Vendedor';
                 case
           UserRole.cliente:
          return 'Cliente';
      }
    }

    String get dashboardRoute {
      switch (this) {
        case UserRole.admin:
          return
        AppRoutes.adminDashboard; case
        UserRole.vendedor:
          return
        AppRoutes.vendedorDashboard; case
        UserRole.cliente:
          return AppRoutes.clienteDashboard;
      }
    }
  }
```

1.1.1 lib/data/models/pedido_estado.dart: Enumeración de estados de pedido.

```
enum PedidoEstado {
  pendientePago,
  pagado,
```

```
  procesando,
  enviado,
  entregado,
  cancelado,
}

/// Extension para obtener el nombre legible del
estado extension PedidoEstadoExtension on
PedidoEstado {
  String get nombre {
    switch (this) {
      case PedidoEstado.pendientePago:
        return 'Pendiente de Pago';
      case
        PedidoEstado.pagado:
        return 'Pagado';
      case PedidoEstado.procesando:
        return 'Procesando';
      case PedidoEstado.enviado:
        return 'Enviado';
      case PedidoEstado.entregado:
        return 'Entregado';
      case PedidoEstado.cancelado:
        return 'Cancelado';
    }
  }
}
```

5 BASE DE DATOS Y ESQUEMAS

5.1 Esquema Relacional Completo: Tablas para usuarios, productos, ventas, pedidos y sus relaciones.

5.1.1 ESQUEMA COMPLETO AZTECA FEST

```
// lib/data/database_helper.dart
import
'package:sqflite/sqflite.dart';
import 'package:path/path.dart';

class DatabaseHelper {
  static final DatabaseHelper instance = DatabaseHelper._init();
  static Database? _database;

  DatabaseHelper._init();
```

```
Future<Database> get database async {
  if (_database != null) return _database!;
  _database = await _initDB('app_database.db');
  return _database!;
}

Future<Database> _initDB(String filePath) async {
  final dbPath = await getDatabasesPath();
  final path = join(dbPath, filePath);

  return await openDatabase(
    path,
    version: 7, // ✅ CAMBIADO DE 6 A 7
    onConfigure: _onConfigure,
    onCreate: _createDB,
    onUpgrade: _onUpgrade,
  );
}

/// Configuración previa a abrir (activar foreign keys y WAL)
Future<void> _onConfigure(Database db) async {
  // Activa claves foráneas para que ON DELETE CASCADE funcione
  await db.execute('PRAGMA foreign_keys = ON');
  // Habilita Write-Ahead Logging para mejor concurrencia
  (opcional) try {
    await db.execute('PRAGMA journal_mode = WAL');
  } catch (_) {
    // No fatal si no soporta WAL en alguna plataforma
  }
}

Future<void> _createDB(Database db, int version) async {
  // Tabla de usuarios
  await db.execute('''
    CREATE TABLE usuarios (
      id INTEGER PRIMARY KEY AUTOINCREMENT,
      nombre TEXT NOT NULL,
      email TEXT UNIQUE NOT
      NULL,
      telefono TEXT,
      direccion TEXT,
      password TEXT NOT
      NULL, rol TEXT NOT
      NULL,
      activo INTEGER NOT NULL DEFAULT 1,
      fecha_creacion TEXT NOT NULL,
      fecha_actualizacion TEXT
```

```
  )
''');

// Tabla de productos
await db.execute('''
  CREATE TABLE productos (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    codigo TEXT NOT NULL UNIQUE,
    nombre TEXT NOT NULL,
    descripcion TEXT,
    categoria TEXT NOT
    NULL,
    precio_compra REAL NOT NULL,
    precio_venta REAL NOT NULL,
    stock INTEGER NOT NULL DEFAULT
    0,
    stock_minimo INTEGER NOT NULL DEFAULT
    5, proveedor TEXT,
    imagen_url TEXT,
    fecha_creacion TEXT NOT
    NULL,
    fecha_actualizacion TEXT NOT NULL,
    activo INTEGER NOT NULL DEFAULT 1,
    informacion_mexicana TEXT
  )
''');

// Tabla de ventas (con numero_venta y
metodo_pago) await db.execute('''
  CREATE TABLE ventas (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    numero_venta TEXT,
    cliente_id INTEGER,
    vendedor_id INTEGER NOT
    NULL, fecha TEXT NOT NULL,
    subtotal REAL NOT NULL,
    descuento REAL DEFAULT 0,
    impuesto REAL DEFAULT 0,
    total REAL NOT NULL,
    metodo_pago TEXT NOT NULL,
    estado TEXT NOT NULL DEFAULT 'completada',
    observaciones TEXT,
    FOREIGN KEY (cliente_id) REFERENCES usuarios (id),
    FOREIGN KEY (vendedor_id) REFERENCES usuarios (id)
  )
''');
```

```
// Tabla de detalle de ventas
await db.execute('''
  CREATE TABLE detalle_ventas (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    venta_id INTEGER NOT NULL,
    producto_id INTEGER NOT
    NULL, cantidad INTEGER NOT
    NULL, precio_unitario REAL
    NOT NULL, subtotal REAL NOT
    NULL, descuento REAL DEFAULT
    0, total REAL NOT NULL,
    FOREIGN KEY (venta_id) REFERENCES ventas (id) ON DELETE CASCADE,
    FOREIGN KEY (producto_id) REFERENCES productos (id)
  )
''');

// Tabla de pedidos
await db.execute('''
  CREATE TABLE pedidos (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    numero_pedido TEXT NOT NULL UNIQUE,
    cliente_id INTEGER NOT NULL,
    vendedor_id INTEGER,
    fecha_pedido TEXT NOT
    NULL, fecha_entrega TEXT,
    direccion_entrega TEXT,
    subtotal REAL NOT NULL,
    descuento REAL DEFAULT 0,
    impuesto REAL DEFAULT 0,
    total REAL NOT NULL,
    estado TEXT NOT NULL DEFAULT 'pendiente',
    observaciones TEXT,
    FOREIGN KEY (cliente_id) REFERENCES usuarios (id),
    FOREIGN KEY (vendedor_id) REFERENCES usuarios (id)
  )
''');

// Tabla de detalle de pedidos
await db.execute('''
  CREATE TABLE detalle_pedidos (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    pedido_id INTEGER NOT NULL,
    producto_id INTEGER NOT NULL,
    cantidad INTEGER NOT NULL,
    precio_unitario REAL NOT NULL,
    subtotal REAL NOT NULL,
```

```
        descuento REAL DEFAULT 0,
        total REAL NOT NULL,
        FOREIGN KEY (pedido_id) REFERENCES pedidos (id) ON DELETE
        CASCADE, FOREIGN KEY (producto_id) REFERENCES productos (id)
      )
    ''');

    // Crear índices
    await db.execute('CREATE INDEX idx_usuarios_email ON
usuarios(email)');
    await db.execute('CREATE INDEX idx_usuarios_rol ON
    usuarios(rol)'); await db.execute('CREATE INDEX
    idx_productos_codigo ON
productos(codigo)');
    await db.execute(
        'CREATE INDEX idx_productos_categoria ON
    productos(categoria)'); await db.execute('CREATE INDEX
    idx_ventas_fecha ON ventas(fecha)'); await db.execute('CREATE
    INDEX idx_ventas_vendedor ON
ventas(vendedor_id)');
    await db.execute('CREATE INDEX idx_pedidos_fecha ON
pedidos(fecha_pedido)');
    await db.execute('CREATE INDEX idx_pedidos_estado ON
pedidos(estado)');

    // Insertar usuario admin (RECOMENDACIÓN: hashear password antes de
guardar)
    await db.insert('usuarios', {
      'nombre': 'Administrador',
      'email':
      'admin@sistema.com',
      'telefono': '',
      'direccion': '',
      // Idealmente reemplazar por hash (sha256/bcrypt) antes de usar en
producción
      'password':
      'admin123', 'rol':
      'admin', 'activo': 1,
      'fecha_creacion': DateTime.now().toIso8601String(),
    });

    // Insertar productos de ejemplo
    await
    _insertarProductosEjemplo(db);
  }

  Future<void> _onUpgrade(Database db, int oldVersion, int
```

```
newVersion) async {
    if (oldVersion < 2) {
      try {
       await db.execute('ALTER TABLE usuarios ADD COLUMN direccion
TEXT');
      } catch (e) {
        print(' Error al agregar direccion: $e');
      }
    }

    if (oldVersion < 3) {
      try {
       await db.execute('ALTER TABLE productos ADD COLUMN imagen_url
TEXT');
      } catch (e) {
        print(' imagen_url ya existe o error: $e');
      }

      try {
       await db.execute('ALTER TABLE productos ADD COLUMN proveedor
TEXT');
      } catch (e) {
        print(' proveedor ya existe o error: $e');
      }
    }

    if (oldVersion < 4) {
      print('📝 Agregando campos a pedidos');

      try {
       await db.execute('ALTER TABLE pedidos ADD COLUMN numero_pedido
TEXT');
      } catch (e) {
        print(' numero_pedido ya existe o error: $e');
      }

      try {
        await db
            .execute('ALTER TABLE pedidos ADD COLUMN direccion_entrega
TEXT');
      } catch (e) {
        print(' direccion_entrega ya existe o error: $e');
      }
    }

    if (oldVersion < 5) {
      print(' Agregando numero_venta a tabla ventas');
```

```
      try {
       await db.execute('ALTER TABLE ventas ADD COLUMN numero_venta
TEXT');
        print(' Columna numero_venta agregada');
      } catch (e) {
        print(' numero_venta ya existe o error: $e');
      }
    }

    if (oldVersion < 6) {
      print(' Migrando tipo_pago a metodo_pago');

      try {
        final tableInfo = await db.rawQuery('PRAGMA
        table_info(ventas)'); final tieneTipoPago =
            tableInfo.any((col) => col['name'] ==
        'tipo_pago'); final tieneMetodoPago =
            tableInfo.any((col) => col['name'] == 'metodo_pago');

        if (tieneTipoPago && !tieneMetodoPago) {
          print(' Renombrando tipo_pago a metodo_pago...');

          // Crear tabla temporal con nueva estructura
          await db.execute('''
            CREATE TABLE ventas_new (
              id INTEGER PRIMARY KEY AUTOINCREMENT,
              numero_venta TEXT,
              cliente_id INTEGER,
              vendedor_id INTEGER NOT
              NULL, fecha TEXT NOT NULL,
              subtotal REAL NOT NULL,
              descuento REAL DEFAULT 0,
              impuesto REAL DEFAULT 0,
              total REAL NOT NULL,
              metodo_pago TEXT NOT NULL,
              estado TEXT NOT NULL DEFAULT 'completada',
              observaciones TEXT,
              FOREIGN KEY (cliente_id) REFERENCES usuarios (id),
              FOREIGN KEY (vendedor_id) REFERENCES usuarios (id)
            )
          ''');

          // Copiar datos (mapear tipo_pago -> metodo_pago)
          await db.execute('''
```

```
            INSERT INTO ventas_new
            SELECT id, numero_venta, cliente_id, vendedor_id, fecha,
                   subtotal, descuento, impuesto, total,
                   tipo_pago as metodo_pago, estado, observaciones
            FROM ventas
          ''');

          // Reemplazar tabla
          await db.execute('DROP TABLE ventas');
          await db.execute('ALTER TABLE ventas_new RENAME TO ventas');

          // Recrear índices relevantes
          await db.execute(
              'CREATE INDEX IF NOT EXISTS idx_ventas_fecha ON
ventas(fecha)');
          await db.execute(
              'CREATE INDEX IF NOT EXISTS idx_ventas_vendedor ON
ventas(vendedor_id)');

          print(' Migración completada: tipo_pago → metodo_pago');
        } else if (tieneMetodoPago) {
          print(' metodo_pago ya
          existe');
        } else {
          print(' Agregando metodo_pago por defecto');
          await db.execute(
              'ALTER TABLE ventas ADD COLUMN metodo_pago TEXT NOT
NULL DEFAULT "efectivo"');
        }
      } catch (e) {
        print(' Error en migración de metodo_pago: $e');
      }
    }

    //  NUEVA MIGRACIÓN PARA
    informacion_mexicana if (oldVersion < 7) {
      (' Agregando columna informacion_mexicana a productos');

      try {
        await db.execute(
            'ALTER TABLE productos ADD COLUMN informacion_mexicana TEXT');
        print(' Columna informacion_mexicana agregada exitosamente');
      } catch (e) {
        print(' Error al agregar informacion_mexicana: $e');
      }
    }
```

```
}

Future<void> _insertarProductosEjemplo(Database db) async
  { final productos = [
    {
      'codigo': 'BEB001',
      'nombre': 'Coca Cola
      500ml',
      'descripcion': 'Bebida gaseosa',
      'categoria': 'Bebidas',
      'precio_compra': 0.50,
      'precio_venta': 1.00,
      'stock': 50,
      'stock_minimo': 10,
      'proveedor': 'Coca Cola
      Company', 'imagen_url': null,
      'activo': 1,
      'fecha_creacion': DateTime.now().toIso8601String(),
      'fecha_actualizacion': DateTime.now().toIso8601String(),
    },
    {
      'codigo': 'COM001',
      'nombre': 'Hamburguesa Clásica',
      'descripcion': 'Hamburguesa con queso y vegetales',
      'categoria': 'Comida',
      'precio_compra': 2.00,
      'precio_venta': 4.50,
      'stock': 30,
      'stock_minimo': 5,
      'proveedor': 'Proveedor
      Local', 'imagen_url': null,
      'activo': 1,
      'fecha_creacion': DateTime.now().toIso8601String(),
      'fecha_actualizacion': DateTime.now().toIso8601String(),
    },
    {
      'codigo': 'SNK001',
      'nombre': 'Papas Fritas',
      'descripcion': 'Porción de papas fritas',
      'categoria': 'Snacks',
      'precio_compra': 0.80,
      'precio_venta': 2.00,
      'stock': 40,
      'stock_minimo': 8,
      'proveedor': 'Snacks Distribuidora',
      'imagen_url': null,
```

```
      'activo': 1,
      'fecha_creacion': DateTime.now().toIso8601String(),
      'fecha_actualizacion': DateTime.now().toIso8601String(),
    },
    {
      'codigo': 'POS001',
      'nombre': 'Helado de Vainilla',
      'descripcion': 'Helado artesanal de
      vainilla', 'categoria': 'Postres',
      'precio_compra': 1.00,
      'precio_venta': 2.50,
      'stock': 3,
      'stock_minimo': 5,
      'proveedor': 'Helados Artesanales',
      'imagen_url': null,
      'activo': 1,
      'fecha_creacion': DateTime.now().toIso8601String(),
      'fecha_actualizacion': DateTime.now().toIso8601String(),
    },
    {
      'codigo': 'BEB002',
      'nombre': 'Agua Mineral 500ml',
      'descripcion': 'Agua mineral sin
      gas', 'categoria': 'Bebidas',
      'precio_compra': 0.30,
      'precio_venta': 0.75,
      'stock': 100,
      'stock_minimo': 20,
      'proveedor': 'Agua Pura',
      'imagen_url': null,
      'activo': 1,
      'fecha_creacion': DateTime.now().toIso8601String(),
      'fecha_actualizacion': DateTime.now().toIso8601String(),
    },
  ];

  for (var producto in productos)
    { try {
      await db.insert('productos', producto);
    } catch (e) {
      print(' Error al insertar producto ${producto['codigo']}: $e');
    }
  }
}
```

```
/// Cierra la conexión y limpia la instancia
Future<void> close() async {
  if (_database != null) {
    try {
      await _database!.close();
    } catch (e) {
      print(' Error cerrando DB: $e');
    } finally {
      _database = null;
    }
  }
}

Future<void> resetDatabase() async {
  final dbPath = await
  getDatabasesPath();
  final path = join(dbPath, 'app_database.db');
  await deleteDatabase(path);
  _database = null;
}

Future<bool> columnaExiste(String tabla, String columna) async {
  final db = await database;
  final resultado = await db.rawQuery('PRAGMA table_info($tabla)');
  return resultado.any((col) => col['name'] == columna);
}

Future<void> mostrarEstructuraTabla(String tabla) async {
  final db = await database;
  final resultado = await db.rawQuery('PRAGMA table_info($tabla)');

  print('\n=======================================');
  print(' ESTRUCTURA DE LA TABLA: $tabla');
  print('=======================================');

  for (var columna in resultado) {
    print('Columna: ${columna['name']}, '
        'Tipo: ${columna['type']}, '
        'Not Null: ${columna['notnull']}, '
        'Default: ${columna['dflt_value']}, '
        'PK: ${columna['pk']}');
  }
  print('=======================================\n
  ');
}

Future<void> imprimirRutaBaseDatos() async {
```

```
    final dbPath = await getDatabasesPath();
    final rutaCompleta = join(dbPath, 'app_database.db');

    print('\n======================================')
    ; print(' UBICACIÓN DE LA BASE DE DATOS:');
    print(rutaCompleta);
    print(' Estado del archivo: (no disponible en web)');
    print('======================================\n');
  }

  Future<Map<String, dynamic>> obtenerInfoBaseDatos() async
    { final dbPath = await getDatabasesPath();
    final rutaCompleta = join(dbPath,
    'app_database.db'); final existe =
        true; // En web, se usa IndexedDB; asumimos conexión si se abre

    Map<String, dynamic> info =
      { 'ruta': rutaCompleta,
      'existe': existe,
      'tamano_bytes': 0,
      'tablas': <String,
      int>{},
    };

    if (existe) {
      // Tamaño del archivo no disponible en web
      info['tamano_bytes'] = 0;

      final db = await database;
      final tablas = [
        'usuarios',
        'productos',
        'ventas',
        'detalle_ventas',
        'pedidos',
        'detalle_pedidos'
      ];

      for (var tabla in tablas) {
        try {
          final resultado =
              await db.rawQuery('SELECT COUNT(*) as count FROM $tabla');
          // Uso de Sqflite.firstIntValue es más robusto cuando el
resultado es List<Map>
          final count = Sqflite.firstIntValue(resultado) ?? 0;
          info['tablas'][tabla] = count;
```

```
        } catch (e) {
          info['tablas'][tabla] = -1;
        }
      }
    }

    return info;
  }

  Future<void> imprimirTodosLosDatos() async {
    final db = await database;

    print('\n======================================')
    ; print(' CONTENIDO DE LA BASE DE DATOS');
    print('======================================\n')
    ;

    print('👥 TABLA: USUARIOS');
    print('                                        ');
    final usuarios = await
    db.query('usuarios'); for (var usuario in
    usuarios) {
      print(
          'ID: ${usuario['id']} | ${usuario['nombre']} |
${usuario['email']} | Rol: ${usuario['rol']}');
    }
    print('Total: ${usuarios.length} registros\n');

    print(' TABLA: PRODUCTOS');
    print('                                        ');
    final productos = await
    db.query('productos'); for (var producto in
    productos) {
      print(
          'ID: ${producto['id']} | ${producto['codigo']} |
${producto['nombre']} | Stock: ${producto['stock']} |
\$${producto['precio_venta']}');
    }
    print('Total: ${productos.length} registros\n');

    print(' TABLA: VENTAS');
    print('                                        ');
    final ventas = await db.query('ventas');
    for (var venta in ventas) {
      print(
          'ID: ${venta['id']} | #${venta['numero_venta']} | Método:
${venta['metodo_pago']} | Total: \$${venta['total']}');
```

```
    }
    print('Total: ${ventas.length} registros\n');

    print(' TABLA: PEDIDOS');
    print('                                          ');
    final pedidos = await db.query('pedidos');
    for (var pedido in pedidos) {
      print(
          'ID: ${pedido['id']} | #${pedido['numero_pedido']} | Total:
\$${pedido['total']}');
    }
    print('Total: ${pedidos.length} registros\n');

    print('========================================\n');
  }

  //  MÉTODO PARA VER ESTRUCTURA EN CONSOLA
  Future<void> verEstructuraTabla(String nombreTabla) async
    { final db = await database;

    print('\n' + '=' * 60);
    print(' ESTRUCTURA DE LA TABLA: $nombreTabla');
    print('=' * 60);

    // Ver columnas
    final columnas = await db.rawQuery('PRAGMA table_info($nombreTabla)');

    print('\n
    COLUMNAS:');
    print('-' * 60);
    for (var col in columnas) {
      print('${col['cid']}. ${col['name']}'.padRight(25) +
          'Tipo: ${col['type']}'.padRight(20) +
            (col['notnull'] == 1 ? '⚠ NOT NULL' : '✓ NULL'));
    }

    // Ver datos de ejemplo
    final datos = await db.query(nombreTabla, limit:
    3); print('\n DATOS (primeros 3 registros):');
    print('-' * 60);

    if (datos.isEmpty) {
      print('⚠ No hay datos en la tabla');
    } else {
      for (var i = 0; i < datos.length; i++) {
        print('\nRegistro ${i + 1}:');
        datos[i].forEach((key, value) {
```

```
          print('  $key: $value');
        });
      }
    }

    print('\n' + '=' * 60 + '\n');
  }

  // MÉTODO PARA VER TODAS LAS TABLAS
  Future<void> verTodasLasTablas() async {
    final db = await database;

    final tablas = await db.rawQuery(
        "SELECT name FROM sqlite_master WHERE type='table' AND name NOT
LIKE 'sqlite_%'");

    print('\n' + '🗄️  TABLAS EN LA BASE DE DATOS'.padRight(60,
    '=')); for (var tabla in tablas) {
      print('  • ${tabla['name']}');
      await verEstructuraTabla(tabla['name'] as String);
    }
  }
}
```

1.2 Políticas RLS (Row Level Security): Reglas de seguridad a nivel de fila para control de acceso por rol.

5.1.2 POLÍTICAS DE SEGURIDAD (RLS):

```
-- Habilitar RLS en todas las
tablas DO $$
DECLARE
    table_name text;
BEGIN
    FOR table_name IN
        SELECT tablename
        FROM pg_tables
        WHERE schemaname = 'public'
        AND tablename NOT LIKE
        'pg_%'
    LOOP
        EXECUTE format('ALTER TABLE %I ENABLE ROW LEVEL SECURITY',
table_name);
    END LOOP;
END $$;
```

```
-- 1. POLÍTICAS PARA USUARIOS
CREATE POLICY "Usuarios pueden ver su propio
perfil" ON usuarios FOR SELECT
USING (auth.uid() = id OR auth.jwt() ->> 'role' = 'admin');

CREATE POLICY "Usuarios pueden actualizar su propio
perfil" ON usuarios FOR UPDATE
USING (auth.uid() = id);

CREATE POLICY "Admin puede gestionar todos los
usuarios" ON usuarios FOR ALL
USING (auth.jwt() ->> 'role' = 'admin');

-- 2. POLÍTICAS PARA PRODUCTOS
CREATE POLICY "Productos visibles a todos cuando están
activos" ON productos FOR SELECT
USING (activo = true);

CREATE POLICY "Admin y vendedores pueden gestionar
productos" ON productos FOR ALL
USING (
  auth.jwt() ->> 'role' IN ('admin', 'vendedor')
);

-- 3. POLÍTICAS PARA PEDIDOS
CREATE POLICY "Clientes ven sus propios
pedidos" ON pedidos FOR SELECT
USING (
  cliente_id = auth.uid() OR
  auth.jwt() ->> 'role' IN ('admin', 'vendedor')
);

CREATE POLICY "Clientes pueden crear
pedidos" ON pedidos FOR INSERT
WITH CHECK (cliente_id = auth.uid());

CREATE POLICY "Vendedores pueden actualizar
pedidos" ON pedidos FOR UPDATE
USING (auth.jwt() ->> 'role' IN ('admin', 'vendedor'));

-- 4. POLÍTICAS PARA VENTAS
CREATE POLICY "Vendedores ven sus propias
ventas" ON ventas FOR SELECT
USING (
```

```
  vendedor_id = auth.uid() OR
  auth.jwt() ->> 'role' = 'admin'
);

CREATE POLICY "Vendedores pueden crear
ventas" ON ventas FOR INSERT
WITH CHECK (vendedor_id = auth.uid());

-- 5. POLÍTICAS PARA INVENTARIO
CREATE POLICY "Solo admin puede gestionar
inventario" ON inventario FOR ALL
USING (auth.jwt() ->> 'role' = 'admin');

-- 6. Crear función para logs automáticos
CREATE OR REPLACE FUNCTION log_auditoria()
RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO logs_auditoria (
        usuario_id,
        accion,
        tabla_afectada,
        registro_id,
        valores_anteriores,
        valores_nuevos
    ) VALUES (
        auth.uid(),
        TG_OP,
        TG_TABLE_NAME,
        COALESCE(NEW.id, OLD.id),
        CASE WHEN TG_OP IN ('UPDATE', 'DELETE') THEN row_to_json(OLD)
        END, CASE WHEN TG_OP IN ('INSERT', 'UPDATE') THEN
        row_to_json(NEW) END
    );
    RETURN COALESCE(NEW, OLD);
END;
$$ LANGUAGE plpgsql SECURITY DEFINER;

-- 7. Triggers para tablas importantes
CREATE TRIGGER usuarios_audit_trigger
AFTER INSERT OR UPDATE OR DELETE ON usuarios
FOR EACH ROW EXECUTE FUNCTION log_auditoria();

CREATE TRIGGER productos_audit_trigger
AFTER INSERT OR UPDATE OR DELETE ON productos
FOR EACH ROW EXECUTE FUNCTION log_auditoria();
```

```
CREATE TRIGGER pedidos_audit_trigger
AFTER INSERT OR UPDATE OR DELETE ON pedidos
FOR EACH ROW EXECUTE FUNCTION log_auditoria();
```

1.3 Funciones y Triggers Avanzados: Automatización de actualizaciones de stock, cálculos de totales, logs de auditoría.

5.1.3 FUNCIONES Y TRIGGERS AVANZADOS

```
-- 1. Función para actualizar automáticamente fecha_actualizacion
CREATE OR REPLACE FUNCTION update_updated_at_column()
RETURNS TRIGGER AS $$
BEGIN
    NEW.fecha_actualizacion =
    NOW(); RETURN NEW;
END;
$$ language 'plpgsql';

-- Aplicar trigger a todas las tablas con
fecha_actualizacion DO $$
DECLARE
    table_name text;
BEGIN
    FOR table_name IN
        SELECT tablename
        FROM pg_tables
        WHERE schemaname = 'public'
        AND EXISTS (
            SELECT 1
            FROM
            information_schema.columns
            WHERE table_schema = 'public'
            AND table_name = tablename
            AND column_name = 'fecha_actualizacion'
        )
    LOOP
        EXECUTE format('
            CREATE TRIGGER update_%s_updated_at
            BEFORE UPDATE ON %I
            FOR EACH ROW EXECUTE FUNCTION update_updated_at_column()',
            table_name, table_name);
```

```
-- 2. Función para actualizar stock al crear detalles de
pedido CREATE OR REPLACE FUNCTION actualizar_stock_pedido()
RETURNS TRIGGER AS $$
BEGIN
    -- Actualizar stock del producto
    UPDATE productos
    SET stock = stock - NEW.cantidad,
        fecha_actualizacion = NOW()
    WHERE id = NEW.producto_id;

    -- Registrar movimiento de
    inventario INSERT INTO inventario (
        producto_id,
        tipo_movimiento,
        cantidad,
        cantidad_anterior,
        cantidad_nueva,
        referencia_id,
        referencia_tipo,
        usuario_id
    ) SELECT
        NEW.producto_id,
        'pedido',
        NEW.cantidad,
        p.stock +
        NEW.cantidad,
        p.stock,
        NEW.pedido_id,
        'pedido',
        p.cliente_id
    FROM pedidos p
    WHERE p.id = NEW.pedido_id;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER actualizar_stock_pedido_trigger
AFTER INSERT ON detalles_pedido
FOR EACH ROW EXECUTE FUNCTION actualizar_stock_pedido();

-- 3. Función para calcular total del pedido automáticamente
CREATE OR REPLACE FUNCTION calcular_total_pedido()
RETURNS TRIGGER AS $$
BEGIN
    NEW.total = COALESCE(
```

```
        (SELECT SUM(subtotal) FROM detalles_pedido WHERE pedido_id =
NEW.id),
        0
    ) - COALESCE(NEW.descuento, 0);
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER calcular_total_pedido_trigger
BEFORE INSERT OR UPDATE ON pedidos
FOR EACH ROW EXECUTE FUNCTION calcular_total_pedido();

-- 4. Función para validar stock antes de crear pedido
CREATE OR REPLACE FUNCTION validar_stock_pedido()
RETURNS TRIGGER AS $$
DECLARE
    producto_stock INTEGER;
BEGIN
    SELECT stock INTO producto_stock
    FROM productos
    WHERE id = NEW.producto_id;

    IF producto_stock < NEW.cantidad THEN
        RAISE EXCEPTION 'Stock insuficiente para el producto ID: %',
NEW.producto_id;
    END IF;

    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER validar_stock_pedido_trigger
BEFORE INSERT ON detalles_pedido
FOR EACH ROW EXECUTE FUNCTION validar_stock_pedido();
```

---

6 CONFIGURACIÓN DE SUPABASE

6.1 Configuración Inicial de Proyecto

6.1.1 Este le hace que sea una Conexión a Supabase con URL y clave anónima.

```
// lib/config/supabase_config.dart

import

'package:supabase_flutter/supabase_flutter.dart';  class

SupabaseConfig {
  // 🔑 TUS CREDENCIALES
  static const String supabaseUrl =
'https://dandioeximpeqlldbscd.supabase.co';
  static const String supabaseAnonKey =
'sb_publishable_Fz0Ow7aqVqR09aSNRwMKOA_Xo18kz7W';

  // Cliente de Supabase (singleton)
  static SupabaseClient get client => Supabase.instance.client;

  // Helpers para acceder a las tablas
  static SupabaseQueryBuilder get usuarios =>
  client.from('usuarios'); static SupabaseQueryBuilder get productos
  => client.from('productos'); static SupabaseQueryBuilder get
  ventas => client.from('ventas'); static SupabaseQueryBuilder get
  detalleVentas =>
client.from('detalle_ventas');
  static SupabaseQueryBuilder get pedidos =>
  client.from('pedidos'); static SupabaseQueryBuilder get
  detallePedidos =>
client.from('detalle_pedidos');

  // Helper para Storage
  static SupabaseStorageClient get storage => client.storage;
  static StorageFileApi get productImages =>
  storage.from('product-
images');

  // Método para inicializar Supabase
  static Future<void> initialize() async
  {
    await
      Supabase.initialize(
      url: supabaseUrl,
      anonKey:
```

```
  // Método para verificar conexión
  static Future<bool> verificarConexion() async
    { try {
      await usuarios.select('id').limit(1);
      print('✅ Conexión a Supabase exitosa');
      return true;
    } catch (e) {
      print('❌ Error de conexión a Supabase: $e');
      return false;
    }
  }
}
```

6.2 Configuración de Storage

6.2.1 Este es el Servicio para subir, eliminar y gestionar imágenes de productos.

```
// lib/services/supabase_storage_service.dart


import 'dart:io';

import 'package:supabase_flutter/supabase_flutter.dart';

import '../config/supabase_config.dart';


class SupabaseStorageService {

  static final _storage = SupabaseConfig.productImages;


  /// Sube una imagen al bucket de Supabase y retorna la URL
pública

  ///

  /// [file] - Archivo de imagen a subir

  /// [nombreProducto] - Nombre del producto (se usa para el
nombre del archivo)

  ///

  /// Retorna la URL pública de la imagen subida

  static Future<String> subirImagen(File file, String
nombreProducto) async {
```

```
    try {
      // Generar nombre único para el archivo
      final timestamp =
DateTime.now().millisecondsSinceEpoch;
      final extension = file.path.split('.').last;
      final fileName = '${nombreProducto.replaceAll(' ',
'_')}_$timestamp.$extension';

      // Subir archivo a Supabase Storage
      await _storage.upload(
        fileName,
        file,
        fileOptions: const FileOptions(
          cacheControl: '3600',
          upsert: false,
        ),
      );

      // Obtener URL pública
      final publicUrl = _storage.getPublicUrl(fileName);

      print('✅ Imagen subida exitosamente: $fileName');
      return publicUrl;

    } catch (e) {
      print('❌ Error al subir imagen: $e');
      rethrow;
    }
  }
```

```
  /// Elimina una imagen del bucket usando su URL
  ///
  /// [imageUrl] - URL pública de la imagen a eliminar
  static Future<void> eliminarImagen(String imageUrl) async {
    try {
      // Extraer el nombre del archivo de la URL
      final fileName = _extractFileNameFromUrl(imageUrl);

      if (fileName != null) {
        await _storage.remove([fileName]);
        print('✅ Imagen eliminada exitosamente: $fileName');
      }
    } catch (e) {
      print('❌ Error al eliminar imagen: $e');
      // No lanzar error para no bloquear operaciones
    }
  }

  /// Actualiza una imagen: elimina la anterior y sube la
nueva
  ///
  /// [file] - Nuevo archivo de imagen
  /// [nombreProducto] - Nombre del producto
  /// [oldImageUrl] - URL de la imagen anterior (opcional)
  ///
  /// Retorna la URL pública de la nueva imagen
  static Future<String> actualizarImagen(
```

```
  File file,
  String nombreProducto, {
  String? oldImageUrl,
}) async {
  try {
    // Eliminar imagen anterior si existe
    if (oldImageUrl != null && oldImageUrl.isNotEmpty) {
      await eliminarImagen(oldImageUrl);
    }

    // Subir nueva imagen
    return await subirImagen(file, nombreProducto);

  } catch (e) {
    print('❌ Error al actualizar imagen: $e');
    rethrow;
  }
}

/// Lista todas las imágenes en el bucket
static Future<List<FileObject>> listarImagenes() async {
  try {
    final files = await _storage.list();
    return files;
  } catch (e) {
    print('❌ Error al listar imágenes: $e');
    return [];
  }
```

```
  }

  /// Limpia imágenes huérfanas (que no están asociadas a
ningún producto)
  ///
  /// [urlsActivas] - Lista de URLs que están siendo usadas
por productos
  static Future<void> limpiarImagenesHuerfanas(List<String>
urlsActivas) async {
    try {
      // Obtener todas las imágenes en el bucket
      final todasLasImagenes = await listarImagenes();

      // Extraer nombres de archivos de las URLs activas
      final nombresActivos = urlsActivas
          .map((url) => _extractFileNameFromUrl(url))
          .where((name) => name != null)
          .cast<String>()
          .toSet();

      // Identificar imágenes huérfanas
      final imagenesHuerfanas = todasLasImagenes
          .where((file) =>
!nombresActivos.contains(file.name))
          .map((file) => file.name)
          .toList();

      // Eliminar imágenes huérfanas
      if (imagenesHuerfanas.isNotEmpty) {
        await _storage.remove(imagenesHuerfanas);
```

```
        print('✅ ${imagenesHuerfanas.length} imágenes huérfanas eliminadas');
      } else {
        print('✅ No hay imágenes huérfanas');
      }

    } catch (e) {
      print('❌ Error al limpiar imágenes huérfanas: $e');
    }
  }

  /// Extrae el nombre del archivo de una URL pública de Supabase
  static String? _extractFileNameFromUrl(String url) {
    try {
      // URL format: https://xxx.supabase.co/storage/v1/object/public/PRODUCT-IMAGES/filename.jpg
      final uri = Uri.parse(url);
      final segments = uri.pathSegments;

      // El último segmento es el nombre del archivo
      if (segments.isNotEmpty) {
        return segments.last;
      }
      return null;
    } catch (e) {
      print('❌ Error al extraer nombre de archivo de URL: $e');
      return null;
```

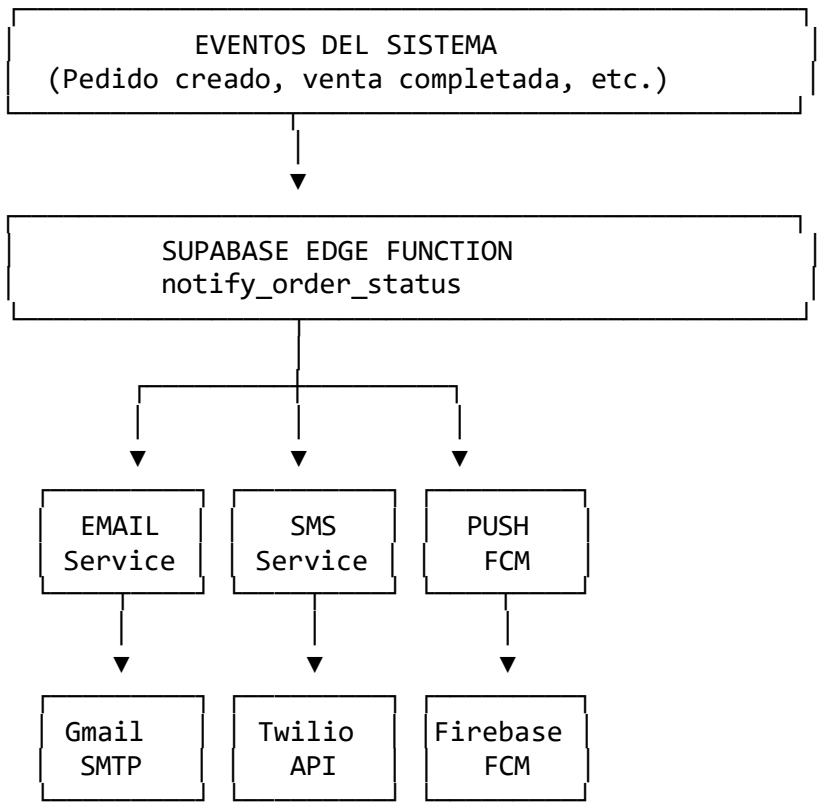
```
    }
  }

  /// Verifica si una URL es de Supabase Storage
  static bool esUrlSupabase(String url) {
    return url.contains('supabase.co/storage');
  }
}
```

6.3 Configuración de Edge Functions

6.3.1 Funciones serverless para notificaciones (email, SMS, push).

Descripción GeneralLas Edge Functions de Supabase son funciones serverless que se ejecutan en el edge (cerca del usuario) para realizar operaciones backend como envío de notificaciones, procesamiento de datos, y webhooks. En Azteca Fest, se utilizan principalmente para sistemas de notificaciones multi-canal.

6.3.2 Arquitectura de Notificaciones



6.3.3 Creacion de Edge Functions

Paso 1: Instalar Supabase CLI

```
# Instalar Supabase CLI
npm install -g supabase

# Iniciar proyecto local
```

```
supabase init

# Crear una nueva Edge Function
supabase functions new notify_order_status
```

Esto creará la estructura:
```
supabase/
└── functions/
    └── notify_order_status/
        ├── index.ts          # Código principal
        └── .env.example      # Variables de entorno
```

6.3.4 Código de la Edge Function: Notificación de Pedidos

```
// ========================================
// EDGE FUNCTION: notify_order_status
// ========================================
// Envía notificaciones multi-canal cuando cambia el estado de un pedido

import { serve } from "https://deno.land/std@0.168.0/http/server.ts";
import { createClient } from "https://esm.sh/@supabase/supabase-js@2";

// ========================================
// TIPOS Y INTERFACES
// ========================================

interface NotificationRequest {
  order_id: number;
  user_id: string;
  status: string;
  notification_types: ("email" | "sms" | "push")[];
}

interface Order {
  id: number;
  numero_pedido: string;
  total: number;
  estado: string;
  fecha_pedido: string;
}

interface User {
  id: string;
  nombre: string;
  email: string;
  telefono?: string;
}

interface FCMToken {
  token: string;
  device_type: string;
}
```

```
// ======================================
// CONFIGURACIÓN
// ======================================

// CORS headers para permitir llamadas desde la app
const corsHeaders = {
  "Access-Control-Allow-Origin": "*",
  "Access-Control-Allow-Headers": "authorization, x-client-info, apikey,
content-type",
};

// ======================================
// FUNCIÓN PRINCIPAL
// ======================================

serve(async (req) => {
  // Manejar preflight CORS
  if (req.method === "OPTIONS") {
    return new Response("ok", { headers: corsHeaders });
  }

  try {
    console.log("🔔 Iniciando proceso de notificaciones...");

    // Crear cliente Supabase autenticado
    const supabaseClient = createClient(
      Deno.env.get("SUPABASE_URL") ?? "",
      Deno.env.get("SUPABASE_ANON_KEY") ?? "",
      {
        global: {
          headers: { Authorization: req.headers.get("Authorization")! },
        },
      }
    );

    // Obtener datos de la petición
    const { order_id, user_id, status, notification_types }:
NotificationRequest =
      await req.json();

    console.log(`🎁 Pedido ID: ${order_id}`);
    console.log(`👤 Usuario ID: ${user_id}`);
    console.log(`📊 Nuevo estado: ${status}`);
    console.log(`📢 Canales: ${notification_types.join(", ")}`);

    // ======================================
    // OBTENER DATOS DEL PEDIDO
    // ======================================

    const { data: order, error: orderError } = await supabaseClient
      .from("pedidos")
      .select("*")
      .eq("id", order_id)
      .single();
```

```
if (orderError) {
  console.error("❌ Error al obtener pedido:", orderError);
  throw orderError;
}

console.log(`✅ Pedido obtenido: ${order.numero_pedido}`);

// ========================================
// OBTENER DATOS DEL USUARIO
// ========================================

const { data: user, error: userError } = await supabaseClient
  .from("usuarios")
  .select("*")
  .eq("id", user_id)
  .single();

if (userError) {
  console.error("❌ Error al obtener usuario:", userError);
  throw userError;
}

console.log(`✅ Usuario obtenido: ${user.nombre}`);

// ========================================
// ENVIAR NOTIFICACIONES POR CADA CANAL
// ========================================

const notifications: Promise<any>[] = [];

// 1️⃣ NOTIFICACIÓN POR EMAIL
if (notification_types.includes("email") && user.email) {
  console.log("📧 Preparando notificación por email...");
  notifications.push(
    sendEmailNotification(user, order, status)
  );
}

// 2️⃣ NOTIFICACIÓN POR SMS
if (notification_types.includes("sms") && user.telefono) {
  console.log("📱 Preparando notificación por SMS...");
  notifications.push(
    sendSMSNotification(user, order, status)
  );
}

// 3️⃣ NOTIFICACIÓN PUSH
if (notification_types.includes("push")) {
  console.log("🔔 Preparando notificación push...");

  // Obtener tokens FCM del usuario
  const { data: fcmTokens } = await supabaseClient
```

```
    .from("user_fcm_tokens")
    .select("token, device_type")
    .eq("user_id", user_id)
    .eq("active", true);

  if (fcmTokens && fcmTokens.length > 0) {
    console.log(`    Tokens encontrados: ${fcmTokens.length}`);
    for (const tokenData of fcmTokens) {
      notifications.push(
        sendPushNotification(tokenData, user, order, status)
      );
    }
  } else {
     console.log("    ⚠ No se encontraron tokens FCM activos");
  }
}

// Ejecutar todas las notificaciones en paralelo
console.log(`🚀 Enviando ${notifications.length} notificaciones...`);
const results = await Promise.allSettled(notifications);

// Analizar resultados
const successful = results.filter(r => r.status === "fulfilled").length;
const failed = results.filter(r => r.status === "rejected").length;

console.log(`✅ Exitosas: ${successful}`);
console.log(`❌ Fallidas: ${failed}`);

// Registrar en tabla de logs (opcional)
await supabaseClient.from("notification_logs").insert({
  order_id,
  user_id,
  status,
  channels_sent: notification_types,
  successful_count: successful,
  failed_count: failed,
  timestamp: new Date().toISOString(),
});

// Retornar respuesta
return new Response(
   JSON.stringify({
    success: true,
    message: "Notificaciones procesadas",
    stats: {
      total: notifications.length,
      successful,
      failed,
    },
  }),
  {
    headers: { ...corsHeaders, "Content-Type": "application/json" },
    status: 200,
  }
```

```
    );

  } catch (error) {
    console.error("❌ Error general:", error);

    return new Response(
      JSON.stringify({
        success: false,
        error: error.message,
      }),
      {
        headers: { ...corsHeaders, "Content-Type": "application/json" },
        status: 400,
      }
    );
  }
});

// ======================================
// FUNCIONES AUXILIARES
// ======================================

/**
 * 📧 Enviar notificación por EMAIL
 */
async function sendEmailNotification(
  user: User,
  order: Order,
  status: string
): Promise<void> {
  try {
    console.log(`   📧 Enviando email a: ${user.email}`);

    // Aquí puedes integrar con servicios como:
    // - SendGrid
    // - Resend
    // - AWS SES
    // - Gmail API

    const emailBody = buildEmailTemplate(user, order, status);

    // Ejemplo con fetch a un servicio de email
    const response = await fetch("https://api.sendgrid.com/v3/mail/send", {
      method: "POST",
      headers: {
        "Authorization": `Bearer ${Deno.env.get("SENDGRID_API_KEY")}`,
        "Content-Type": "application/json",
      },
      body: JSON.stringify({
        personalizations: [{
          to: [{ email: user.email, name: user.nombre }],
        }],
        from: {
          email: "noreply@aztecafest.com",
```

```
          name: "AztecaFest",
        },
        subject: `Actualización de Pedido #${order.numero_pedido}`,
        content: [{
          type: "text/html",
          value: emailBody,
        }],
      }),
    });

    if (!response.ok) {
      throw new Error(`Error al enviar email: ${response.statusText}`);
    }

    console.log(`    ✅ Email enviado correctamente`);
  } catch (error) {
    console.error(`    ❌ Error en email:`, error);
    throw error;
  }
}

/**
 * 📱 Enviar notificación por SMS
 */
async function sendSMSNotification(
  user: User,
  order: Order,
  status: string
): Promise<void> {
  try {
    console.log(`    📱 Enviando SMS a: ${user.telefono}`);

    // Integración con Twilio
    const twilioAccountSid = Deno.env.get("TWILIO_ACCOUNT_SID");
    const twilioAuthToken = Deno.env.get("TWILIO_AUTH_TOKEN");
    const twilioPhoneNumber = Deno.env.get("TWILIO_PHONE_NUMBER");

    const message = buildSMSMessage(user, order, status);

    const response = await fetch(
      `https://api.twilio.com/2010-04-
01/Accounts/${twilioAccountSid}/Messages.json`,
      {
        method: "POST",
        headers: {
          "Authorization": `Basic
${btoa(`${twilioAccountSid}:${twilioAuthToken}`)}`,
          "Content-Type":  "application/x-www-form-urlencoded",
        },
        body: new URLSearchParams({
          To:  user.telefono!,
          From: twilioPhoneNumber!,
          Body: message,
        }),
```

```
      }
    );

    if (!response.ok) {
      throw new Error(`Error al enviar SMS: ${response.statusText}`);
    }

    console.log(`    ✅ SMS enviado correctamente`);
  } catch (error) {
    console.error(`    ❌ Error en SMS:`, error);
    throw error;
  }
}

/**
 * 🔔 Enviar notificación PUSH (FCM)
 */
async function sendPushNotification(
  tokenData: FCMToken,
  user: User,
  order: Order,
  status: string
): Promise<void> {
  try {
    console.log(`    🔔 Enviando push a: ${tokenData.device_type}`);

    const fcmServerKey = Deno.env.get("FCM_SERVER_KEY");

    const notification = buildPushNotification(user, order, status);

    const response = await fetch("https://fcm.googleapis.com/fcm/send", {
      method: "POST",
      headers: {
        "Authorization": `key=${fcmServerKey}`,
        "Content-Type": "application/json",
      },
      body: JSON.stringify({
        to: tokenData.token,
        notification: notification,
        data: {
          order_id: order.id.toString(),
          order_number: order.numero_pedido,
          status: status,
          click_action: "FLUTTER_NOTIFICATION_CLICK",
        },
        priority: "high",
      }),
    });

    if (!response.ok) {
      throw new Error(`Error al enviar push: ${response.statusText}`);
    }

    console.log(`    ✅ Push enviado correctamente`);
```

```
  } catch (error) {
    console.error(`    ❌ Error en push:`, error);
    throw error;
  }
}

// ========================================
// TEMPLATES DE MENSAJES
// ========================================

/**
 * Construir template HTML para email
 */
function buildEmailTemplate(user: User, order: Order, status: string): string
{
  const statusMessages = {
    "pendiente": "Tu pedido ha sido recibido y está pendiente de
confirmación.",
    "procesando": "¡Buenas noticias! Tu pedido está siendo preparado.",
    "enviado": "Tu pedido ha sido enviado y está en camino.",
    "entregado": "¡Tu pedido ha sido entregado! Esperamos que lo disfrutes.",
    "cancelado": "Tu pedido ha sido cancelado.",
  };

  const statusEmojis = {
    "pendiente": "⏳",
    "procesando": "👨‍🍳",
    "enviado": "🚚",
    "entregado": "✅",
    "cancelado": "❌",
  };

  return `
    <!DOCTYPE html>
    <html>
    <head>
      <meta charset="UTF-8">
      <style>
        body {
          font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
          line-height: 1.6;
          color: #333;
          background-color: #f5f5f5;
          margin: 0;
          padding: 20px;
        }
        .container {
          max-width: 600px;
          margin: 0 auto;
          background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
          padding: 20px;
          border-radius: 10px;
        }
        .content {
```

```
      background: white;
      padding: 30px;
      border-radius: 8px;
      box-shadow: 0 4px 6px rgba(0,0,0,0.1);
    }
    .header {
      text-align: center;
      margin-bottom: 30px;
    }
    .logo {
      font-size: 32px;
      font-weight: bold;
      color: #E74C3C;
      margin-bottom: 10px;
    }
    .status-badge {
      display: inline-block;
      padding: 10px 20px;
      background: linear-gradient(135deg, #E74C3C, #FF6B6B);
      color: white;
      border-radius: 25px;
      font-weight: bold;
      margin: 20px 0;
      font-size: 18px;
    }
    .order-info {
      background: #f8f9fa;
      padding: 20px;
      border-radius: 8px;
      margin: 20px 0;
    }
    .info-row {
      display: flex;
      justify-content: space-between;
      padding: 10px 0;
      border-bottom: 1px solid #dee2e6;
    }
    .info-row:last-child {
      border-bottom: none;
    }
    .footer {
      text-align: center;
      margin-top: 30px;
      color: #6c757d;
      font-size: 12px;
    }
  </style>
</head>
<body>
  <div class="container">
    <div class="content">
      <div class="header">
        <div class="logo">🏟️ AztecaFest</div>
        <h1>Actualización de Pedido</h1>
```

```
          </div>

          <p>¡Hola <strong>${user.nombre}</strong>!</p>

          <div class="status-badge">
            ${statusEmojis[status] || "📦"} ${status.toUpperCase()}
          </div>

          <p>${statusMessages[status] || "Tu pedido ha sido
actualizado."}</p>

          <div class="order-info">
            <div class="info-row">
              <strong>Número de Pedido:</strong>
              <span>${order.numero_pedido}</span>
            </div>
            <div class="info-row">
              <strong>Total:</strong>
              <span>$${order.total.toFixed(2)}</span>
            </div>
            <div class="info-row">
              <strong>Fecha:</strong>
              <span>${new Date(order.fecha_pedido).toLocaleDateString('es-
ES')}</span>
            </div>
            <div class="info-row">
              <strong>Estado:</strong>
              <span><strong>${status}</strong></span>
            </div>
          </div>

          <p>Gracias por confiar en AztecaFest. ¡Disfruta de nuestros
deliciosos tacos mexicanos!</p>

          <div class="footer">
            <p>© 2025 AztecaFest. Todos los derechos reservados.</p>
            <p>Este email fue enviado automáticamente. No responda a este
correo.</p>
          </div>
        </div>
      </div>
    </body>
    </html>
  `;
}

/**
 * Construir mensaje para SMS
 */
function buildSMSMessage(user: User, order: Order, status: string): string {
  const statusMessages = {
    "pendiente": "recibido",
    "procesando": "está siendo preparado",
    "enviado": "ha sido enviado",
```

```
    "entregado": "ha sido entregado",
    "cancelado": "ha sido cancelado",
  };

  return `AztecaFest: Hola ${user.nombre}, tu pedido #${order.numero_pedido}
${statusMessages[status] || "ha sido actualizado"}. Total:
$${order.total.toFixed(2)}`;
}

/**
 * Construir notificación push
 */
function buildPushNotification(user: User, order: Order, status: string) {
  const statusTitles = {
    "pendiente": "Pedido Recibido",
    "procesando": "Preparando tu Pedido",
    "enviado": "Pedido en Camino",
    "entregado": "Pedido Entregado",
    "cancelado": "Pedido Cancelado",
  };

  const statusBodies = {
    "pendiente": `Tu pedido #${order.numero_pedido} ha sido recibido.`,
    "procesando": `¡Tu pedido está siendo preparado con mucho cariño! 👨‍🍳`,
    "enviado": `Tu pedido está en camino. ¡Pronto lo disfrutarás! 🚚`,
    "entregado": `Tu pedido ha sido entregado. ¡Buen provecho! 😋`,
    "cancelado": `Tu pedido #${order.numero_pedido} ha sido cancelado.`,
  };

  return {
    title: statusTitles[status] || "Actualización de Pedido",
    body: statusBodies[status] || `Tu pedido ha sido actualizado a:
${status}`,
    icon: "ic_notification",
    sound: "default",
  };
}
```

6.3.5 Variables de Entorno

```
# ======================================
# VARIABLES DE ENTORNO - EDGE FUNCTION
# ======================================

# Supabase (auto-configuradas)
SUPABASE_URL=https://tu-proyecto.supabase.co
SUPABASE_ANON_KEY=tu-clave-anonima

# SendGrid (Email)
SENDGRID_API_KEY=SG.xxxxxxxxxxxxxxxxxxxx

# Twilio (SMS)
TWILIO_ACCOUNT_SID=ACxxxxxxxxxxxxxxxxx
TWILIO_AUTH_TOKEN=xxxxxxxxxxxxxxx
TWILIO_PHONE_NUMBER=+1234567890
```

```
# Firebase Cloud Messaging (Push)
FCM_SERVER_KEY=AAAAxxxxxxxxxxxxxxxxxxxxxx
```

6.3.6 Desplegar la Edge Function

```
# Desplegar a Supabase
supabase functions deploy notify_order_status

# Establecer variables de entorno (secrets)
supabase secrets set SENDGRID_API_KEY=SG.xxxxxxx
supabase secrets set TWILIO_ACCOUNT_SID=ACxxxxxx
supabase secrets set TWILIO_AUTH_TOKEN=xxxxxxxx
supabase secrets set TWILIO_PHONE_NUMBER=+1234567890
supabase secrets set FCM_SERVER_KEY=AAAAxxxxxxx

# Listar funciones desplegadas
supabase functions list

# Ver logs en tiempo real
supabase functions logs notify_order_status
```

6.3.7 Invocar la Edge Function desde Flutter

```
import 'package:supabase_flutter/supabase_flutter.dart';

class PedidoViewModel extends ChangeNotifier {
  final _supabase = Supabase.instance.client;

  /// Actualizar estado del pedido y enviar notificaciones
  Future<bool> actualizarEstadoConNotificacion(
    int pedidoId,
    String nuevoEstado,
    int clienteId,
  ) async {
    try {
      // 1. Actualizar estado en base de datos
      await _supabase
          .from('pedidos')
          .update({'estado': nuevoEstado})
          .eq('id', pedidoId);

      // 2. Invocar Edge Function para notificaciones
      final response = await _supabase.functions.invoke(
        'notify_order_status',
        body: {
          'order_id': pedidoId,
          'user_id': clienteId.toString(),
          'status': nuevoEstado,
          'notification_types': ['email', 'push'], // Canales deseados
        },
      );

      if (response.status == 200) {
        print('✅ Notificaciones enviadas correctamente');
        return true;
      } else {
        print('⚠️ Error al enviar notificaciones: ${response.data}'); return
        false;
```

```
      }
    } catch (e) {
      print('❌ Error: $e');
      return false;
    }
  }
}
```

6.3.8 Tabla de Logs de Notificaciones (Opcional)

```
-- Crear tabla de logs
CREATE TABLE notification_logs (
  id BIGSERIAL PRIMARY KEY,
  order_id INTEGER REFERENCES pedidos(id),
  user_id UUID REFERENCES usuarios(id),
  status TEXT NOT NULL,
  channels_sent TEXT[] NOT NULL,
  successful_count INTEGER DEFAULT 0,
  failed_count INTEGER DEFAULT 0,
  timestamp TIMESTAMPTZ DEFAULT NOW()
);

-- Índices
CREATE INDEX idx_notification_logs_order ON notification_logs(order_id);
CREATE INDEX idx_notification_logs_user ON notification_logs(user_id);
CREATE INDEX idx_notification_logs_timestamp ON notification_logs(timestamp);
```

6.3.9 Monitoreo y Troubleshooting

Problemas comunes:

Error	Causa	Solución
`401 Unauthorized`	API keys incorrectas	Verificar secrets en Supabase
`CORS error`	Headers CORS faltantes	Agregar `corsHeaders` en respuesta
`Timeout`	Función tarda más de 60s	Optimizar código o dividir en funciones
Email no se envía	SendGrid no configurado	Verificar `SENDGRID_API_KEY`
SMS no se envía	Twilio no configurado	Verificar credenciales Twilio

7 SERVICIOS Y LÓGICA DE NEGOCIO

7.1 Auth Service Completo

7.1.1 Autenticación de usuarios (login, registro, recuperación de contraseña).

```
// services/auth_service.dart
import 'package:flutter/material.dart';
import '../data/dao/usuario_dao_supabase.dart';
import '../data/models/usuario.dart';

class AuthService extends ChangeNotifier {
  final UsuarioDaoSupabase _usuarioDao = UsuarioDaoSupabase();
  Usuario? _usuarioActual;

  Usuario? get usuario => _usuarioActual;
  Usuario? get usuarioActual => _usuarioActual;
```

```
bool get estaAutenticado => _usuarioActual != null;

// Iniciar sesión
Future<ResultadoAuth> iniciarSesion(String email, String password) async {
  try {
    final usuario = await _usuarioDao.login(email, password);

    if (usuario == null) {
      return ResultadoAuth(
        exitoso: false,
        mensaje: 'Email o contraseña incorrectos',
      );
    }

    if (!usuario.activo) {
      return ResultadoAuth(
        exitoso: false,
        mensaje: 'Usuario inactivo. Contacte al administrador',
      );
    }

    _usuarioActual = usuario;
    notifyListeners();

    return ResultadoAuth(
      exitoso: true,
      mensaje: 'Bienvenido ${usuario.nombre}',
      usuario: usuario,
    );
  } catch (e) {
    return ResultadoAuth(
      exitoso: false,
      mensaje: 'Error al iniciar sesión: ${e.toString()}',
    );
  }
}

// Registrar usuario
Future<ResultadoAuth> registrar(Usuario usuario) async {
  try {
    final emailExiste = await _usuarioDao.emailExiste(usuario.email);

    if (emailExiste) {
      return ResultadoAuth(
        exitoso: false,
        mensaje: 'El email ya está registrado',
      );
    }

    final nuevoUsuario = Usuario(
      nombre: usuario.nombre,
      email: usuario.email,
      password: usuario.password,
      rol: usuario.rol,
```

```
        telefono: usuario.telefono,
        direccion: usuario.direccion,
        fechaCreacion: DateTime.now(),
        activo: true,
      );

      final id = await _usuarioDao.crear(nuevoUsuario);

      if (id > 0) {
        return ResultadoAuth(
          exitoso: true,
          mensaje: 'Usuario registrado exitosamente',
        );
      }

      return ResultadoAuth(
        exitoso: false,
        mensaje: 'Error al registrar usuario',
      );
    } catch (e) {
      return ResultadoAuth(
        exitoso: false,
        mensaje: 'Error al registrar: ${e.toString()}',
      );
    }
  }

  // Cerrar sesión
  void cerrarSesion() {
    _usuarioActual = null;
    notifyListeners();
  }

  // Cambiar contraseña - ahora acepta (id, passwordActual, passwordNueva) o
(passwordActual, passwordNueva)
  Future<ResultadoAuth> cambiarPassword(
    dynamic param1,
    String param2, [
    String? param3,
  ]) async {
    try {
      int? userId;
      String passwordActual;
      String passwordNueva;

      // Detectar qué formato se usa
      if (param3 != null) {
        // Formato: (id, passwordActual, passwordNueva)
        userId = param1 as int;
        passwordActual = param2;
        passwordNueva = param3;
      } else {
        // Formato antiguo: (passwordActual, passwordNueva)
        if (_usuarioActual == null) {
          return ResultadoAuth(
```

```
            exitoso: false,
            mensaje: 'No hay sesión activa',
          );
        }
        userId = _usuarioActual!.id!;
        passwordActual = param1 as String;
        passwordNueva = param2;
      }

      final esValida = await _usuarioDao.validarCredenciales(
        userId,
        passwordActual,
      );

      if (!esValida) {
        return ResultadoAuth(
          exitoso: false,
          mensaje: 'La contraseña actual es incorrecta',
        );
      }

      final resultado = await _usuarioDao.cambiarPassword(userId,
passwordNueva);

      if (resultado) {
        // Recargar usuario si es el actual
        if (_usuarioActual != null && _usuarioActual!.id == userId) {
          _usuarioActual = await _usuarioDao.obtenerPorId(userId);
          notifyListeners();
        }

        return ResultadoAuth(
          exitoso: true,
          mensaje: 'Contraseña actualizada exitosamente',
        );
      }

      return ResultadoAuth(
        exitoso: false,
        mensaje: 'Error al cambiar contraseña',
      );
    } catch (e) {
      return ResultadoAuth(
        exitoso: false,
        mensaje: 'Error al cambiar contraseña: ${e.toString()}',
      );
    }
  }

  // Actualizar perfil
  Future<ResultadoAuth> actualizarPerfil(Usuario usuario) async {
    try {
      if (_usuarioActual == null) {
        return ResultadoAuth(
          exitoso: false,
```

```
        mensaje: 'No hay sesión activa',
      );
    }

    if (usuario.email != _usuarioActual!.email) {
      final emailExiste = await _usuarioDao.emailExiste(
        usuario.email,
        excluyendoId: _usuarioActual!.id,
      );

      if (emailExiste) {
        return ResultadoAuth(
          exitoso: false,
          mensaje: 'El email ya está en uso',
        );
      }
    }

    // Mantener la contraseña original
    final usuarioParaActualizar = usuario.copyWith(
      password: _usuarioActual!.password,
    );

    final resultado = await _usuarioDao.actualizar(usuarioParaActualizar);
    if (resultado) {
      _usuarioActual = usuarioParaActualizar;
      notifyListeners();

      return ResultadoAuth(
        exitoso: true,
        mensaje: 'Perfil actualizado exitosamente',
        usuario: usuarioParaActualizar,
      );
    }

    return ResultadoAuth(
      exitoso: false,
      mensaje: 'Error al actualizar perfil',
    );
  } catch (e) {
    return ResultadoAuth(
      exitoso: false,
      mensaje: 'Error al actualizar perfil: ${e.toString()}',
    );
  }
}

// Recargar datos del usuario actual
Future<void> recargarUsuario() async {
  if (_usuarioActual != null && _usuarioActual!.id != null) {
    _usuarioActual = await _usuarioDao.obtenerPorId(_usuarioActual!.id!);
    notifyListeners();
  }
}
```

```
  // Verificar permisos
  bool tienePermiso(String permiso) {
    if (_usuarioActual == null) return false;

    switch (permiso) {
      case 'admin':
        return _usuarioActual!.rol == 'admin';
      case 'vendedor':
        return _usuarioActual!.rol == 'admin' ||
               _usuarioActual!.rol == 'vendedor';
      case 'cliente':
        return true;
      default:
        return false;
    }
  }

  bool get esAdmin => _usuarioActual?.rol == 'admin';
  bool get esVendedor => _usuarioActual?.rol == 'vendedor' || esAdmin;
  bool get esCliente => _usuarioActual?.rol == 'cliente';
  String? get rolActual => _usuarioActual?.rol;
  int? get idUsuarioActual => _usuarioActual?.id;
  String? get nombreUsuarioActual => _usuarioActual?.nombre;
}

class ResultadoAuth {
  final bool exitoso;
  final String mensaje;
  final Usuario? usuario;

  ResultadoAuth({
    required this.exitoso,
    required this.mensaje,
    this.usuario,
  });
}
```

7.2 Carrito Service Completo

7.2.1 Gestión del carrito de compras (agregar, eliminar, actualizar cantidades).

```
import 'package:flutter/foundation.dart';
import '../data/models/producto.dart';

class ItemCarrito {
  final Producto producto;
  int cantidad;

  ItemCarrito({
    required this.producto,
    this.cantidad = 1,
  });

  double get subtotal => producto.precioVenta * cantidad;
}

/// ✅ CarritoService - Lógica de negocio pura sin navegación
/// La navegación debe manejarse en las pantallas (UI)
class CarritoService extends ChangeNotifier {
  final List<ItemCarrito> _items = [];

  // Getters
  List<ItemCarrito> get items => List.unmodifiable(_items);

  int get cantidadTotal => _items.fold(0, (sum, item) => sum +
item.cantidad);

  // Alias para compatibilidad con código existente
  int get cantidadItems => cantidadTotal;

  double get total => _items.fold(0.0, (sum, item) => sum + item.subtotal);

  bool get estaVacio => _items.isEmpty;

  // MÉTODO PRINCIPAL: Agregar producto al carrito
  void agregarProducto(Producto producto, {int cantidad = 1}) {
    // Validación crítica: verificar que el producto tenga ID
    if (producto.id == null) {
      debugPrint('❌ Error: Producto sin ID no puede ser agregado al
carrito');
      return;
    }

    // Validar cantidad positiva
    if (cantidad <= 0) {
      debugPrint('❌ Error: Cantidad debe ser mayor a 0');
      return;
    }
```

```
    // Validar stock disponible
    if (cantidad > producto.stock) {
      debugPrint('❌ Error: Stock insuficiente. Disponible:
${producto.stock}');
      return;
    }

    final index = _items.indexWhere((item) => item.producto.id ==
producto.id);

    if (index >= 0) {
      // Producto ya existe - verificar stock antes de aumentar
      final nuevaCantidad = _items[index].cantidad + cantidad;

      if (nuevaCantidad > producto.stock) {
        debugPrint('❌ Stock insuficiente. Máximo: ${producto.stock}');
        return;
      }

      _items[index].cantidad = nuevaCantidad;
      debugPrint('✅ Cantidad actualizada: ${producto.nombre}
x$nuevaCantidad');
    } else {
      // Producto nuevo - agregar al carrito
      _items.add(ItemCarrito(producto: producto, cantidad: cantidad));
      debugPrint('✅ Producto agregado: ${producto.nombre} x$cantidad');
    }

    notifyListeners();
  }

  // Eliminar producto del carrito
  void eliminarProducto(int productoId) {
    final cantidadInicial = _items.length;
    _items.removeWhere((item) => item.producto.id == productoId);

    if (_items.length < cantidadInicial) {
      notifyListeners();
      debugPrint('🗑️ Producto eliminado del carrito (ID: $productoId)');
    }
  }

  // Actualizar cantidad de un producto
  void actualizarCantidad(int productoId, int nuevaCantidad) {
    // Si la cantidad es 0 o negativa, eliminar el producto
    if (nuevaCantidad <= 0) {
      eliminarProducto(productoId);
      return;
    }

    final index = _items.indexWhere((item) => item.producto.id ==
productoId);

    if (index >= 0) {
```

```
      // Validar stock disponible
      if (nuevaCantidad > _items[index].producto.stock) {
        debugPrint('❌ Stock insuficiente. Disponible:
${_items[index].producto.stock}');
        return;
      }

      _items[index].cantidad = nuevaCantidad;
      notifyListeners();
      debugPrint('✅ Cantidad actualizada a: $nuevaCantidad');
    } else {
      debugPrint('❌ Producto no encontrado en el carrito');
    }
  }

  // Incrementar cantidad en 1
  void incrementarCantidad(int productoId) {
    final index = _items.indexWhere((item) => item.producto.id ==
productoId);

    if (index >= 0) {
      final item = _items[index];

      // Validar stock antes de incrementar
      if (item.cantidad >= item.producto.stock) {
        debugPrint('❌ Stock máximo alcanzado: ${item.producto.stock}');
        return;
      }

      item.cantidad++;
      notifyListeners();
      debugPrint('✅ Cantidad incrementada: ${item.cantidad}');
    }
  }

  // Decrementar cantidad en 1
  void decrementarCantidad(int productoId) {
    final index = _items.indexWhere((item) => item.producto.id ==
productoId);

    if (index >= 0) {
      if (_items[index].cantidad > 1) {
        _items[index].cantidad--;
        notifyListeners();
        debugPrint('✅ Cantidad decrementada: ${_items[index].cantidad}');
      } else {
        // Si la cantidad es 1, eliminar el producto
        eliminarProducto(productoId);
      }
    }
  }

  // Vaciar todo el carrito
```

```
  void vaciar() {
    if (_items.isNotEmpty) {
      _items.clear();
      notifyListeners();
      debugPrint('🗑️ Carrito vaciado completamente');
    }
  }

  // Verificar si un producto está en el carrito
  bool contieneProducto(int productoId) {
    return _items.any((item) => item.producto.id == productoId);
  }

  // Alias para compatibilidad con código existente
  bool tieneProducto(int productoId) => contieneProducto(productoId);

  // Obtener cantidad de un producto específico
  int getCantidadProducto(int productoId) {
    try {
      final item = _items.firstWhere((item) => item.producto.id ==
productoId);
      return item.cantidad;
    } catch (e) {
      return 0;
    }
  }

  // Obtener un item específico
  ItemCarrito? getItem(int productoId) {
    try {
      return _items.firstWhere((item) => item.producto.id == productoId);
    } catch (e) {
      return null;
    }
  }

  // Validar que todos los productos tengan stock suficiente
  bool validarStock() {
    for (var item in _items) {
      if (item.cantidad > item.producto.stock) {
        debugPrint('❌ Stock insuficiente para: ${item.producto.nombre}');
        return false;
      }
    }
    return true;
  }

  // Obtener lista de productos con stock insuficiente
  List<ItemCarrito> getProductosStockInsuficiente() {
    return _items
        .where((item) => item.cantidad > item.producto.stock)
        .toList();
  }
```

```
  // Obtener mensaje de error específico de stock
  String? obtenerErrorStock() {
    for (var item in _items) {
      if (item.cantidad > item.producto.stock) {
        return 'Stock insuficiente para "${item.producto.nombre}". '
               'Disponible: ${item.producto.stock}, Solicitado:
${item.cantidad}';
      }
    }
    return null;
  }

  // Verificar si hay productos con stock insuficiente
  bool tieneProductosConStockInsuficiente() {
    return _items.any((item) => item.cantidad > item.producto.stock);
  }

  // Obtener información del carrito para depuración
  void imprimirResumen() {
    debugPrint('📊 === RESUMEN DEL CARRITO ===');
    debugPrint('Total de items: ${_items.length}');
    debugPrint('Cantidad total: $cantidadTotal');
    debugPrint('Total: \$${total.toStringAsFixed(2)}');

    for (var item in _items) {
      debugPrint('  • ${item.producto.nombre} x${item.cantidad} =
\$${item.subtotal.toStringAsFixed(2)}');
    }
    debugPrint('================================');
  }
}
```

7.3 Email Service

7.3.1 Envío de emails con plantillas HTML personalizadas.

```
import 'package:mailer/mailer.dart';
import 'package:mailer/smtp_server.dart';
import 'package:flutter/foundation.dart';

/// 📧 EmailService - Servicio para enviar emails de recuperación de
contraseña
///
/// Este servicio utiliza SMTP de Gmail para enviar emails de reset de
contraseña.
///
/// CONFIGURACIÓN NECESARIA:
/// 1. Crear cuenta de Gmail
/// 2. Habilitar "Contraseñas de aplicación" en Google
(https://myaccount.google.com/apppasswords)
/// 3. Configurar las credenciales abajo
///
/// ⚠️ IMPORTANTE: Configurar estas credenciales ANTES de usar
class EmailService {
```

```
  // 🔧 CONFIGURAR AQUÍ TUS CREDENCIALES DE GMAIL
  static const String _senderEmail = 'winstonalvarado94@gmail.com'; // ➤
Cambiar por tu email
  static const String _senderPassword = 'phfm hrlt mccp axzt'; // ➤
Contraseña de aplicación (16 caracteres)
  static const String _senderName = 'AztecaFest Sistema';
  /// Enviar email de recuperación de contraseña
  ///
  /// Parámetros:
  /// - [recipientEmail]: Email del usuario
  /// - [resetLink]: Link para resetear contraseña (incluye token)
  /// - [userName]: Nombre del usuario
  ///
  /// Retorna: true si se envió correctamente, false si hubo error
  Future<bool> enviarEmailRecuperacion({
    required String recipientEmail,
    required String resetLink,
    required String userName,
  }) async {
    try {
      debugPrint('📧 Intentando enviar email de recuperación a:
$recipientEmail');

      // Validar configuración
      // Solo simular si las credenciales son claramente placeholders
      if (_senderEmail.isEmpty || _senderPassword.isEmpty ||
          _senderEmail == 'tu_email@gmail.com' ||
          _senderPassword == 'xxxx xxxx xxxx xxxx') {
        debugPrint('⚠️ ADVERTENCIA: Credenciales de email NO configuradas');
        debugPrint('📧 Simulando envío de email a: $recipientEmail');
        debugPrint('🔗 Link de reset: $resetLink');
        debugPrint('⚠️ Configura _senderEmail y _senderPassword en
email_service.dart');
        return true; // Retornar true para permitir flujo de desarrollo
      }

      // Si las credenciales están configuradas pero son las mismas que los
valores por defecto,
      // asumir que el usuario las configuró intencionalmente
      debugPrint('✅ Credenciales configuradas. Procediendo con envío
real.');

      // Crear mensaje
      final message = Message()
        ..from = Address(_senderEmail, _senderName)
        ..recipients.add(recipientEmail)
        ..subject = '🔐 Recupera tu contraseña - AztecaFest'
        ..html = _buildHtmlEmail(userName, resetLink);

      // Configurar servidor SMTP de Gmail explícitamente
      final smtpServer = SmtpServer(
        'smtp.gmail.com',
        port: 587,
```

```
        username: _senderEmail,
        password: _senderPassword,
        ssl: false,
        allowInsecure: true,
      );

      debugPrint('🔄 Conectando a servidor SMTP de Gmail...');
      debugPrint('📤 Enviando email a: $recipientEmail');

      // Enviar email REAL
      final sendReport = await send(message, smtpServer);

      debugPrint('✅ Email enviado correctamente');
      debugPrint('📊 Reporte: ${sendReport.toString()}');
      return true;
    } catch (e, stackTrace) {
      debugPrint('❌ Error al enviar email: $e');
      debugPrint('📄 Stack trace: $stackTrace');
      debugPrint('💡 Posibles causas y soluciones:');
      debugPrint('   1. Verifica que el email y contraseña sean correctos');
      debugPrint('   2. Usa una "Contraseña de aplicación" de Google (NO tu
contraseña normal)');
      debugPrint('   3. Asegúrate de que la verificación en dos pasos está
activada en tu cuenta de Google');
      debugPrint('   4. Verifica tu conexión a Internet');
      debugPrint('   5. Comprueba que el puerto 587 no está bloqueado por tu
firewall');

      // Verificar credenciales específicamente
      if (_senderEmail.contains('@') == false || _senderEmail.contains('.')
== false) {
        debugPrint('⚠️ El email del remitente no parece válido:
$_senderEmail');
      }

      if (_senderPassword.length != 19 || _senderPassword.contains(' ') ==
false) {
        debugPrint('⚠️ La contraseña no tiene el formato esperado de
contraseña de aplicación (debe ser xxxx xxxx xxxx xxxx)');
      }

      return false;
    }
  }

  /// Construir HTML del email con estilo profesional
  static String _buildHtmlEmail(String userName, String resetLink) {
    return '''
    <!DOCTYPE html>
    <html>
      <head>
        <meta charset="UTF-8">
        <style>
```

```
body {
  font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
  line-height: 1.6;
  color: #333;
}
.container {
  max-width: 600px;
  margin: 0 auto;
  background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
  padding: 20px;
  border-radius: 10px;
}
.content {
  background: white;
  padding: 30px;
  border-radius: 8px;
  box-shadow: 0 4px 6px rgba(0,0,0,0.1);
}
.header {
  text-align: center;
  margin-bottom: 30px;
}
.logo {
  font-size: 28px;
  font-weight: bold;
  color: #E74C3C;
  margin-bottom: 10px;
}
.title {
  color: #2C3E50;
  margin-bottom: 20px;
}
.button {
  display: inline-block;
  padding: 15px 30px;
  background: linear-gradient(135deg, #E74C3C, #FF6B6B);
  color: white !important;
  text-decoration: none;
  border-radius: 25px;
  font-weight: bold;
  margin: 20px 0;
  box-shadow: 0 4px 15px rgba(231, 76, 60, 0.3);
}
.warning {
  background: #FFF3CD;
  border-left: 4px solid #FFC107;
  padding: 15px;
  margin: 20px 0;
  border-radius: 4px;
}
.footer {
  text-align: center;
  margin-top: 30px;
  color: #6c757d;
```

```
            font-size: 12px;
          }
        </style>
      </head>
      <body>
        <div class="container">
          <div class="content">
            <div class="header">
              <div class="logo">🫔 AztecaFest</div>
              <h1 class="title">Recupera tu contraseña</h1>
            </div>

            <p>¡Hola <strong>$userName</strong>!</p>

            <p>Hemos recibido una solicitud para restablecer tu contraseña en
AztecaFest.</p>

            <p>Para crear una nueva contraseña, haz clic en el siguiente
botón:</p>

            <div style="text-align: center;">
              <a href="$resetLink" class="button">Restablecer Contraseña</a>
            </div>

            <p>Si no puedes hacer clic en el botón, copia y pega este enlace
en tu navegador:</p>
            <p style="word-break: break-all; color: #007bff;">$resetLink</p>

            <div class="warning">
              <p><strong>Importante:</strong></p>
              <ul>
                <li>Este enlace expira en 24 horas</li>
                <li>No compartas este email con nadie</li>
                <li>Si no solicitaste este cambio, ignora este email</li>
              </ul>
            </div>

            <p>Gracias por usar AztecaFest. ¡Disfruta de nuestros deliciosos
tacos!</p>

            <div class="footer">
              <p>© 2025 AztecaFest. Todos los derechos reservados.</p>
              <p>Este email fue enviado automáticamente. No responda a este
correo.</p>
            </div>
          </div>
        </div>
      </body>
    </html>
    ''';
  }

  /// Validar email - versión simplificada
  static bool validarEmail(String email) {
```

```
    if (email.isEmpty) return false;
    // Verificar que tenga @ y . y que no estén al inicio o final
    return email.contains('@') &&
           email.contains('.') &&
           email.indexOf('@') > 0 &&
           email.lastIndexOf('.') > email.indexOf('@') + 1 &&
           email.lastIndexOf('.') < email.length - 1;
  }

  /// Enviar email genérico con asunto y cuerpo HTML personalizado
  ///
  /// Parámetros:
  /// - [destinatario]: Email del destinatario
  /// - [asunto]: Asunto del email
  /// - [cuerpo]: Cuerpo en HTML del email
  ///
  /// Retorna: true si se envió correctamente
  Future<bool> enviarEmail({
    required String destinatario,
    required String asunto,
    required String cuerpo,
  }) async {
    try {
      debugPrint('📧 Intentando enviar email a: $destinatario');
      debugPrint('Asunto: $asunto');

      // Validar configuración
      // Solo simular si las credenciales son claramente placeholders
      if (_senderEmail.isEmpty || _senderPassword.isEmpty ||
          _senderEmail == 'tu_email@gmail.com' ||
          _senderPassword == 'xxxx xxxx xxxx xxxx') {
        debugPrint('⚠️ ADVERTENCIA: Credenciales de email NO configuradas');
        debugPrint('📧 Simulando envío de email a: $destinatario');
        debugPrint('⚠️ Configura _senderEmail y _senderPassword en
email_service.dart');
        return true; // Retornar true para permitir flujo de desarrollo
      }

      // Si las credenciales están configuradas pero son las mismas que los
valores por defecto,
      // asumir que el usuario las configuró intencionalmente
      debugPrint('✅ Credenciales configuradas. Procediendo con envío
real.');

      // Crear mensaje
      final message = Message()
        ..from = Address(_senderEmail, _senderName)
        ..recipients.add(destinatario)
        ..subject = asunto
        ..html = cuerpo;

      // Configurar servidor SMTP de Gmail explícitamente
      final smtpServer = SmtpServer(
        'smtp.gmail.com',
```

```
        port: 587,
        username: _senderEmail,
        password: _senderPassword,
        ssl: false,
        allowInsecure: true,
      );

      // Enviar email
      await send(message, smtpServer);

      debugPrint('✅ Email enviado correctamente');
      return true;
    } catch (e, stackTrace) {
      debugPrint('❌ Error al enviar email: $e');
      debugPrint('📄 Stack trace: $stackTrace');
      debugPrint('💡 Posibles causas:');
      debugPrint('   1. Verifica tu conexión a Internet');
      debugPrint('   2. Comprueba que el puerto 587 no está bloqueado por tu
firewall');
      debugPrint('   3. Verifica que las credenciales sean correctas');

      // Verificar credenciales específicamente
      if (_senderEmail.contains('@') == false || _senderEmail.contains('.')
== false) {
        debugPrint('⚠️ El email del remitente no parece válido:
$_senderEmail');
      }

      if (_senderPassword.length != 19 || _senderPassword.contains(' ') ==
false) {
        debugPrint('⚠️ La contraseña no tiene el formato esperado de
contraseña de aplicación');
      }

      return false;
    }
  }
}
```

7.4 **Image Storage Service**

```
// data/services/image_storage_service.dart
import 'dart:io';
import 'package:path_provider/path_provider.dart';
import 'package:path/path.dart' as path;


class ImageStorageService {
  static const String _imageFolder = 'product_images';
```

```
  /// Obtiene el directorio donde se guardarán las
imágenes
  Future<Directory> _getImageDirectory() async {
    final appDir = await
getApplicationDocumentsDirectory();
    final imageDir =
Directory('${appDir.path}/$_imageFolder');

    if (!await imageDir.exists()) {
      await imageDir.create(recursive: true);
    }

    return imageDir;
  }

  Future<String> getImageDirectoryPath() async {
    final dir = await _getImageDirectory();
    return dir.path;
  }

  /// Guarda una imagen y retorna la ruta donde se guardó
  Future<String?> guardarImagen(File imagen, String
codigoProducto) async {
    try {
      final imageDir = await _getImageDirectory();
      final extension = path.extension(imagen.path);
      final timestamp =
DateTime.now().millisecondsSinceEpoch;
      final nombreArchivo =
'${codigoProducto}_$timestamp$extension';
      final rutaDestino =
```

```
'${imageDir.path}/$nombreArchivo';

      // Copiar la imagen al directorio de la app
      await imagen.copy(rutaDestino);

      return rutaDestino;
    } catch (e) {
      print('Error al guardar imagen: $e');
      return null;
    }
  }

  /// Elimina una imagen del almacenamiento
  Future<bool> eliminarImagen(String rutaImagen) async {
    try {
      final archivo = File(rutaImagen);
      if (await archivo.exists()) {
        await archivo.delete();
        return true;
      }
      return false;
    } catch (e) {
      print('Error al eliminar imagen: $e');
      return false;
    }
  }

  /// Actualiza una imagen (elimina la anterior y guarda
la nueva)
  Future<String?> actualizarImagen({
```

```
    required File nuevaImagen,
    required String codigoProducto,
    String? rutaImagenAnterior,
  }) async {
    try {
      // Eliminar imagen anterior si existe
      if (rutaImagenAnterior != null &&
rutaImagenAnterior.isNotEmpty) {
        await eliminarImagen(rutaImagenAnterior);
      }

      // Guardar nueva imagen
      return await guardarImagen(nuevaImagen,
codigoProducto);
    } catch (e) {
      print('Error al actualizar imagen: $e');
      return null;
    }
  }

  /// Verifica si una imagen existe
  Future<bool> existeImagen(String rutaImagen) async {
    try {
      final archivo = File(rutaImagen);
      return await archivo.exists();
    } catch (e) {
      return false;
    }
  }

  /// Obtiene el tamano de una imagen en bytes
```

```
  Future<int?> obtenerTamanoImagen(String rutaImagen)
async {
    try {
      final archivo = File(rutaImagen);
      if (await archivo.exists()) {
        return await archivo.length();
      }
      return null;
    } catch (e) {
      print('Error al obtener tamano de imagen: $e');
      return null;
    }
  }

  /// Lista todas las imágenes guardadas
  Future<List<String>> listarImagenes() async {
    try {
      final imageDir = await _getImageDirectory();
      final archivos = await imageDir.list().toList();

      return archivos
          .whereType<File>()
          .map((f) => f.path)
          .toList();
    } catch (e) {
      print('Error al listar imagenes: $e');
      return [];
    }
  }
```

```
  /// Limpia imágenes huérfanas (que no están en uso)
  Future<void> limpiarImagenesHuerfanas(List<String>
rutasEnUso) async {
    try {
      final todasLasImagenes = await listarImagenes();

      for (final ruta in todasLasImagenes) {
        if (!rutasEnUso.contains(ruta)) {
          await eliminarImagen(ruta);
          print('Imagen huerfana eliminada: $ruta');
        }
      }
    } catch (e) {
      print('Error al limpiar imagenes huerfanas: $e');
    }
  }

  /// Obtiene el tamano total usado por las imágenes
  Future<int> obtenerEspacioUsado() async {
    try {
      final imagenes = await listarImagenes();
      int totalBytes = 0;

      for (final ruta in imagenes) {
        final tamano = await obtenerTamanoImagen(ruta);
        if (tamano != null) {
          totalBytes += tamano;
        }
      }
```

```
      return totalBytes;
    } catch (e) {
      print('Error al calcular espacio usado: $e');
      return 0;
    }
  }

  /// Formatea el tamaño en bytes a una cadena legible
(KB, MB, etc.)
  String formatearTamano(int bytes) {
    if (bytes < 1024) {
      return '$bytes B';
    } else if (bytes < 1024 * 1024) {
      return '${(bytes / 1024).toStringAsFixed(2)} KB';
    } else if (bytes < 1024 * 1024 * 1024) {
      return '${(bytes / (1024 *
1024)).toStringAsFixed(2)} MB';
    } else {
      return '${(bytes / (1024 * 1024 *
1024)).toStringAsFixed(2)} GB';
    }
  }

  /// Obtiene información estadística de las imágenes
  Future<Map<String, dynamic>> obtenerEstadisticas() async
{
    try {
      final imagenes = await listarImagenes();
      final espacioUsado = await obtenerEspacioUsado();

      return {
```

```
          'totalImagenes': imagenes.length,
          'espacioUsadoBytes': espacioUsado,
          'espacioUsadoFormateado':
   formatearTamano(espacioUsado),
        };
      } catch (e) {
        print('Error al obtener estadisticas: $e');
        return {
          'totalImagenes': 0,
          'espacioUsadoBytes': 0,
          'espacioUsadoFormateado': '0 B',
        };
      }
    }
  }
```

7.5 Migration Service

```
// lib/services/migration_service.dart
import 'dart:convert';
import 'dart:io';
import 'package:path_provider/path_provider.dart';
import 'package:sqflite/sqflite.dart';
import '../data/database_helper.dart';

class MigrationService {
  static final MigrationService instance = MigrationService._init();
  MigrationService._init();

  /// Exporta TODOS los datos de la base de datos a un archivo JSON
  Future<File> exportarDatos({String? nombreArchivo}) async {
    try {
      final db = await DatabaseHelper.instance.database;

      // Obtener todos los datos de todas las tablas
      final Map<String, dynamic> datosExportacion = {
        'version': 6,
        'fecha_exportacion': DateTime.now().toIso8601String(),
        'datos': {
          'usuarios': await db.query('usuarios'),
          'productos': await db.query('productos'),
          'ventas': await db.query('ventas'),
          'detalle_ventas': await db.query('detalle_ventas'),
```

```
          'pedidos': await db.query('pedidos'),
          'detalle_pedidos': await db.query('detalle_pedidos'),
        },
        'metadatos': {
          'total_usuarios': await _contarRegistros(db, 'usuarios'),
          'total_productos': await _contarRegistros(db, 'productos'),
          'total_ventas': await _contarRegistros(db, 'ventas'),
          'total_pedidos': await _contarRegistros(db, 'pedidos'),
        }
      };

      // Convertir a JSON
      final jsonString = JsonEncoder.withIndent('
').convert(datosExportacion);

      // Guardar en archivo
      final directorio = await getApplicationDocumentsDirectory();
      final nombre = nombreArchivo ??
'backup_aztecafest_${DateTime.now().millisecondsSinceEpoch}.json';
      final archivo = File('${directorio.path}/$nombre');

      await archivo.writeAsString(jsonString);

      print('✅ Exportación exitosa: ${archivo.path}');
      print('📊 Total de registros exportados:');
      print('   - Usuarios:
${datosExportacion['metadatos']['total_usuarios']}');
      print('   - Productos:
${datosExportacion['metadatos']['total_productos']}');
      print('   - Ventas: ${datosExportacion['metadatos']['total_ventas']}');
      print('   - Pedidos:
${datosExportacion['metadatos']['total_pedidos']}');

      return archivo;
    } catch (e) {
      print('❌ Error al exportar datos: $e');
      rethrow;
    }
  }

  /// Importa datos desde un archivo JSON
  Future<bool> importarDatos(String rutaArchivo, {bool reemplazarExistentes =
false}) async {
    try {
      final archivo = File(rutaArchivo);

      if (!await archivo.exists()) {
        throw Exception('El archivo no existe: $rutaArchivo');
      }

      // Leer archivo JSON
      final jsonString = await archivo.readAsString();
      final Map<String, dynamic> datosImportados = json.decode(jsonString);
```

```
      // Validar versión
      final versionImportada = datosImportados['version'] as int?;
      print('🎁 Versión de datos importados: $versionImportada');

      final db = await DatabaseHelper.instance.database;

      // Si se desea reemplazar datos existentes, limpiar tablas
      if (reemplazarExistentes) {
        await _limpiarTablas(db);
      }

      // Importar datos en orden correcto (respetando foreign keys)
      final datos = datosImportados['datos'] as Map<String, dynamic>;

      await db.transaction((txn) async {
        // 1. Usuarios (no tiene dependencias)
        await _importarTabla(txn, 'usuarios', datos['usuarios'] as List);

        // 2. Productos (no tiene dependencias)
        await _importarTabla(txn, 'productos', datos['productos'] as List);

        // 3. Ventas (depende de usuarios)
        await _importarTabla(txn, 'ventas', datos['ventas'] as List);

        // 4. Detalle ventas (depende de ventas y productos)
        await _importarTabla(txn, 'detalle_ventas', datos['detalle_ventas']
as List);

        // 5. Pedidos (depende de usuarios)
        await _importarTabla(txn, 'pedidos', datos['pedidos'] as List);

        // 6. Detalle pedidos (depende de pedidos y productos)
        await _importarTabla(txn, 'detalle_pedidos', datos['detalle_pedidos']
as List);
      });

      print('✅ Importación completada exitosamente');
      print('📊 Resumen:');
      print('   - Usuarios importados: ${(datos['usuarios'] as
List).length}');
      print('   - Productos importados: ${(datos['productos'] as
List).length}');
      print('   - Ventas importadas: ${(datos['ventas'] as List).length}');
      print('   - Pedidos importados: ${(datos['pedidos'] as List).length}');

      return true;
    } catch (e) {
      print('❌ Error al importar datos: $e');
      return false;
    }
  }

  /// Importa tabla específica
  Future<void> _importarTabla(DatabaseExecutor db, String tabla, List
```

```
registros) async {
    int importados = 0;
    int omitidos = 0;

    for (var registro in registros) {
      try {
        final Map<String, dynamic> datos = Map<String,
dynamic>.from(registro);
        await db.insert(
          tabla,
          datos,
          conflictAlgorithm: ConflictAlgorithm.replace,
        );
        importados++;
      } catch (e) {
        print('⚠ Error al importar registro en $tabla: $e');
        omitidos++;
      }
    }

    print('   ✓ $tabla: $importados importados, $omitidos omitidos');
  }

  /// Limpia todas las tablas antes de importar
  Future<void> _limpiarTablas(Database db) async {
    print('🗑 Limpiando tablas existentes...');

    await db.transaction((txn) async {
      // Eliminar en orden inverso a las dependencias
      await txn.delete('detalle_pedidos');
      await txn.delete('detalle_ventas');
      await txn.delete('pedidos');
      await txn.delete('ventas');
      await txn.delete('productos');
      await txn.delete('usuarios');
    });

    print('✅ Tablas limpiadas');
  }

  /// Cuenta registros en una tabla
  Future<int> _contarRegistros(Database db, String tabla) async {
    final resultado = await db.rawQuery('SELECT COUNT(*) as count FROM
$tabla');
    return Sqflite.firstIntValue(resultado) ?? 0;
  }

  /// Obtiene información del último backup
  Future<Map<String, dynamic>?> obtenerInfoBackup(String rutaArchivo) async {
    try {
      final archivo = File(rutaArchivo);

      if (!await archivo.exists()) {
        return null;
```

```
      }

      final jsonString = await archivo.readAsString();
      final Map<String, dynamic> datos = json.decode(jsonString);

      return {
        'version': datos['version'],
        'fecha_exportacion': datos['fecha_exportacion'],
        'metadatos': datos['metadatos'],
        'tamano_bytes': await archivo.length(),
      };
    } catch (e) {
      print('❌ Error al leer info del backup: $e');
      return null;
    }
  }

  /// Lista todos los archivos de backup disponibles
  Future<List<FileSystemEntity>> listarBackups() async {
    try {
      final directorio = await getApplicationDocumentsDirectory();
      final archivos = directorio.listSync()
          .where((f) => f.path.contains('backup_aztecafest') &&
f.path.endsWith('.json'))
          .toList();

      // Ordenar por fecha (más reciente primero)
      archivos.sort((a, b) => b.path.compareTo(a.path));

      return archivos;
    } catch (e) {
      print('❌ Error al listar backups: $e');
      return [];
    }
  }

  /// Crea un backup automático con nombre basado en fecha
  Future<String> crearBackupAutomatico() async {
    final ahora = DateTime.now();
    final nombre = 'backup_aztecafest_'
        '${ahora.year}${ahora.month.toString().padLeft(2,
'0')}${ahora.day.toString().padLeft(2, '0')}_'
        '${ahora.hour.toString().padLeft(2,
'0')}${ahora.minute.toString().padLeft(2, '0')}.json';

    final archivo = await exportarDatos(nombreArchivo: nombre);
    return archivo.path;
  }

  /// Exporta solo datos específicos (por ejemplo, solo productos)
  Future<File> exportarTablaEspecifica(String nombreTabla, {String?
nombreArchivo}) async {
    try {
      final db = await DatabaseHelper.instance.database;
```

```
      final datos = await db.query(nombreTabla);

      final Map<String, dynamic> exportacion = {
        'tabla': nombreTabla,
        'fecha_exportacion': DateTime.now().toIso8601String(),
        'registros': datos,
        'total': datos.length,
      };

      final jsonString = JsonEncoder.withIndent('  ').convert(exportacion);

      final directorio = await getApplicationDocumentsDirectory();
      final nombre = nombreArchivo ??
'export_${nombreTabla}_${DateTime.now().millisecondsSinceEpoch}.json';
      final archivo = File('${directorio.path}/$nombre');

      await archivo.writeAsString(jsonString);

      print('✅ Exportación de $nombreTabla exitosa: ${archivo.path}');
      print('📊 Total de registros: ${datos.length}');

      return archivo;
    } catch (e) {
      print('❌ Error al exportar tabla $nombreTabla: $e');
      rethrow;
    }
  }

  /// Valida la integridad de un archivo de backup
  Future<bool> validarBackup(String rutaArchivo) async {
    try {
      final archivo = File(rutaArchivo);

      if (!await archivo.exists()) {
        print('❌ El archivo no existe');
        return false;
      }

      final jsonString = await archivo.readAsString();
      final datos = json.decode(jsonString);

      // Validar estructura básica
      if (!datos.containsKey('version') ||
          !datos.containsKey('datos') ||
          !datos.containsKey('metadatos')) {
        print('❌ Estructura de backup inválida');
        return false;
      }

      final datosTablas = datos['datos'] as Map<String, dynamic>;
      final tablasRequeridas = ['usuarios', 'productos', 'ventas',
'detalle_ventas', 'pedidos', 'detalle_pedidos'];
```

```
      for (var tabla in tablasRequeridas) {
        if (!datosTablas.containsKey(tabla)) {
          print('❌ Falta tabla: $tabla');
          return false;
        }
      }

      print('✅ Backup válido');
      return true;
    } catch (e) {
      print('❌ Error al validar backup: $e');
      return false;
    }
  }
}
```

7.6 Password Recovery Service

```
import 'package:flutter/foundation.dart';
import 'package:uuid/uuid.dart';
import 'package:crypto/crypto.dart';
import 'dart:convert';
import '../data/dao/usuario_dao_supabase.dart';
import '../data/models/usuario.dart';
import 'email_service.dart';

class PasswordRecoveryService extends ChangeNotifier {
  final UsuarioDaoSupabase _usuarioDao = UsuarioDaoSupabase();
  final EmailService _emailService = EmailService();

  final Map<String, Map<String, dynamic>> _resetTokens = {};

  String? _error;
  bool _isLoading = false;

  String? get error => _error;
  bool get isLoading => _isLoading;

  Future<bool> solicitarRecuperacion(String email) async {
    _isLoading = true;
    _error = null;
    notifyListeners();

    try {
      debugPrint('🔄 Iniciando solicitud de recuperación para: $email');

      if (!EmailService.validarEmail(email)) {
        _error = 'Email inválido';
        _isLoading = false;
        notifyListeners();
        return false;
      }
```

```
    final usuario = await _usuarioDao.obtenerPorEmail(email);
    if (usuario == null) {
      _error = null;
      _isLoading = false;
      notifyListeners();
      return true;
    }

    final token = const Uuid().v4();
    final ahora = DateTime.now();
    final expiracion = ahora.add(const Duration(hours: 24));

    _resetTokens[token] = {
      'email': email,
      'nombre': usuario.nombre,
      'expirado': false,
      'creado': ahora.toIso8601String(),
      'expira': expiracion.toIso8601String(),
    };

    debugPrint('✅ Token generado: $token');
    debugPrint('⏰ Expira en: $expiracion');

    final resetLink = 'aztecafest://reset-password?token=$token';

    final emailEnviado = await _emailService.enviarEmailRecuperacion(
      recipientEmail: email,
      resetLink: resetLink,
      userName: usuario.nombre,
    );

    if (!emailEnviado) {
      _error = 'No se pudo enviar el email. Intenta más tarde.';
      _isLoading = false;
      notifyListeners();
      return false;
    }

    _isLoading = false;
    notifyListeners();
    return true;
  } catch (e) {
    _error = 'Error al procesar solicitud: $e';
    debugPrint('❌ Error: $e');
    _isLoading = false;
    notifyListeners();
    return false;
  }
}

Future<bool> validarToken(String token) async {
  try {
    if (!_resetTokens.containsKey(token)) {
      _error = 'Token inválido o expirado';
```

```
      notifyListeners();
      return false;
    }

    final tokenData = _resetTokens[token]!;
    final expiracion = DateTime.parse(tokenData['expira']);
    final ahora = DateTime.now();

    if (ahora.isAfter(expiracion)) {
      _error = 'El link de recuperación ha expirado (24 horas)';
      _resetTokens.remove(token);
      notifyListeners();
      return false;
    }

    _error = null;
    notifyListeners();
    return true;
  } catch (e) {
    _error = 'Error al validar token: $e';
    notifyListeners();
    return false;
  }
}

Future<bool> resetearContrasena({
  required String token,
  required String nuevaContrasena,
  required String confirmarContrasena,
}) async {
  _isLoading = true;
  _error = null;
  notifyListeners();

  try {
    final tokenValido = await validarToken(token);
    if (!tokenValido) {
      _isLoading = false;
      notifyListeners();
      return false;
    }

    if (nuevaContrasena.isEmpty) {
      _error = 'La contraseña no puede estar vacía';
      _isLoading = false;
      notifyListeners();
      return false;
    }

    if (nuevaContrasena.length < 6) {
      _error = 'La contraseña debe tener al menos 6 caracteres';
      _isLoading = false;
      notifyListeners();
      return false;
```

```
    }

    if (nuevaContrasena != confirmarContrasena) {
      _error = 'Las contraseñas no coinciden';
      _isLoading = false;
      notifyListeners();
      return false;
    }

    final tokenData = _resetTokens[token]!;
    final email = tokenData['email'];

    final usuario = await _usuarioDao.obtenerPorEmail(email);
    if (usuario == null) {
      _error = 'Usuario no encontrado';
      _isLoading = false;
      notifyListeners();
      return false;
    }

    final contraseniaHash = _hashPassword(nuevaContrasena);

    final usuarioActualizado = Usuario(
      id: usuario.id,
      nombre: usuario.nombre,
      email: usuario.email,
      password: contraseniaHash,
      rol: usuario.rol,
      activo: usuario.activo,
      telefono: usuario.telefono,
      direccion: usuario.direccion,
    );
    await _usuarioDao.actualizar(usuarioActualizado);

    debugPrint('✅ Contraseña actualizada para: $email');

    _resetTokens.remove(token);

    _isLoading = false;
    notifyListeners();
    return true;
  } catch (e) {
    _error = 'Error al resetear contraseña: $e';
    debugPrint('❌ Error: $e');
    _isLoading = false;
    notifyListeners();
    return false;
  }
}

String _hashPassword(String password) {
  return sha256.convert(utf8.encode(password)).toString();
}
```

```
  Future<void> limpiarTokensExpirados() async {
    final ahora = DateTime.now();
    final tokensExpirados = <String>[];

    _resetTokens.forEach((token, data) {
      final expiracion = DateTime.parse(data['expira']);
      if (ahora.isAfter(expiracion)) {
        tokensExpirados.add(token);
      }
    });

    for (var token in tokensExpirados) {
      _resetTokens.remove(token);
    }

    if (tokensExpirados.isNotEmpty) {
      debugPrint('🧹 Tokens expirados limpiados: ${tokensExpirados.length}');
    }
  }

  String? obtenerEmailDelToken(String token) {
    return _resetTokens[token]?['email'];
  }

  Map<String, dynamic>? obtenerDatosToken(String token) {
    return _resetTokens[token];
  }
}
```

7.7 Permission Service

```
class PermissionService {
  // Roles del sistema
  static const String ADMIN = 'admin';
  static const String EMPLEADO = 'empleado';
  static const String Cliente = 'cliente';
}
  // Verificar si el usuario puede gest
```

7.8 Report Service

```
import '../data/dao/venta_dao.dart';
import '../data/dao/producto_dao.dart';
import '../data/dao/pedido_dao.dart';

class ReportService {
  final VentaDao _ventaDao = VentaDao();
  final ProductoDao _productoDao = ProductoDao();
  final PedidoDao _pedidoDao = PedidoDao();

  // Reporte de ventas por período
  Future<ReporteVentas> reporteVentasPeriodo(
    DateTime inicio,
    DateTime fin,
```

```
) async {
  try {
    final ventas = await _ventaDao.obtenerPorFecha(inicio, fin);
    final total = await _ventaDao.obtenerTotalPorPeriodo(inicio, fin);

    final ventasCompletadas = ventas
        .where((v) => v.estado == 'completada')
        .length;

    final ventasCanceladas = ventas
        .where((v) => v.estado == 'cancelada')
        .length;

    final promedioVenta = ventasCompletadas > 0
        ? total / ventasCompletadas
        : 0.0;

    return ReporteVentas(
      totalVentas: total,
      cantidadVentas: ventas.length,
      ventasCompletadas: ventasCompletadas,
      ventasCanceladas: ventasCanceladas,
      promedioVenta: promedioVenta,
      ventas: ventas,
    );
  } catch (e) {
    throw Exception('Error al generar reporte de ventas: $e');
  }
}

// Reporte de inventario
Future<ReporteInventario> reporteInventario() async {
  try {
    final productos = await _productoDao.obtenerActivos();
    final productosBajoStock = await _productoDao.obtenerBajoStock();

    double valorInventario = 0;
    int totalProductos = productos.length;

    for (var producto in productos) {
      valorInventario += producto.stock * producto.precioCompra;
    }

    return ReporteInventario(
      totalProductos: totalProductos,
      productosBajoStock: productosBajoStock.length,
      valorInventario: valorInventario,
      productos: productos,
      productosCriticos: productosBajoStock,
    );
  } catch (e) {
    throw Exception('Error al generar reporte de inventario: $e');
  }
}
```

```
  // Reporte de pedidos
  Future<ReportePedidos> reportePedidos() async {
    try {
      final pedidos = await _pedidoDao.obtenerTodos();

      final pendientes = pedidos
          .where((p) => p.estado == 'pendiente')
          .length;

      final procesando = pedidos
          .where((p) => p.estado == 'procesando')
          .length;

      final enviados = pedidos
          .where((p) => p.estado == 'enviado')
          .length;

      final entregados = pedidos
          .where((p) => p.estado == 'entregado')
          .length;

      return ReportePedidos(
        totalPedidos: pedidos.length,
        pendientes: pendientes,
        procesando: procesando,
        enviados: enviados,
        entregados: entregados,
        pedidos: pedidos,
      );
    } catch (e) {
      throw Exception('Error al generar reporte de pedidos: $e');
    }
  }

  // Resumen del dashboard
  Future<ResumenDashboard> resumenDashboard() async {
    try {
      final hoy = DateTime.now();
      final inicioMes = DateTime(hoy.year, hoy.month, 1);

      final ventasMes = await _ventaDao.obtenerTotalPorPeriodo(inicioMes,
hoy);
      final productos = await _productoDao.obtenerActivos();
      final productosBajoStock = await _productoDao.obtenerBajoStock();
      final pedidosPendientes = await
_pedidoDao.obtenerPorEstado('pendiente');

      return ResumenDashboard(
        ventasMes: ventasMes,
        totalProductos: productos.length,
        productosBajoStock: productosBajoStock.length,
        pedidosPendientes: pedidosPendientes.length,
      );
```

```
    } catch (e) {
      throw Exception('Error al generar resumen: $e');
    }
  }
}

// Clases para resultados de reportes
class ReporteVentas {
  final double totalVentas;
  final int cantidadVentas;
  final int ventasCompletadas;
  final int ventasCanceladas;
  final double promedioVenta;
  final List ventas;

  ReporteVentas({
    required this.totalVentas,
    required this.cantidadVentas,
    required this.ventasCompletadas,
    required this.ventasCanceladas,
    required this.promedioVenta,
    required this.ventas,
  });
}

class  ReporteInventario  {
  final int totalProductos;
  final int productosBajoStock;
  final double valorInventario;
  final List productos;
  final List productosCriticos;

  ReporteInventario({
    required this.totalProductos,
    required this.productosBajoStock,
    required this.valorInventario,
    required this.productos,
    required this.productosCriticos,
  });
}

class ReportePedidos {
  final int totalPedidos;
  final int pendientes;
  final int procesando;
  final int enviados;
  final int entregados;
  final List pedidos;

  ReportePedidos({
    required this.totalPedidos,
    required this.pendientes,
    required this.procesando,
    required this.enviados,
```

```
    required this.entregados,
    required this.pedidos,
  });
}

class ResumenDashboard {
  final double ventasMes;
  final int totalProductos;
  final int productosBajoStock;
  final int pedidosPendientes;

  ResumenDashboard({
    required this.ventasMes,
    required this.totalProductos,
    required this.productosBajoStock,
    required this.pedidosPendientes,
  });
}
```

7.9 Session Time Out Service

```
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import 'dart:async';
import '../services/auth_service.dart';

/// Servicio para manejar el timeout automático de sesión
/// Cierra la sesión automáticamente después de un período de inactividad
class SessionTimeoutService {
  static const int _TIMEOUT_DURATION = 5 * 60 * 1000; // 5 minutos en
milisegundos

  Timer? _timer;
  BuildContext? _context;

  /// Inicializa el servicio con el contexto necesario
  void initialize(BuildContext context) {
    _context = context;
    _resetTimer();
    _setupUserActivityListener();
  }

  /// Reinicia el temporizador de timeout
  void _resetTimer() {
    _timer?.cancel();
    _timer = Timer(const Duration(milliseconds: _TIMEOUT_DURATION),
_handleTimeout);
  }

  /// Configura listeners para detectar actividad del usuario
  void _setupUserActivityListener() {
    if (_context == null) return;

    // Escuchar gestos del usuario para reiniciar el temporizador
```

```
    final binding = WidgetsBinding.instance;
    binding.addObserver(_UserActivityObserver(_resetTimer));
  }

  /// Maneja el timeout de sesión
  void _handleTimeout() {
    if (_context?.mounted == true) {
      final authService = _context!.read<AuthService>();
      authService.cerrarSesion();

      // Mostrar mensaje de timeout
      if (_context!.mounted) {
        ScaffoldMessenger.of(_context!).showSnackBar(
          const SnackBar(
            content: Text(
              'Sesión cerrada por inactividad (5 minutos)',
              style: TextStyle(color: Colors.white),
            ),
            backgroundColor: Color(0xFFE74C3C),
            behavior: SnackBarBehavior.floating,
            duration: Duration(seconds: 3),
            shape: RoundedRectangleBorder(
              borderRadius: BorderRadius.all(Radius.circular(12)),
            ),
          ),
        );

        // Redirigir al login
        Navigator.pushNamedAndRemoveUntil(
          _context!,
          '/login',
          (route) => false,
        );
      }
    }
  }

  /// Cancela el temporizador (usar al cerrar sesión manualmente)
  void cancel() {
    _timer?.cancel();
    _timer = null;
    _context = null;
  }
}

/// Observer para detectar actividad del usuario
class _UserActivityObserver extends WidgetsBindingObserver {
  final VoidCallback onUserActivity;

  _UserActivityObserver(this.onUserActivity);

  @override
  void didChangeAppLifecycleState(AppLifecycleState state) {
    super.didChangeAppLifecycleState(state);
```

```
    if (state == AppLifecycleState.resumed) {
      onUserActivity();
    }
  }

  @override
  void didHaveMemoryPressure() {
    super.didHaveMemoryPressure();
    onUserActivity();
  }
}
```

7.10 **Supabase Storage Service**

```
// lib/services/supabase_storage_service.dart

import 'dart:io';
import 'package:supabase_flutter/supabase_flutter.dart';
import '../config/supabase_config.dart';

class SupabaseStorageService {
  static final _storage = SupabaseConfig.productImages;

  /// Sube una imagen al bucket de Supabase y retorna la URL pública
  ///
  /// [file] - Archivo de imagen a subir
  /// [nombreProducto] - Nombre del producto (se usa para el nombre del
archivo)
  ///
  /// Retorna la URL pública de la imagen subida
  static Future<String> subirImagen(File file, String nombreProducto) async {
    try {
      // Generar nombre único para el archivo
      final timestamp = DateTime.now().millisecondsSinceEpoch;
      final extension = file.path.split('.').last;
      final fileName = '${nombreProducto.replaceAll(' ',
'_')}_$timestamp.$extension';

      // Subir archivo a Supabase Storage
      await _storage.upload(
        fileName,
        file,
        fileOptions: const FileOptions(
          cacheControl: '3600',
          upsert: false,
        ),
      );

      // Obtener URL pública
      final publicUrl = _storage.getPublicUrl(fileName);

      print('✅ Imagen subida exitosamente: $fileName');
      return publicUrl;
```

```
  } catch (e) {
    print('❌ Error al subir imagen: $e');
    rethrow;
  }
}

/// Elimina una imagen del bucket usando su URL
///
/// [imageUrl] - URL pública de la imagen a eliminar
static Future<void> eliminarImagen(String imageUrl) async {
  try {
    // Extraer el nombre del archivo de la URL
    final fileName = _extractFileNameFromUrl(imageUrl);

    if (fileName != null) {
      await _storage.remove([fileName]);
      print('✅ Imagen eliminada exitosamente: $fileName');
    }
  } catch (e) {
    print('❌ Error al eliminar imagen: $e');
    // No lanzar error para no bloquear operaciones
  }
}

/// Actualiza una imagen: elimina la anterior y sube la nueva
///
/// [file] - Nuevo archivo de imagen
/// [nombreProducto] - Nombre del producto
/// [oldImageUrl] - URL de la imagen anterior (opcional)
///
/// Retorna la URL pública de la nueva imagen
static Future<String> actualizarImagen(
  File file,
  String nombreProducto, {
  String? oldImageUrl,
}) async {
  try {
    // Eliminar imagen anterior si existe
    if (oldImageUrl != null && oldImageUrl.isNotEmpty) {
      await eliminarImagen(oldImageUrl);
    }

    // Subir nueva imagen
    return await subirImagen(file, nombreProducto);

  } catch (e) {
    print('❌ Error al actualizar imagen: $e');
    rethrow;
  }
}

/// Lista todas las imágenes en el bucket
static Future<List<FileObject>> listarImagenes() async {
  try {
```

```
      final files = await _storage.list();
      return files;
    } catch (e) {
      print('❌ Error al listar imágenes: $e');
      return [];
    }
  }

  /// Limpia imágenes huérfanas (que no están asociadas a ningún producto)
  ///
  /// [urlsActivas] - Lista de URLs que están siendo usadas por productos
  static Future<void> limpiarImagenesHuerfanas(List<String> urlsActivas)
async {
    try {
      // Obtener todas las imágenes en el bucket
      final todasLasImagenes = await listarImagenes();

      // Extraer nombres de archivos de las URLs activas
      final nombresActivos = urlsActivas
          .map((url) => _extractFileNameFromUrl(url))
          .where((name) => name != null)
          .cast<String>()
          .toSet();

      // Identificar imágenes huérfanas
      final imagenesHuerfanas = todasLasImagenes
          .where((file) => !nombresActivos.contains(file.name))
          .map((file) => file.name)
          .toList();

      // Eliminar imágenes huérfanas
      if (imagenesHuerfanas.isNotEmpty) {
        await _storage.remove(imagenesHuerfanas);
        print('✅ ${imagenesHuerfanas.length} imágenes huérfanas
eliminadas');
      } else {
        print('✅ No hay imágenes huérfanas');
      }

    } catch (e) {
      print('❌ Error al limpiar imágenes huérfanas: $e');
    }
  }

  /// Extrae el nombre del archivo de una URL pública de Supabase
  static String? _extractFileNameFromUrl(String url) {
    try {
      // URL format:
https://xxx.supabase.co/storage/v1/object/public/PRODUCT-IMAGES/filename.jpg
      final uri = Uri.parse(url);
      final segments = uri.pathSegments;

      // El último segmento es el nombre del archivo
      if (segments.isNotEmpty) {
```

```
        return segments.last;
      }
      return null;
    } catch (e) {
      print('❌ Error al extraer nombre de archivo de URL: $e');
      return null;
    }
  }

  /// Verifica si una URL es de Supabase Storage
  static bool esUrlSupabase(String url) {
    return url.contains('supabase.co/storage');
  }
}
```

8 IMPLEMENTACIÓN MVVM

8.1 ProductoViewModel Completo

8.1.1 Gestiona la parte de productos (CRUD, imágenes, stock).

```
// lib/viewmodels/producto_viewmodel.dart

import 'dart:io';
import 'package:flutter/material.dart';
import
'../data/dao/producto_dao_supabase.dart';
import '../data/models/producto.dart';
import '../services/supabase_storage_service.dart';

class ProductoViewModel extends ChangeNotifier {
  final ProductoDaoSupabase _dao = ProductoDaoSupabase();

  List<Producto> _productos = [];
  List<Producto> get productos =>
  _productos;

  bool _cargando = false;
  bool get cargando => _cargando;

  String? _error;
  String? get error => _error;

  /// Cargar todos los productos activos
```

```
  Future<void> cargarProductos() async {
    _cargando = true;
    _error = null;
    notifyListeners();

    try {
      _productos = await _dao.obtenerActivos();
      print('✅ ${_productos.length} productos cargados');
    } catch (e) {
      _error = 'Error al cargar productos: ${e.toString()}';
      print('❌ $_error');
    } finally {
      _cargando = false;
      notifyListeners();
    }
  }

  /// Cargar productos por categoría
  Future<void> cargarPorCategoria(String categoria) async {
    _cargando = true;
    _error = null;
    notifyListeners();

    try {
      _productos = await
      _dao.obtenerPorCategoria(categoria); print('✅
      ${_productos.length} productos de categoría
"$categoria"');
    } catch (e) {
      _error = 'Error al cargar productos: ${e.toString()}';
      print('❌ $_error');
    } finally {
      _cargando = false;
      notifyListeners();
    }
  }

  /// Cargar productos con bajo stock
  Future<void> cargarBajoStock() async
  {
    _cargando = true;
    _error = null;
    notifyListeners();

    try {
      _productos = await _dao.obtenerConStockBajo(); print('⚠
      ${_productos.length} productos con stock bajo');
```

```
    } catch (e) {
      _error = 'Error al cargar productos: ${e.toString()}';
      print('❌ $_error');
    } finally {
      _cargando = false;
      notifyListeners();
    }
  }

  /// Buscar productos por término
  Future<void> buscar(String termino) async
  {
    if (termino.trim().isEmpty) {
      await cargarProductos();
      return;
    }

    _cargando = true;
    _error = null;
    notifyListeners();

    try {
      _productos = await _dao.buscarPorNombre(termino.trim());
      print('✅ ${_productos.length} productos encontrados para
"$termino"');
    } catch (e) {
      _error = 'Error al buscar productos: ${e.toString()}';
      print('❌ $_error');
    } finally {
      _cargando = false;
      notifyListeners();
    }
  }

  /// Crear producto CON imagen en Supabase Storage
  Future<bool> crearProducto(Producto producto, {File? imagenFile}) async
{
    try {
      print('[v0] === crearProducto ===');
      print('[v0] Código:
      ${producto.codigo}'); print('[v0]
      Nombre: ${producto.nombre}');
      print('[v0] imagenFile recibido: ${imagenFile !=
      null}'); if (imagenFile != null) {
        print('[v0] Path imagen: ${imagenFile.path}');
        print('[v0] Imagen existe: ${await imagenFile.exists()}');
      }
```

```
// Verificar código duplicado
final codigoExiste = await
_dao.existeCodigo(producto.codigo); if (codigoExiste) {
  _error = 'El código ya existe';
  notifyListeners();
  return false;
}

// Subir imagen a Supabase Storage si existe
String? imagenUrl;
if (imagenFile != null && await imagenFile.exists()) {
  print('[v0] 📤 Subiendo imagen a Supabase
  Storage...'); try {
    imagenUrl = await SupabaseStorageService.subirImagen(
      imagenFile,
      producto.codigo,
    );
    print('[v0] ✅ Imagen subida exitosamente: $imagenUrl');
  } catch (e) {
    print('[v0] ❌ Error al subir imagen: $e');
    // Continuar sin imagen en caso de error
  }
} else {
  print('[v0] No hay imagen para subir');
}

// Crear producto con URL de
imagen final nuevoProducto =
Producto(
  codigo: producto.codigo,
  nombre: producto.nombre,
  descripcion: producto.descripcion,
  categoria: producto.categoria,
  precioCompra: producto.precioCompra,
  precioVenta: producto.precioVenta,
  stock: producto.stock,
  stockMinimo: producto.stockMinimo,
  proveedor: producto.proveedor,
  imagenUrl: imagenUrl,
  activo: true,
);

final id = await _dao.insertar(nuevoProducto);

if (id != null) {
```

```
        await cargarProductos();
        print('[v0] ✅ Producto creado con ID:
        $id'); return true;
      } else {
        // Si falla, eliminar imagen subida
        if (imagenUrl != null) {
          await SupabaseStorageService.eliminarImagen(imagenUrl);
        }
        _error = 'Error al crear producto';
        notifyListeners();
        return false;
      }
    } catch (e) {
      _error = 'Error al crear producto: ${e.toString()}';
      print('[v0] ❌ $_error');
      notifyListeners();
      return false;
    }
  }

  /// Actualizar producto CON manejo de imagen
  Future<bool> actualizarProducto(Producto producto, {File?
nuevaImagen}) async {
    try {
      print('[v0] === actualizarProducto
      ==='); print('[v0] ID: ${producto.id}');
      print('[v0] Código:
      ${producto.codigo}');
      print('[v0] nuevaImagen recibida: ${nuevaImagen !=
      null}'); if (nuevaImagen != null) {
        print('[v0] Path imagen: ${nuevaImagen.path}');
        print('[v0] Imagen existe: ${await nuevaImagen.exists()}');
      }
      print('[v0] imagenUrl actual: ${producto.imagenUrl}');

      if (producto.id == null) {
        _error = 'Producto sin ID';
        notifyListeners();
        return false;
      }

      String? imagenUrl = producto.imagenUrl;

      // Si hay nueva imagen, actualizar en Storage
      if (nuevaImagen != null && await nuevaImagen.exists()) {
        print('[v0] 📤 Actualizando imagen en Supabase Storage...');
```

```
      try {
        imagenUrl = await SupabaseStorageService.actualizarImagen(
          nuevaImagen,
          producto.codigo,
          oldImageUrl: producto.imagenUrl,
        );
        print('[v0] ✅ Imagen actualizada: $imagenUrl');
      } catch (e) {
        print('[v0] ❌ Error al actualizar imagen: $e');
        // Mantener la imagen anterior en caso de error
      }
    }

    // Actualizar producto
    final productoActualizado = Producto(
      id: producto.id,
      codigo: producto.codigo,
      nombre: producto.nombre,
      descripcion: producto.descripcion,
      categoria: producto.categoria,
      precioCompra: producto.precioCompra,
      precioVenta: producto.precioVenta,
      stock: producto.stock,
      stockMinimo: producto.stockMinimo,
      proveedor: producto.proveedor,
      imagenUrl: imagenUrl,
      activo: producto.activo,
    );

    final exitoso = await _dao.actualizar(productoActualizado);

    if (exitoso) {
      await cargarProductos();
      print('[v0] ✅ Producto actualizado: ${producto.nombre}');
      return true;
    } else {
      _error = 'Error al actualizar producto';
      notifyListeners();
      return false;
    }
  } catch (e) {
    _error = 'Error al actualizar producto: ${e.toString()}';
    print('[v0] ❌ $_error');
    notifyListeners();
    return false;
```

```
    }
  }

  /// Eliminar producto Y su imagen de Storage
  Future<bool> eliminarProducto(int id) async {
    try {
      // Obtener producto para eliminar su imagen
      final producto = await _dao.obtenerPorId(id);

      if (producto != null) {
        // Eliminar imagen si existe
        if (producto.imagenUrl != null && producto.imagenUrl!.isNotEmpty)
{
          print('🗑️ Eliminando imagen de Storage...');
          await
SupabaseStorageService.eliminarImagen(producto.imagenUrl!);
        }

        // Eliminar producto (soft delete)
        final exitoso = await _dao.eliminar(id);

        if (exitoso) {
          await cargarProductos();
          print('✅ Producto
          eliminado'); return true;
        }
      }

      _error = 'Error al eliminar producto';
      notifyListeners();
      return false;
    } catch (e) {
      _error = 'Error al eliminar producto:
      ${e.toString()}'; print('❌ $_error');
      notifyListeners();
      return false;
    }
  }

  /// Eliminar solo la imagen de un producto
  Future<bool> eliminarImagenProducto(int id) async
  {
    try {
      final producto = await
      _dao.obtenerPorId(id); if (producto ==
      null) {
        _error = 'Producto no encontrado';
```

```
      notifyListeners();
      return false;
    }

    // Eliminar imagen de Storage
    if (producto.imagenUrl != null && producto.imagenUrl!.isNotEmpty) {
      print('🗑️ Eliminando imagen...');
      await SupabaseStorageService.eliminarImagen(producto.imagenUrl!);
    }

    // Actualizar producto sin imagen
    final productoSinImagen =
    Producto(
      id: producto.id,
      codigo:
      producto.codigo,
      nombre:
      producto.nombre,
      descripcion: producto.descripcion,
      categoria: producto.categoria,
      precioCompra: producto.precioCompra,
      precioVenta: producto.precioVenta,
      stock: producto.stock,
      stockMinimo: producto.stockMinimo,
      proveedor: producto.proveedor,
      imagenUrl: null, // Eliminar URL
      activo: producto.activo,
    );

    final exitoso = await _dao.actualizar(productoSinImagen);

    if (exitoso) {
      await cargarProductos();
      print('✅ Imagen eliminada del producto');
      return true;
    }

    _error = 'Error al eliminar imagen';
    notifyListeners();
    return false;
  } catch (e) {
    _error = 'Error al eliminar imagen: ${e.toString()}';
    print('❌ $_error');
    notifyListeners();
    return false;
  }
}
```

```
/// Actualizar stock de un producto
Future<bool> actualizarStock(int id, int nuevoStock) async
  { try {
    final exitoso = await _dao.actualizarStock(id, nuevoStock);

    if (exitoso) {
      await cargarProductos();
      print('✅ Stock actualizado');
      return true;
    }

    _error = 'Error al actualizar stock';
    notifyListeners();
    return false;
  } catch (e) {
    _error = 'Error al actualizar stock: ${e.toString()}';
    print('❌ $_error');
    notifyListeners();
    return false;
  }
}

/// Limpiar imágenes huérfanas en Storage
Future<void> limpiarImagenesHuerfanas() async {
  try {
    print('🧹 Limpiando imágenes huérfanas...');

    // Obtener URLs en uso
    final urlsEnUso = await _dao.obtenerTodasLasImagenesUrl();

    // Limpiar Storage
    await SupabaseStorageService.limpiarImagenesHuerfanas(urlsEnUso);

    print('✅ Limpieza completada');
  } catch (e) {
    print('❌ Error al limpiar imágenes huérfanas: $e');
  }
}

/// Obtener categorías disponibles
Future<List<String>> obtenerCategorias() async {
  try {
    return await _dao.obtenerCategorias();
  } catch (e) {
    print('❌ Error al obtener categorías: $e');
```

8.2 PedidoViewModel Completo

8.2.1 Gestión de pedidos para: crear, actualizar, cancelar, filtrar por estado.

```
// lib/viewmodels/pedido_viewmodel.dart

import 'package:flutter/material.dart';
import '../data/dao/pedido_dao_supabase.dart';
import '../data/models/pedido.dart';
import '../data/models/detalle_pedido.dart';
import '../data/models/usuario.dart';
import '../data/models/pedido_estado.dart';

class PedidoViewModel extends ChangeNotifier {
  final PedidoDaoSupabase _dao = PedidoDaoSupabase();

  // ================================
  //   USUARIO ACTUAL
  // ================================
  Usuario? _usuarioActual;
  Usuario? get usuarioActual => _usuarioActual;

  void setUsuarioActual(Usuario usuario) {
    _usuarioActual = usuario;
    notifyListeners();
  }

  // ================================
  //   DATOS
  // ================================
```

```
List<Pedido> _pedidos = [];
List<Pedido> get pedidos => _pedidos;

Map<String, dynamic> _estadisticas = {};
Map<String, dynamic> get estadisticas => _estadisticas;

bool _cargando = false;
bool get cargando => _cargando;

String? _error;
String? get error => _error;

PedidoEstado? _filtroEstado;
PedidoEstado? get filtroEstado => _filtroEstado;

// ================================
//   MÉTODOS
// ================================

/// Cargar todos los pedidos
Future<void> cargarPedidos() async {
  _cargando = true;
  _error = null;
  notifyListeners();

  try {
    _pedidos = await _dao.obtenerTodos();
    _estadisticas = await _dao.obtenerEstadisticas();
    print('✅ ${_pedidos.length} pedidos cargados');
```

```
    } catch (e) {
      _error = 'Error al cargar pedidos: ${e.toString()}';
      print('❌ $_error');
    } finally {
      _cargando = false;
      notifyListeners();
    }
  }

  /// Cargar pedidos por estado
  Future<void> cargarPorEstado(PedidoEstado estado) async {
    _cargando = true;
    _error = null;
    _filtroEstado = estado;
    notifyListeners();

    try {
      _pedidos = await _dao.obtenerPorEstado(estado);
      print('✅ ${_pedidos.length} pedidos con estado
"${estado.name}"');
    } catch (e) {
      _error = 'Error al cargar pedidos: ${e.toString()}';
      print('❌ $_error');
    } finally {
      _cargando = false;
      notifyListeners();
    }
  }
```

```
/// Cargar pedidos pendientes
Future<void> cargarPendientes() async {
  await cargarPorEstado(PedidoEstado.pendientePago);
}

/// Cargar pedidos en proceso
Future<void> cargarEnProceso() async {
  await cargarPorEstado(PedidoEstado.procesando);
}

/// Cargar pedidos entregados
Future<void> cargarEntregados() async {
  await cargarPorEstado(PedidoEstado.entregado);
}

/// Cargar pedidos cancelados
Future<void> cargarCancelados() async {
  await cargarPorEstado(PedidoEstado.cancelado);
}

/// Cargar pedidos por cliente
Future<void> cargarPorCliente(int clienteId) async {
  _cargando = true;
  _error = null;
  notifyListeners();

  try {
    _pedidos = await _dao.obtenerPorCliente(clienteId);
    print('✅ ${_pedidos.length} pedidos del cliente
```

```
$clienteId');
    } catch (e) {
      _error = 'Error al cargar pedidos: ${e.toString()}';
      print('❌ $_error');
    } finally {
      _cargando = false;
      notifyListeners();
    }
  }

  /// Cargar pedidos por vendedor
  Future<void> cargarPorVendedor(int vendedorId) async {
    _cargando = true;
    _error = null;
    notifyListeners();

    try {
      _pedidos = await _dao.obtenerPorVendedor(vendedorId);
      print('✅ ${_pedidos.length} pedidos del vendedor
$vendedorId');
    } catch (e) {
      _error = 'Error al cargar pedidos: ${e.toString()}';
      print('❌ $_error');
    } finally {
      _cargando = false;
      notifyListeners();
    }
  }
```

```
  /// Cargar pedidos por rango de fechas
  Future<void> cargarPorFechas(DateTime inicio, DateTime fin)
async {
    _cargando = true;
    _error = null;
    notifyListeners();

    try {
      _pedidos = await _dao.obtenerPorFechas(inicio, fin);
      print('✅ ${_pedidos.length} pedidos en el rango de
fechas');
    } catch (e) {
      _error = 'Error al cargar pedidos: ${e.toString()}';
      print('❌ $_error');
    } finally {
      _cargando = false;
      notifyListeners();
    }
  }

  /// Crear pedido con detalles
  Future<int?> crearPedido(Pedido pedido, List<DetallePedido>
detalles) async {
    try {
      print('=== PedidoViewModel.crearPedido ===');
      print('Cliente ID: ${pedido.clienteId}');
      print('Vendedor ID: ${pedido.vendedorId}');
      print('Total: \$${pedido.total}');
      print('Estado: ${pedido.estado}');
      print('Cantidad detalles: ${detalles.length}');
```

```
    // Asignar detalles al pedido
    pedido.detalles = detalles;

    // Insertar pedido
    final pedidoId = await _dao.insertar(pedido);

    if (pedidoId != null) {
      print('✅ Pedido creado con ID: $pedidoId');
      await cargarPedidos();
      return pedidoId;
    } else {
      _error = 'Error al crear pedido';
      print('❌ $_error');
      notifyListeners();
      return null;
    }
  } catch (e) {
    _error = 'Error al crear pedido: ${e.toString()}';
    print('❌ $_error');
    notifyListeners();
    return null;
  }
}

/// Actualizar pedido (solo datos principales, no detalles)
Future<bool> actualizarPedido(Pedido pedido) async {
  try {
    if (pedido.id == null) {
```

```
        _error = 'Pedido sin ID';
        notifyListeners();
        return false;
      }

      final exitoso = await _dao.actualizar(pedido);

      if (exitoso) {
        await cargarPedidos();
        print('✅ Pedido actualizado');
        return true;
      } else {
        _error = 'Error al actualizar pedido';
        notifyListeners();
        return false;
      }
    } catch (e) {
      _error = 'Error al actualizar pedido: ${e.toString()}';
      print('❌ $_error');
      notifyListeners();
      return false;
    }
  }

  /// Actualizar estado del pedido
  Future<bool> actualizarEstado(int id, PedidoEstado nuevoEstado,
{int? vendedorId}) async {
    try {
      final exitoso = await _dao.actualizarEstado(id,
```

```
nuevoEstado, vendedorId: vendedorId);

      if (exitoso) {
        await cargarPedidos();
        print('✅ Estado actualizado a "${nuevoEstado.name}"');
        return true;
      } else {
        _error = 'Error al actualizar estado';
        notifyListeners();
        return false;
      }
    } catch (e) {
      _error = 'Error al actualizar estado: ${e.toString()}';
      print('❌ $_error');
      notifyListeners();
      return false;
    }
  }

  /// Marcar pedido como en proceso
  Future<bool> marcarEnProceso(int id) async {
    return await actualizarEstado(id, PedidoEstado.procesando);
  }

  /// Marcar pedido como entregado
  Future<bool> marcarEntregado(int id) async {
    return await actualizarEstado(id, PedidoEstado.entregado);
  }
```

```
  /// Cancelar pedido
  Future<bool> cancelarPedido(int id) async {
    try {
      final exitoso = await _dao.actualizarEstado(id,
PedidoEstado.cancelado);

      if (exitoso) {
        await cargarPedidos();
        print('✅ Pedido cancelado');
        return true;
      } else {
        _error = 'Error al cancelar pedido';
        notifyListeners();
        return false;
      }
    } catch (e) {
      _error = 'Error al cancelar pedido: ${e.toString()}';
      print('❌ $_error');
      notifyListeners();
      return false;
    }
  }

  /// Eliminar pedido permanentemente
  Future<bool> eliminarPedido(int id) async {
    try {
      final exitoso = await _dao.eliminar(id);

      if (exitoso) {
```

```
      await cargarPedidos();
      print('✅ Pedido eliminado');
      return true;
    } else {
      _error = 'Error al eliminar pedido';
      notifyListeners();
      return false;
    }
  } catch (e) {
    _error = 'Error al eliminar pedido: ${e.toString()}';
    print('❌ $_error');
    notifyListeners();
    return false;
  }
}

/// Obtener pedido completo con detalles
Future<Pedido?> obtenerPedidoDetalle(int id) async {
  try {
    return await _dao.obtenerPorId(id);
  } catch (e) {
    _error = 'Error al obtener pedido: ${e.toString()}';
    print('❌ $_error');
    notifyListeners();
    return null;
  }
}

/// Generar número de pedido único (formato: PED-YYYYMMDD-0001)
```

```
  Future<String> generarNumeroPedido() async {
    try {
      final ahora = DateTime.now();
      final fecha =
'${ahora.year}${ahora.month.toString().padLeft(2,
'0')}${ahora.day.toString().padLeft(2, '0')}';

      // Obtener pedidos de hoy
      final inicioHoy = DateTime(ahora.year, ahora.month,
ahora.day);
      final finHoy = DateTime(ahora.year, ahora.month, ahora.day,
23, 59, 59);
      final pedidosHoy = await _dao.obtenerPorFechas(inicioHoy,
finHoy);

      // Generar número secuencial
      final numeroSecuencial = (pedidosHoy.length +
1).toString().padLeft(4, '0');

      return 'PED-$fecha-$numeroSecuencial';
    } catch (e) {
      print('❌ Error al generar número de pedido: $e');
      // Fallback en caso de error
      return 'PED-${DateTime.now().millisecondsSinceEpoch}';
    }
  }

  /// Obtener estadísticas del cliente
  Map<String, dynamic> obtenerEstadisticasCliente(int clienteId)
{
    final pedidosCliente = _pedidos
```

```
        .where((pedido) => pedido.clienteId == clienteId)
        .toList();

    final totalGastado = pedidosCliente.fold<double>(
      0.0,
      (sum, pedido) => sum + pedido.total,
    );

    final totalPedidos = pedidosCliente.length;
    final pendientes = pedidosCliente.where((p) => p.estado ==
PedidoEstado.pendientePago).length;
    final entregados = pedidosCliente.where((p) => p.estado ==
PedidoEstado.entregado).length;

    return {
      'total_gastado': totalGastado,
      'total_pedidos': totalPedidos,
      'pedidos_pendientes': pendientes,
      'pedidos_entregados': entregados,
    };
  }

  /// Limpiar filtros
  void limpiarFiltros() {
    _filtroEstado = null;
    notifyListeners();
  }

  /// Limpiar error
  void limpiarError() {
```

```
    _error = null;
    notifyListeners();
  }
}
```

8.3 ClienteViewModel Completo

8.3.1 Gestión de clientes (CRUD, búsqueda).

```
// lib/viewmodels/cliente_viewmodel.dart

import 'package:flutter/material.dart';
import '../data/dao/usuario_dao_supabase.dart';
import '../data/models/usuario.dart';

class ClienteViewModel extends ChangeNotifier {
  final UsuarioDaoSupabase _dao = UsuarioDaoSupabase();

  List<Usuario> _clientes = [];
  List<Usuario> get clientes => _clientes;

  Usuario? _clienteSeleccionado;
  Usuario? get clienteSeleccionado => _clienteSeleccionado;

  bool _cargando = false;
  bool get cargando => _cargando;

  String? _error;
  String? get error => _error;

  /// Cargar todos los clientes (usuarios con rol='cliente')
  Future<void> cargarClientes() async {
    _cargando = true;
    _error = null;
    notifyListeners();

    try {
      _clientes = await _dao.obtenerClientes();
      print('✅ ${_clientes.length} clientes cargados');
    } catch (e) {
      _error = 'Error al cargar clientes: ${e.toString()}';
      print('❌ $_error');
    } finally {
      _cargando = false;
      notifyListeners();
    }
  }

  /// Buscar clientes por nombre o email
  Future<void> buscarCliente(String busqueda) async {
    if (busqueda.isEmpty) {
      await cargarClientes();
```

```
      return;
    }

    _cargando = true;
    _error = null;
    notifyListeners();

    try {
      // Obtener todos los clientes
      final todosClientes = await _dao.obtenerClientes();

      // Filtrar por nombre o email
      final busquedaLower = busqueda.toLowerCase();
      _clientes = todosClientes.where((cliente) {
        final nombreMatch =
cliente.nombre.toLowerCase().contains(busquedaLower);
        final emailMatch =
cliente.email.toLowerCase().contains(busquedaLower);
        return nombreMatch || emailMatch;
      }).toList();

      print('✅ ${_clientes.length} clientes encontrados para "$busqueda"');
    } catch (e) {
      _error = 'Error al buscar cliente: ${e.toString()}';
      print('❌ $_error');
    } finally {
      _cargando = false;
      notifyListeners();
    }
  }

  /// Crear nuevo cliente
  Future<bool> crearCliente(Usuario cliente) async {
    try {
      // Verificar que el rol sea 'cliente'
      if (cliente.rol != 'cliente') {
        _error = 'El usuario debe tener rol de cliente';
        notifyListeners();
        return false;
      }

      // Verificar que el email no exista
      final emailExiste = await _dao.existeEmail(cliente.email);
      if (emailExiste) {
        _error = 'El email ya está registrado';
        notifyListeners();
        return false;
      }

      // Insertar cliente
      final id = await _dao.insertar(cliente);

      if (id != null) {
        _clienteSeleccionado = cliente.copyWith(id: id);
```

```
      await cargarClientes();
      print('✅ Cliente creado: ${cliente.nombre}');
      return true;
    } else {
      _error = 'Error al crear cliente';
      notifyListeners();
      return false;
    }
  } catch (e) {
    _error = 'Error al crear cliente: ${e.toString()}';
    print('❌ $_error');
    notifyListeners();
    return false;
  }
}

/// Actualizar cliente existente
Future<bool> actualizarCliente(Usuario cliente) async {
  try {
    if (cliente.id == null) {
      _error = 'Cliente sin ID';
      notifyListeners();
      return false;
    }

    // Verificar email duplicado (excluyendo el cliente actual)
    final emailExiste = await _dao.existeEmail(
      cliente.email,
      exceptoId: cliente.id,
    );

    if (emailExiste) {
      _error = 'El email ya está en uso';
      notifyListeners();
      return false;
    }

    // Actualizar
    final exitoso = await _dao.actualizar(cliente);

    if (exitoso) {
      await cargarClientes();
      print('✅ Cliente actualizado: ${cliente.nombre}');
      return true;
    } else {
      _error = 'Error al actualizar cliente';
      notifyListeners();
      return false;
    }
  } catch (e) {
    _error = 'Error al actualizar cliente: ${e.toString()}';
    print('❌ $_error');
    notifyListeners();
    return false;
```

```
    }
  }

  /// Eliminar cliente (soft delete)
  Future<bool> eliminarCliente(int id) async {
    try {
      final exitoso = await _dao.eliminar(id);

      if (exitoso) {
        await cargarClientes();
        print('✅ Cliente desactivado');
        return true;
      } else {
        _error = 'Error al eliminar cliente';
        notifyListeners();
        return false;
      }
    } catch (e) {
      _error = 'Error al eliminar cliente: ${e.toString()}';
      print('❌ $_error');
      notifyListeners();
      return false;
    }
  }

  /// Seleccionar cliente
  void seleccionarCliente(Usuario cliente) {
    _clienteSeleccionado = cliente;
    notifyListeners();
  }

  /// Limpiar selección
  void limpiarSeleccion() {
    _clienteSeleccionado = null;
    notifyListeners();
  }

  /// Limpiar error
  void limpiarError() {
    _error = null;
    notifyListeners();
  }

  /// Obtener cliente por ID
  Future<Usuario?> obtenerClientePorId(int id) async {
    try {
      return await _dao.obtenerPorId(id);
    } catch (e) {
      _error = 'Error al obtener cliente: ${e.toString()}';
      notifyListeners();
      return null;
    }
  }
}
```

8.4 UsuarioViewModel Completo

8.4.1 Gestión de usuarios (login, roles, estadísticas).

```
// lib/viewmodels/usuario_viewmodel.dart

import 'package:flutter/material.dart';
import '../data/dao/usuario_dao_supabase.dart';
import '../data/models/usuario.dart';

class UsuarioViewModel extends ChangeNotifier {
  final UsuarioDaoSupabase _dao = UsuarioDaoSupabase();

  List<Usuario> _usuarios = [];
  List<Usuario> get usuarios => _usuarios;

  Usuario? _usuarioActual;
  Usuario? get usuarioActual => _usuarioActual;

  bool _cargando = false;
  bool get cargando => _cargando;

  String? _error;
  String? get error => _error;

  /// Establecer usuario actual (para sesión)
  void setUsuarioActual(Usuario usuario) {
    _usuarioActual = usuario;
    notifyListeners();
  }
```

```
  /// Cerrar sesión
  void cerrarSesion() {
    _usuarioActual = null;
    notifyListeners();
  }

  /// Validar login
  Future<Usuario?> validarLogin(String email, String password)
async {
    _cargando = true;
    _error = null;
    notifyListeners();

    try {
      final usuario = await _dao.validarLogin(email, password);

      if (usuario != null) {
        _usuarioActual = usuario;
        print('✅ Login exitoso: ${usuario.nombre}
(${usuario.rol})');
        return usuario;
      } else {
        _error = 'Credenciales incorrectas';
        print('❌ Login fallido: credenciales incorrectas');
        return null;
      }
    } catch (e) {
      _error = 'Error al validar login: ${e.toString()}';
      print('❌ $_error');
      return null;
```

```
  } finally {
    _cargando = false;
    notifyListeners();
  }
}

/// Cargar todos los usuarios
Future<void> cargarUsuarios() async {
  _cargando = true;
  _error = null;
  notifyListeners();

  try {
    _usuarios = await _dao.obtenerTodos();
    print('✅ ${_usuarios.length} usuarios cargados');
  } catch (e) {
    _error = 'Error al cargar usuarios: ${e.toString()}';
    print('❌ $_error');
  } finally {
    _cargando = false;
    notifyListeners();
  }
}

/// Cargar usuarios activos
Future<void> cargarUsuariosActivos() async {
  _cargando = true;
  _error = null;
  notifyListeners();
```

```
  try {
    _usuarios = await _dao.obtenerActivos();
    print('✅ ${_usuarios.length} usuarios activos');
  } catch (e) {
    _error = 'Error al cargar usuarios: ${e.toString()}';
    print('❌ $_error');
  } finally {
    _cargando = false;
    notifyListeners();
  }
}

/// Cargar usuarios por rol
Future<void> cargarPorRol(String rol) async {
  _cargando = true;
  _error = null;
  notifyListeners();

  try {
    _usuarios = await _dao.obtenerPorRol(rol);
    print('✅ ${_usuarios.length} usuarios con rol "$rol"');
  } catch (e) {
    _error = 'Error al cargar usuarios: ${e.toString()}';
    print('❌ $_error');
  } finally {
    _cargando = false;
    notifyListeners();
  }
}
```

```
/// Cargar clientes
Future<void> cargarClientes() async {
  _cargando = true;
  _error = null;
  notifyListeners();

  try {
    _usuarios = await _dao.obtenerClientes();
    print('✅ ${_usuarios.length} clientes cargados');
  } catch (e) {
    _error = 'Error al cargar clientes: ${e.toString()}';
    print('❌ $_error');
  } finally {
    _cargando = false;
    notifyListeners();
  }
}

/// Cargar vendedores
Future<void> cargarVendedores() async {
  _cargando = true;
  _error = null;
  notifyListeners();

  try {
    _usuarios = await _dao.obtenerVendedores();
    print('✅ ${_usuarios.length} vendedores cargados');
  } catch (e) {
    _error = 'Error al cargar vendedores: ${e.toString()}';
    print('❌ $_error');
```

```
  } finally {
    _cargando = false;
    notifyListeners();
  }
}

/// Crear usuario
Future<bool> crearUsuario(Usuario usuario) async {
  try {
    final emailExiste = await _dao.existeEmail(usuario.email);
    if (emailExiste) {
      _error = 'El email ya está registrado';
      notifyListeners();
      return false;
    }

    final id = await _dao.insertar(usuario);

    if (id != null) {
      await cargarUsuarios();
      print('✅ Usuario creado: ${usuario.nombre}');
      return true;
    } else {
      _error = 'Error al crear usuario';
      notifyListeners();
      return false;
    }
  } catch (e) {
    _error = 'Error al crear usuario: ${e.toString()}';
    print('❌ $_error');
```

```
    notifyListeners();
    return false;
  }
}

/// Actualizar usuario
Future<bool> actualizarUsuario(Usuario usuario) async {
  try {
    if (usuario.id == null) {
      _error = 'Usuario sin ID';
      notifyListeners();
      return false;
    }

    final emailExiste = await _dao.existeEmail(
      usuario.email,
      exceptoId: usuario.id,
    );

    if (emailExiste) {
      _error = 'El email ya está en uso';
      notifyListeners();
      return false;
    }

    final exitoso = await _dao.actualizar(usuario);

    if (exitoso) {
      await cargarUsuarios();
```

```
      // Actualizar usuario actual si es el mismo
      if (_usuarioActual?.id == usuario.id) {
        _usuarioActual = usuario;
      }

      print('✅ Usuario actualizado: ${usuario.nombre}');
      return true;
    } else {
      _error = 'Error al actualizar usuario';
      notifyListeners();
      return false;
    }
  } catch (e) {
    _error = 'Error al actualizar usuario: ${e.toString()}';
    print('❌ $_error');
    notifyListeners();
    return false;
  }
}

/// Desactivar usuario (soft delete)
Future<bool> desactivarUsuario(int id) async {
  try {
    final exitoso = await _dao.eliminar(id);

    if (exitoso) {
      await cargarUsuarios();
      print('✅ Usuario desactivado');
      return true;
    } else {
```

```
      _error = 'Error al desactivar usuario';
      notifyListeners();
      return false;
    }
  } catch (e) {
    _error = 'Error al desactivar usuario: ${e.toString()}';
    print('❌ $_error');
    notifyListeners();
    return false;
  }
}

/// Activar usuario (reactivar después de soft delete)
Future<bool> activarUsuario(int id) async {
  try {
    // Obtener usuario
    final usuario = await _dao.obtenerPorId(id);
    if (usuario == null) {
      _error = 'Usuario no encontrado';
      notifyListeners();
      return false;
    }

    // Activar
    final usuarioActivo = usuario.copyWith(activo: true);
    final exitoso = await _dao.actualizar(usuarioActivo);

    if (exitoso) {
      await cargarUsuarios();
      print('✅ Usuario activado');
```

```
      return true;
    } else {
      _error = 'Error al activar usuario';
      notifyListeners();
      return false;
    }
  } catch (e) {
    _error = 'Error al activar usuario: ${e.toString()}';
    print('❌ $_error');
    notifyListeners();
    return false;
  }
}

/// Eliminar usuario permanentemente
Future<bool> eliminarUsuario(int id) async {
  try {
    final exitoso = await _dao.eliminarPermanente(id);

    if (exitoso) {
      await cargarUsuarios();
      print('✅ Usuario eliminado permanentemente');
      return true;
    } else {
      _error = 'Error al eliminar usuario';
      notifyListeners();
      return false;
    }
  } catch (e) {
    _error = 'Error al eliminar usuario: ${e.toString()}';
```

```
      print('❌ $_error');
      notifyListeners();
      return false;
    }
  }

  /// Buscar usuarios por nombre o email
  Future<void> buscarUsuarios(String termino) async {
    if (termino.trim().isEmpty) {
      await cargarUsuarios();
      return;
    }

    _cargando = true;
    _error = null;
    notifyListeners();

    try {
      // Obtener todos los usuarios
      final todosUsuarios = await _dao.obtenerTodos();

      // Filtrar
      final terminoLower = termino.toLowerCase();
      _usuarios = todosUsuarios.where((usuario) {
        final nombreMatch =
usuario.nombre.toLowerCase().contains(terminoLower);
        final emailMatch =
usuario.email.toLowerCase().contains(terminoLower);
        return nombreMatch || emailMatch;
      }).toList();
```

```
      print('✅ ${_usuarios.length} usuarios encontrados');
    } catch (e) {
      _error = 'Error al buscar usuarios: ${e.toString()}';
      print('❌ $_error');
    } finally {
      _cargando = false;
      notifyListeners();
    }
  }

  /// Cambiar contraseña
  Future<bool> cambiarPassword(int id, String nuevoPassword)
async {
    try {
      final exitoso = await _dao.cambiarPassword(id,
nuevoPassword);

      if (exitoso) {
        print('✅ Contraseña actualizada');
        return true;
      } else {
        _error = 'Error al cambiar contraseña';
        notifyListeners();
        return false;
      }
    } catch (e) {
      _error = 'Error al cambiar contraseña: ${e.toString()}';
      print('❌ $_error');
      notifyListeners();
      return false;
```

```
  }
}

/// Obtener estadísticas de usuarios
Future<Map<String, int>> obtenerEstadisticas() async {
  try {
    final todos = await _dao.obtenerTodos();
    final activos = await _dao.obtenerActivos();
    final admins = await _dao.obtenerPorRol('admin');
    final vendedores = await _dao.obtenerVendedores();
    final clientes = await _dao.obtenerClientes();

    return {
      'total': todos.length,
      'activos': activos.length,
      'inactivos': todos.length - activos.length,
      'admins': admins.length,
      'vendedores': vendedores.length,
      'clientes': clientes.length,
    };
  } catch (e) {
    _error = 'Error al obtener estadísticas: ${e.toString()}';
    print('❌ $_error');
    notifyListeners();
    return {};
  }
}

/// Obtener usuario por ID
Future<Usuario?> obtenerUsuarioPorId(int id) async {
```

```
    try {
      return await _dao.obtenerPorId(id);
    } catch (e) {
      _error = 'Error al obtener usuario: ${e.toString()}';
      notifyListeners();
      return null;
    }
  }

  /// Limpiar error
  void limpiarError() {
    _error = null;
    notifyListeners();
  }
}
```

8.5 VentaViewModel Completo

8.5.1 Gestión de ventas para: crear, reportes, estadísticas.

```
// lib/viewmodels/venta_viewmodel.dart

import 'package:flutter/material.dart';
import '../data/dao/venta_dao_supabase.dart';
import '../data/models/venta.dart';
import '../data/models/detalle_venta.dart';
import '../core/constants.dart';

class VentaViewModel extends ChangeNotifier {
  final VentaDaoSupabase _dao = VentaDaoSupabase();
```

```
  List<Venta> _ventas = [];
  List<Venta> get ventas => _ventas;

  List<Venta> _ventasHoy = [];
  List<Venta> get ventasHoy => _ventasHoy;

  Map<String, dynamic> _estadisticas = {};
  Map<String, dynamic> get estadisticas => _estadisticas;

  bool _cargando = false;
  bool get cargando => _cargando;

  String? _error;
  String? get error => _error;

  /// Getter para total de ventas de hoy
  double get totalVentasHoy {
    return _ventasHoy.fold(0.0, (sum, venta) => sum +
venta.total);
  }

  /// Getter para cantidad de ventas completadas hoy
  int get ventasCompletadasHoy {
```

```
  return _ventasHoy.where((venta) =>
    venta.estado == AppConstants.ventaCompletada
  ).length;
}

/// Cargar todas las ventas
Future<void> cargarVentas() async {
  _cargando = true;
  _error = null;
  notifyListeners();

  try {
    _ventas = await _dao.obtenerTodas();
    _estadisticas = await _dao.obtenerEstadisticas();
    print('✅ ${_ventas.length} ventas cargadas');
  } catch (e) {
    _error = 'Error al cargar ventas: ${e.toString()}';
    print('❌ $_error');
  } finally {
    _cargando = false;
    notifyListeners();
  }
}
```

```
  /// Cargar ventas por cliente
  Future<void> cargarPorCliente(int clienteId) async {
    _cargando = true;
    _error = null;
    notifyListeners();

    try {
      _ventas = await _dao.obtenerPorCliente(clienteId);
      print('✅ ${_ventas.length} ventas del cliente $clienteId');
    } catch (e) {
      _error = 'Error al cargar ventas del cliente: ${e.toString()}';
      print('❌ $_error');
      _ventas = [];
    } finally {
      _cargando = false;
      notifyListeners();
    }
  }

  /// Cargar ventas por vendedor
  Future<void> cargarPorVendedor(int vendedorId) async {
    _cargando = true;
    _error = null;
```

```
    notifyListeners();

    try {
      _ventas = await _dao.obtenerPorVendedor(vendedorId);
      print('✅ ${_ventas.length} ventas del vendedor $vendedorId');
    } catch (e) {
      _error = 'Error al cargar ventas del vendedor: ${e.toString()}';
      print('❌ $_error');
      _ventas = [];
    } finally {
      _cargando = false;
      notifyListeners();
    }
  }

  /// Cargar ventas de hoy
  Future<void> cargarVentasHoy() async {
    _cargando = true;
    _error = null;
    notifyListeners();

    print('=== VentaViewModel.cargarVentasHoy ===');
```

```
    try {
      // Obtener ventas de hoy (últimas 24 horas)
      final ahora = DateTime.now();
      final inicioHoy = DateTime(ahora.year, ahora.month,
ahora.day);
      final finHoy = DateTime(ahora.year, ahora.month,
ahora.day, 23, 59, 59);

      _ventasHoy = await _dao.obtenerPorFechas(inicioHoy,
finHoy);
      _ventas = _ventasHoy; // También actualizar _ventas
para compatibilidad
      _estadisticas = await _dao.obtenerEstadisticas();

      print('✅ ${_ventasHoy.length} ventas de hoy');
      print('📊 Total: \$${totalVentasHoy}');
      print('📊 Completadas: $ventasCompletadasHoy');
    } catch (e) {
      _error = 'Error al cargar ventas de hoy:
${e.toString()}';
      print('❌ $_error');
    } finally {
      _cargando = false;
      notifyListeners();
    }
  }
```

```
  /// Cargar ventas por rango de fechas

  Future<void> cargarPorFecha(DateTime inicio, DateTime fin)
async {

    _cargando = true;

    _error = null;

    notifyListeners();



    try {

      _ventas = await _dao.obtenerPorFechas(inicio, fin);

      print('✅ ${_ventas.length} ventas entre
${inicio.toLocal()} y ${fin.toLocal()}');

    } catch (e) {

      _error = 'Error al cargar ventas: ${e.toString()}';

      print('❌ $_error');

    } finally {

      _cargando = false;

      notifyListeners();

    }

  }



  /// Crear venta con detalles

  Future<bool> crearVenta(Venta venta, List<DetalleVenta>
detalles) async {

    _cargando = true;
```

```
_error = null;
notifyListeners();

try {
  print('=== VentaViewModel.crearVenta ===');
  print('Cliente ID: ${venta.clienteId}');
  print('Vendedor ID: ${venta.vendedorId}');
  print('Total: \$${venta.total}');
  print('Cantidad detalles: ${detalles.length}');

  // Asignar detalles a la venta
  venta.detalles = detalles;

  // Insertar venta
  final ventaId = await _dao.insertar(venta);

  if (ventaId != null) {
    print('✅ Venta creada con ID: $ventaId');

    // Recargar ventas
    await cargarVentas();

    _cargando = false;
    notifyListeners();
```

```
      return true;
    } else {
      _error = 'Error al crear venta';
      print('❌ $_error');
      _cargando = false;
      notifyListeners();
      return false;
    }
  } catch (e) {
    _error = 'Error al crear venta: ${e.toString()}';
    print('❌ $_error');
    _cargando = false;
    notifyListeners();
    return false;
  }
}

/// Actualizar venta (solo datos principales, no detalles)
Future<bool> actualizarVenta(Venta venta) async {
  try {
    if (venta.id == null) {
      _error = 'Venta sin ID';
      notifyListeners();
      return false;
```

```
    }

    final exitoso = await _dao.actualizar(venta);

    if (exitoso) {
      await cargarVentas();
      print('✅ Venta actualizada');
      return true;
    } else {
      _error = 'Error al actualizar venta';
      notifyListeners();
      return false;
    }
  } catch (e) {
    _error = 'Error al actualizar venta: ${e.toString()}';
    print('❌ $_error');
    notifyListeners();
    return false;
  }
}

/// Cancelar/Eliminar venta
Future<bool> cancelarVenta(int id) async {
  try {
```

```
    final exitoso = await _dao.eliminar(id);

    if (exitoso) {
      await cargarVentas();
      print('✅ Venta cancelada');
      return true;
    } else {
      _error = 'Error al cancelar venta';
      notifyListeners();
      return false;
    }
  } catch (e) {
    _error = 'Error al cancelar venta: ${e.toString()}';
    print('❌ $_error');
    notifyListeners();
    return false;
  }
}

/// Obtener venta completa con detalles
Future<Venta?> obtenerVentaDetalle(int id) async {
  try {
    return await _dao.obtenerPorId(id);
  } catch (e) {
```

```
    _error = 'Error al obtener venta: ${e.toString()}';
    print('❌ $_error');
    notifyListeners();
    return null;
  }
}

/// Obtener total de ventas
Future<double> obtenerTotalVentas() async {
  try {
    return await _dao.obtenerTotalVentas();
  } catch (e) {
    print('❌ Error al obtener total de ventas: $e');
    return 0.0;
  }
}

/// Obtener estadísticas generales
Future<void> cargarEstadisticas() async {
  try {
    _estadisticas = await _dao.obtenerEstadisticas();
    notifyListeners();
  } catch (e) {
    print('❌ Error al cargar estadísticas: $e');
```

```
    }
  }


  /// Obtener estadísticas del cliente
  Map<String, dynamic> obtenerEstadisticasCliente(int
clienteId) {
    final ventasCliente = _ventas
        .where((venta) => venta.clienteId == clienteId)
        .toList();


    final totalGastado = ventasCliente.fold<double>(
      0.0,
      (sum, venta) => sum + venta.total,
    );


    final totalCompras = ventasCliente.length;


    return {
      'total_gastado': totalGastado,
      'total_compras': totalCompras,
      'promedio_compra': totalCompras > 0 ? totalGastado /
totalCompras : 0,
    };
  }
```

```
  /// Obtener estadísticas del vendedor

  Map<String, dynamic> obtenerEstadisticasVendedor(int
vendedorId) {

    final ventasVendedor = _ventas

        .where((venta) => venta.vendedorId == vendedorId)

        .toList();



    final totalVendido = ventasVendedor.fold<double>(

      0.0,

      (sum, venta) => sum + venta.total,

    );



    final totalVentas = ventasVendedor.length;



    return {

      'total_vendido': totalVendido,

      'total_ventas': totalVentas,

      'promedio_venta': totalVentas > 0 ? totalVendido /
totalVentas : 0,

    };

  }



  /// Limpiar error

  void limpiarError() {

    _error = null;
```

```
    notifyListeners();
  }

  @override
  void dispose() {
    super.dispose();
  }
}
```

9 Capa de Datos DAO

9.1 Cliente DAO

```
import '../database_helper.dart';
import '../models/cliente.dart';

class ClienteDao {
  final DatabaseHelper _dbHelper = DatabaseHelper.instance;

  // Obtener todos los clientes
  Future<List<Cliente>> getAllClientes() async {
    final db = await _dbHelper.database;
    final maps = await db.query(
      'usuarios',
      where: 'rol = ?',
      whereArgs: ['cliente'],
      orderBy: 'nombre ASC',
    );

    return maps.map((map) => Cliente.fromMap(map)).toList();
  }

  // Obtener cliente por ID
  Future<Cliente?> getClienteById(int id) async {
    final db = await _dbHelper.database;
    final maps = await db.query(
      'usuarios',
      where: 'id = ? AND rol = ?',
      whereArgs: [id, 'cliente'],
    );

    if (maps.isNotEmpty) {
      return Cliente.fromMap(maps.first);
    }
    return null;
  }
```

```
  // Buscar clientes por nombre o teléfono
  Future<List<Cliente>> searchClientes(String query) async {
    final db = await _dbHelper.database;
    final maps = await db.query(
      'usuarios',
      where: 'rol = ? AND (nombre LIKE ? OR telefono LIKE ?)',
      whereArgs: ['cliente', '%$query%', '%$query%'],
      orderBy: 'nombre ASC',
    );

    return maps.map((map) => Cliente.fromMap(map)).toList();
  }

  // Insertar nuevo cliente
  Future<int> insertCliente(Cliente cliente) async {
    final db = await _dbHelper.database;
    return await db.insert('usuarios', cliente.toMap());
  }

  // Actualizar cliente
  Future<int> updateCliente(Cliente cliente) async {
    final db = await _dbHelper.database;
    return await db.update(
      'usuarios',
      cliente.toMap(),
      where: 'id = ?',
      whereArgs: [cliente.id],
    );
  }

  // Eliminar cliente
  Future<int> deleteCliente(int id) async {
    final db = await _dbHelper.database;
    return await db.delete(
      'usuarios',
      where: 'id = ?',
      whereArgs: [id],
    );
  }
}
```

9.2 **Detalle Pedido DAO**

```
class DetallePedido {
  final int? id;
  final int pedidoId;
  final int productoId;
  final int cantidad;
  final double precioUnitario;
  final double subtotal;
  final double descuento;
  final double total;
```

```
  // Para consultas con JOIN
  String? nombreProducto;
  String? codigoProducto;

  DetallePedido({
    this.id,
    required this.pedidoId,
    required this.productoId,
    required this.cantidad,
    required this.precioUnitario,
    required this.subtotal,
    this.descuento = 0,
    required this.total,
    this.nombreProducto,
    this.codigoProducto,
  });

  Map<String, dynamic> toMap() {
    return {
      'id': id,
      'pedido_id': pedidoId,
      'producto_id': productoId,
      'cantidad': cantidad,
      'precio_unitario': precioUnitario,
      'subtotal': subtotal,
      'descuento': descuento,
      'total': total,
    };
  }

  factory DetallePedido.fromMap(Map<String, dynamic> map) {
    return DetallePedido(
      id: map['id'],
      pedidoId: map['pedido_id'],
      productoId: map['producto_id'],
      cantidad: map['cantidad'],
      precioUnitario: map['precio_unitario'].toDouble(),
      subtotal: map['subtotal'].toDouble(),
      descuento: map['descuento']?.toDouble() ?? 0,
      total: map['total'].toDouble(),
      nombreProducto: map['nombre_producto'],
      codigoProducto: map['codigo_producto'],
    );
  }
}
```

9.3 **Pedido Dao**

```
import 'package:sqflite/sqflite.dart';
import '../database_helper.dart';
import '../models/pedido.dart';
import '../models/detalle_pedido.dart';
```

```
class PedidoDao {
  final DatabaseHelper _dbHelper = DatabaseHelper.instance;

  // Crear pedido con sus detalles
  Future<int> crear(Pedido pedido, List<DetallePedido> detalles) async {
    final db = await _dbHelper.database;
    int pedidoId = 0;

    await db.transaction((txn) async {
      pedidoId = await txn.insert('pedidos', pedido.toMap());

      for (var detalle in detalles) {
        final detalleConPedidoId = DetallePedido(
          pedidoId: pedidoId,
          productoId: detalle.productoId,
          cantidad: detalle.cantidad,
          precioUnitario: detalle.precioUnitario,
          subtotal: detalle.subtotal,
          descuento: detalle.descuento,
          total: detalle.total,
        );

        await txn.insert('detalle_pedidos', detalleConPedidoId.toMap());
      }
    });

    return pedidoId;
  }

  // Obtener pedido por ID con detalles
  Future<Pedido?> obtenerPorId(int id) async {
    final db = await _dbHelper.database;

    final pedidoMaps = await db.rawQuery('''
      SELECT p.*,
             c.nombre as nombre_cliente,
             v.nombre as nombre_vendedor
      FROM pedidos p
      INNER JOIN usuarios c ON p.cliente_id = c.id
      LEFT JOIN usuarios v ON p.vendedor_id = v.id
      WHERE p.id = ?
    ''', [id]);

    if (pedidoMaps.isEmpty) return null;

    final pedido = Pedido.fromMap(pedidoMaps.first);

    final detallesMaps = await db.rawQuery('''
      SELECT dp.*,
             pr.nombre as nombre_producto,
             pr.codigo as codigo_producto
      FROM detalle_pedidos dp
      INNER JOIN productos pr ON dp.producto_id = pr.id
      WHERE dp.pedido_id = ?
```

```
  ''', [id]);

  pedido.detalles =
      detallesMaps.map((map) => DetallePedido.fromMap(map)).toList();

  return pedido;
}

// Obtener todos los pedidos
Future<List<Pedido>> obtenerTodos() async {
  final db = await _dbHelper.database;
  final resultado = await db.rawQuery('''
    SELECT p.*,
           c.nombre as nombre_cliente,
           v.nombre as nombre_vendedor
    FROM pedidos p
    INNER JOIN usuarios c ON p.cliente_id = c.id
    LEFT JOIN usuarios v ON p.vendedor_id = v.id
    ORDER BY p.fecha_pedido DESC
  ''');

  return resultado.map((map) => Pedido.fromMap(map)).toList();
}

// Obtener pedidos por estado
Future<List<Pedido>> obtenerPorEstado(String estado) async {
  final db = await _dbHelper.database;
  final resultado = await db.rawQuery('''
    SELECT p.*,
           c.nombre as nombre_cliente,
           v.nombre as nombre_vendedor
    FROM pedidos p
    INNER JOIN usuarios c ON p.cliente_id = c.id
    LEFT JOIN usuarios v ON p.vendedor_id = v.id
    WHERE p.estado = ?
    ORDER BY p.fecha_pedido DESC
  ''', [estado]);

  return resultado.map((map) => Pedido.fromMap(map)).toList();
}

// Obtener pedidos por cliente
Future<List<Pedido>> obtenerPorCliente(int clienteId) async {
  final db = await _dbHelper.database;
  final resultado = await db.rawQuery('''
    SELECT p.*,
           c.nombre as nombre_cliente,
           v.nombre as nombre_vendedor
    FROM pedidos p
    INNER JOIN usuarios c ON p.cliente_id = c.id
    LEFT JOIN usuarios v ON p.vendedor_id = v.id
    WHERE p.cliente_id = ?
    ORDER BY p.fecha_pedido DESC
  ''', [clienteId]);
```

```
  return resultado.map((map) => Pedido.fromMap(map)).toList();
}

// Obtener pedidos por vendedor
Future<List<Pedido>> obtenerPorVendedor(int vendedorId) async {
  final db = await _dbHelper.database;
  final resultado = await db.rawQuery('''
    SELECT p.*,
           c.nombre as nombre_cliente,
           v.nombre as nombre_vendedor
    FROM pedidos p
    INNER JOIN usuarios c ON p.cliente_id = c.id
    LEFT JOIN usuarios v ON p.vendedor_id = v.id
    WHERE p.vendedor_id = ?
    ORDER BY p.fecha_pedido DESC
  ''', [vendedorId]);

  return resultado.map((map) => Pedido.fromMap(map)).toList();
}

// Obtener pedidos por rango de fechas
Future<List<Pedido>> obtenerPorFecha(
    DateTime fechaInicio, DateTime fechaFin) async {
  final db = await _dbHelper.database;
  final resultado = await db.rawQuery('''
    SELECT p.*,
           c.nombre as nombre_cliente,
           v.nombre as nombre_vendedor
    FROM pedidos p
    INNER JOIN usuarios c ON p.cliente_id = c.id
    LEFT JOIN usuarios v ON p.vendedor_id = v.id
    WHERE p.fecha_pedido BETWEEN ? AND ?
    ORDER BY p.fecha_pedido DESC
  ''', [fechaInicio.toIso8601String(), fechaFin.toIso8601String()]);

  return resultado.map((map) => Pedido.fromMap(map)).toList();
}

// Buscar pedido por número
Future<Pedido?> obtenerPorNumero(String numeroPedido) async {
  final db = await _dbHelper.database;
  final resultado = await db.rawQuery('''
    SELECT p.*,
           c.nombre as nombre_cliente,
           v.nombre as nombre_vendedor
    FROM pedidos p
    INNER JOIN usuarios c ON p.cliente_id = c.id
    LEFT JOIN usuarios v ON p.vendedor_id = v.id
    WHERE p.numero_pedido = ?
  ''', [numeroPedido]);

  if (resultado.isEmpty) return null;
```

```
  final pedido = Pedido.fromMap(resultado.first);

  // Cargar detalles
  final detallesMaps = await db.rawQuery('''
    SELECT dp.*,
           pr.nombre as nombre_producto,
           pr.codigo as codigo_producto
    FROM detalle_pedidos dp
    INNER JOIN productos pr ON dp.producto_id = pr.id
    WHERE dp.pedido_id = ?
  ''', [pedido.id]);

  pedido.detalles =
      detallesMaps.map((map) => DetallePedido.fromMap(map)).toList();

  return pedido;
}

// Actualizar estado del pedido
Future<int> actualizarEstado(int id, String estado) async {
  final db = await _dbHelper.database;
  final map = {'estado': estado};

  if (estado == 'entregado') {
    map['fecha_entrega'] = DateTime.now().toIso8601String();
  }

  return await db.update(
    'pedidos',
    map,
    where: 'id = ?',
    whereArgs: [id],
  );
}

// Actualizar pedido
Future<int> actualizar(Pedido pedido) async {
  final db = await _dbHelper.database;
  return await db.update(
    'pedidos',
    pedido.toMap(),
    where: 'id = ?',
    whereArgs: [pedido.id],
  );
}

// Actualizar pedido con sus detalles
Future<void> actualizarConDetalles(
    Pedido pedido, List<DetallePedido> detalles) async {
  final db = await _dbHelper.database;

  await db.transaction((txn) async {
    // Actualizar el pedido
    await txn.update(
```

```
      'pedidos',
      pedido.toMap(),
      where: 'id = ?',
      whereArgs: [pedido.id],
    );

    // Eliminar detalles anteriores
    await txn.delete(
      'detalle_pedidos',
      where: 'pedido_id = ?',
      whereArgs: [pedido.id],
    );

    // Insertar nuevos detalles
    for (var detalle in detalles) {
      await txn.insert('detalle_pedidos', detalle.toMap());
    }
  });
}

// Cancelar pedido
Future<int> cancelar(int id) async {
  return await actualizarEstado(id, 'cancelado');
}

// Eliminar pedido (con sus detalles por CASCADE)
Future<int> eliminar(int id) async {
  final db = await _dbHelper.database;
  return await db.delete(
    'pedidos',
    where: 'id = ?',
    whereArgs: [id],
  );
}

// Contar pedidos totales
Future<int> contar() async {
  final db = await _dbHelper.database;
  final resultado =
      await db.rawQuery('SELECT COUNT(*) as total FROM pedidos');
  return Sqflite.firstIntValue(resultado) ?? 0;
}

// Contar pedidos por estado
Future<int> contarPorEstado(String estado) async {
  final db = await _dbHelper.database;
  final resultado = await db.rawQuery(
    'SELECT COUNT(*) as total FROM pedidos WHERE estado = ?',
    [estado],
  );
  return Sqflite.firstIntValue(resultado) ?? 0;
}

// Obtener total de ventas por período
```

```
  Future<double> totalPorPeriodo(DateTime inicio, DateTime fin) async {
    final db = await _dbHelper.database;
    final resultado = await db.rawQuery('''
      SELECT COALESCE(SUM(total), 0) as total
      FROM pedidos
      WHERE fecha_pedido BETWEEN ? AND ?
        AND estado != 'cancelado'
    ''', [inicio.toIso8601String(), fin.toIso8601String()]);

    return (resultado.first['total'] as num?)?.toDouble() ?? 0.0;
  }

  // Generar número de pedido único
  Future<String> generarNumeroPedido() async {
    final db = await _dbHelper.database;
    final hoy = DateTime.now();
    final fechaStr = hoy.toIso8601String().substring(0, 10);

    final resultado = await db.rawQuery(
      'SELECT COUNT(*) as total FROM pedidos WHERE fecha_pedido >= ?',
      [fechaStr],
    );

    final total = (Sqflite.firstIntValue(resultado) ?? 0) + 1;
    return 'PED-${hoy.year}${hoy.month.toString().padLeft(2,
'0')}${hoy.day.toString().padLeft(2, '0')}-${total.toString().padLeft(4,
'0')}';
  }

  // Obtener pedidos pendientes
  Future<List<Pedido>> obtenerPendientes() async {
    return await obtenerPorEstado('pendiente');
  }

  // Obtener pedidos en proceso
  Future<List<Pedido>> obtenerEnProceso() async {
    return await obtenerPorEstado('procesando');
  }

  // Obtener estadísticas de pedidos
  Future<Map<String, dynamic>> obtenerEstadisticas() async {
    final db = await _dbHelper.database;

    final resultado = await db.rawQuery('''
      SELECT
        COUNT(*) as total_pedidos,
        SUM(CASE WHEN estado = 'pendiente' THEN 1 ELSE 0 END) as pendientes,
        SUM(CASE WHEN estado = 'procesando' THEN 1 ELSE 0 END) as procesando,
        SUM(CASE WHEN estado = 'enviado' THEN 1 ELSE 0 END) as enviados,
        SUM(CASE WHEN estado = 'entregado' THEN 1 ELSE 0 END) as entregados,
        SUM(CASE WHEN estado = 'cancelado' THEN 1 ELSE 0 END) as cancelados,
        COALESCE(SUM(CASE WHEN estado != 'cancelado' THEN total ELSE 0 END),
0) as total_ventas
      FROM pedidos
```

```
    ''');

    if (resultado.isNotEmpty) {
      final data = resultado.first;
      return {
        'total_pedidos': data['total_pedidos'] ?? 0,
        'pendientes': data['pendientes'] ?? 0,
        'procesando': data['procesando'] ?? 0,
        'enviados': data['enviados'] ?? 0,
        'entregados': data['entregados'] ?? 0,
        'cancelados': data['cancelados'] ?? 0,
        'total_ventas': (data['total_ventas'] as num?)?.toDouble() ?? 0.0,
      };
    }

    return {
      'total_pedidos': 0,
      'pendientes': 0,
      'procesando': 0,
      'enviados': 0,
      'entregados': 0,
      'cancelados': 0,
      'total_ventas': 0.0,
    };
  }
}
```

9.4 **Producto DAO**

```
// data/dao/producto_dao.dart
import 'package:sqflite/sqflite.dart';
import '../database_helper.dart';
import '../models/producto.dart';

class ProductoDao {
  final DatabaseHelper _dbHelper = DatabaseHelper.instance;

  // Crear producto
  Future<int> crear(Producto producto) async {
    final db = await _dbHelper.database;
    return await db.insert('productos', producto.toMap());
  }

  // Obtener producto por ID
  Future<Producto?> obtenerPorId(int id) async {
    final db = await _dbHelper.database;
    final maps = await db.query(
      'productos',
      where: 'id = ?',
      whereArgs: [id],
    );

    if (maps.isNotEmpty) {
```

```
    return Producto.fromMap(maps.first);
  }
  return null;
}

// Obtener producto por código
Future<Producto?> obtenerPorCodigo(String codigo) async {
  final db = await _dbHelper.database;
  final maps = await db.query(
    'productos',
    where: 'codigo = ?',
    whereArgs: [codigo],
  );

  if (maps.isNotEmpty) {
    return Producto.fromMap(maps.first);
  }
  return null;
}

// Obtener todos los productos
Future<List<Producto>> obtenerTodos() async {
  final db = await _dbHelper.database;
  final resultado = await db.query('productos', orderBy: 'nombre ASC');
  return resultado.map((map) => Producto.fromMap(map)).toList();
}

// Obtener productos activos
Future<List<Producto>> obtenerActivos() async {
  final db = await _dbHelper.database;
  final resultado = await db.query(
    'productos',
    where: 'activo = 1',
    orderBy: 'nombre ASC',
  );
  return resultado.map((map) => Producto.fromMap(map)).toList();
}

// Obtener productos por categoría
Future<List<Producto>> obtenerPorCategoria(String categoria) async {
  final db = await _dbHelper.database;
  final resultado = await db.query(
    'productos',
    where: 'categoria = ? AND activo = 1',
    whereArgs: [categoria],
    orderBy: 'nombre ASC',
  );
  return resultado.map((map) => Producto.fromMap(map)).toList();
}

// Obtener productos con bajo stock
Future<List<Producto>> obtenerBajoStock() async {
  final db = await _dbHelper.database;
  final resultado = await db.rawQuery(
```

```
      'SELECT * FROM productos WHERE stock <= stock_minimo AND activo = 1
ORDER BY stock ASC',
    );
    return resultado.map((map) => Producto.fromMap(map)).toList();
  }

  // Actualizar producto
  Future<int> actualizar(Producto producto) async {
    final db = await _dbHelper.database;
    return await db.update(
      'productos',
      producto.toMap(),
      where: 'id = ?',
      whereArgs: [producto.id],
    );
  }

  // Actualizar stock
  Future<int> actualizarStock(int id, int nuevoStock) async {
    final db = await _dbHelper.database;
    return await db.update(
      'productos',
      {
        'stock': nuevoStock,
        'fecha_actualizacion': DateTime.now().toIso8601String(),
      },
      where: 'id = ?',
      whereArgs: [id],
    );
  }

  // Incrementar stock
  Future<int> incrementarStock(int id, int cantidad) async {
    final db = await _dbHelper.database;
    await db.rawUpdate(
      'UPDATE productos SET stock = stock + ?, fecha_actualizacion = ? WHERE
id = ?',
      [cantidad, DateTime.now().toIso8601String(), id],
    );
    final producto = await obtenerPorId(id);
    return producto?.stock ?? 0;
  }

  // Decrementar stock
  Future<int> decrementarStock(int id, int cantidad) async {
    final db = await _dbHelper.database;
    await db.rawUpdate(
      'UPDATE productos SET stock = stock - ?, fecha_actualizacion = ? WHERE
id = ?',
      [cantidad, DateTime.now().toIso8601String(), id],
    );
    final producto = await obtenerPorId(id);
    return producto?.stock ?? 0;
  }
```

```
// Eliminar producto (soft delete)
Future<int> eliminar(int id) async {
  final db = await _dbHelper.database;
  return await db.update(
    'productos',
    {'activo': 0},
    where: 'id = ?',
    whereArgs: [id],
  );
}

// Eliminar permanentemente
Future<int> eliminarPermanente(int id) async {
  final db = await _dbHelper.database;
  return await db.delete(
    'productos',
    where: 'id = ?',
    whereArgs: [id],
  );
}

// Verificar si existe un código
Future<bool> codigoExiste(String codigo, {int? excluyendoId}) async {
  final db = await _dbHelper.database;
  String where = 'codigo = ?';
  List<dynamic> whereArgs = [codigo];

  if (excluyendoId != null) {
    where += ' AND id != ?';
    whereArgs.add(excluyendoId);
  }

  final resultado = await db.query(
    'productos',
    where: where,
    whereArgs: whereArgs,
  );

  return resultado.isNotEmpty;
}

// Buscar productos
Future<List<Producto>> buscar(String termino) async {
  final db = await _dbHelper.database;
  final resultado = await db.query(
    'productos',
    where: 'nombre LIKE ? OR codigo LIKE ? OR categoria LIKE ?',
    whereArgs: ['%$termino%', '%$termino%', '%$termino%'],
    orderBy: 'nombre ASC',
  );
  return resultado.map((map) => Producto.fromMap(map)).toList();
}
```

```
  // Contar productos
  Future<int> contar() async {
    final db = await _dbHelper.database;
    final resultado = await db.rawQuery('SELECT COUNT(*) as total FROM
productos WHERE activo = 1');
    return Sqflite.firstIntValue(resultado) ?? 0;
  }

  // Valor total del inventario
  Future<double> valorInventario() async {
    final db = await _dbHelper.database;
    final resultado = await db.rawQuery(
      'SELECT SUM(stock * precio_compra) as total FROM productos WHERE activo
= 1',
    );
    return (resultado.first['total'] as num?)?.toDouble() ?? 0.0;
  }
}
```

9.5 Usuario DAO

```
// data/dao/usuario_dao.dart
import 'package:sqflite/sqflite.dart';
import 'package:crypto/crypto.dart';
import 'dart:convert';
import '../database_helper.dart';
import '../models/usuario.dart';

class UsuarioDao {
  final DatabaseHelper _dbHelper = DatabaseHelper.instance;

  String _hash(String password) {
    return sha256.convert(utf8.encode(password)).toString();
  }

  // Crear usuario
  Future<int> crear(Usuario usuario) async {
    final db = await _dbHelper.database;
    return await db.insert('usuarios', usuario.toMap());
  }

  // Obtener usuario por ID
  Future<Usuario?> obtenerPorId(int id) async {
    final db = await _dbHelper.database;
    final maps = await db.query(
      'usuarios',
      where: 'id = ?',
      whereArgs: [id],
    );

    if (maps.isNotEmpty) {
      return Usuario.fromMap(maps.first);
    }
```

```
  return null;
}

// Obtener usuario por email
Future<Usuario?> obtenerPorEmail(String email) async {
  final db = await _dbHelper.database;
  final maps = await db.query(
    'usuarios',
    where: 'email = ?',
    whereArgs: [email],
  );

  if (maps.isNotEmpty) {
    return Usuario.fromMap(maps.first);
  }
  return null;
}

// Login - Verificar credenciales
Future<Usuario?> login(String email, String password) async {
  final db = await _dbHelper.database;
  final passwordHash = _hash(password);
  final maps = await db.query(
    'usuarios',
    where: 'email = ? AND activo = 1 AND (password = ? OR password = ?)',
    whereArgs: [email, password, passwordHash],
  );

  if (maps.isNotEmpty) {
    return Usuario.fromMap(maps.first);
  }
  return null;
}

// Obtener todos los usuarios
Future<List<Usuario>> obtenerTodos() async {
  final db = await _dbHelper.database;
  final resultado = await db.query('usuarios', orderBy: 'nombre ASC');
  return resultado.map((map) => Usuario.fromMap(map)).toList();
}

// Obtener usuarios activos
Future<List<Usuario>> obtenerActivos() async {
  final db = await _dbHelper.database;
  final resultado = await db.query(
    'usuarios',
    where: 'activo = 1',
    orderBy: 'nombre ASC',
  );
  return resultado.map((map) => Usuario.fromMap(map)).toList();
}

// Obtener usuarios por rol
Future<List<Usuario>> obtenerPorRol(String rol) async {
```

```
  final db = await _dbHelper.database;
  final resultado = await db.query(
    'usuarios',
    where: 'rol = ? AND activo = 1',
    whereArgs: [rol],
    orderBy: 'nombre ASC',
  );
  return resultado.map((map) => Usuario.fromMap(map)).toList();
}

// Obtener clientes
Future<List<Usuario>> obtenerClientes() async {
  return await obtenerPorRol('cliente');
}

// Obtener vendedores
Future<List<Usuario>> obtenerVendedores() async {
  return await obtenerPorRol('vendedor');
}

// Obtener administradores
Future<List<Usuario>> obtenerAdministradores() async {
  return await obtenerPorRol('admin');
}

// Actualizar usuario
Future<int> actualizar(Usuario usuario) async {
  final db = await _dbHelper.database;
  return await db.update(
    'usuarios',
    usuario.toMap(),
    where: 'id = ?',
    whereArgs: [usuario.id],
  );
}

// Cambiar contraseña
Future<int> cambiarPassword(int id, String nuevoPassword) async {
  final db = await _dbHelper.database;
  return await db.update(
    'usuarios',
    {'password': nuevoPassword},
    where: 'id = ?',
    whereArgs: [id],
  );
}

// Actualizar perfil (sin cambiar password)
Future<int> actualizarPerfil(int id, {
  String? nombre,
  String? email,
  String? telefono,
  String? direccion,
}) async {
```

```
  final db = await _dbHelper.database;
  final Map<String, dynamic> datos = {};

  if (nombre != null) datos['nombre'] = nombre;
  if (email != null) datos['email'] = email;
  if (telefono != null) datos['telefono'] = telefono;
  if (direccion != null) datos['direccion'] = direccion;

  if (datos.isEmpty) return 0;

  return await db.update(
    'usuarios',
    datos,
    where: 'id = ?',
    whereArgs: [id],
  );
}

// Desactivar usuario (soft delete)
Future<int> desactivar(int id) async {
  final db = await _dbHelper.database;
  return await db.update(
    'usuarios',
    {'activo': 0},
    where: 'id = ?',
    whereArgs: [id],
  );
}

// Activar usuario
Future<int> activar(int id) async {
  final db = await _dbHelper.database;
  return await db.update(
    'usuarios',
    {'activo': 1},
    where: 'id = ?',
    whereArgs: [id],
  );
}

// Eliminar usuario permanentemente
Future<int> eliminarPermanente(int id) async {
  final db = await _dbHelper.database;
  return await db.delete(
    'usuarios',
    where: 'id = ?',
    whereArgs: [id],
  );
}

// Verificar si existe un email
Future<bool> emailExiste(String email, {int? excluyendoId}) async {
  final db = await _dbHelper.database;
  String where = 'email = ?';
```

```
    List<dynamic> whereArgs = [email];

    if (excluyendoId != null) {
      where += ' AND id != ?';
      whereArgs.add(excluyendoId);
    }

    final resultado = await db.query(
      'usuarios',
      where: where,
      whereArgs: whereArgs,
    );

    return resultado.isNotEmpty;
  }

  // Buscar usuarios
  Future<List<Usuario>> buscar(String termino) async {
    final db = await _dbHelper.database;
    final resultado = await db.query(
      'usuarios',
      where: 'nombre LIKE ? OR email LIKE ? OR telefono LIKE ?',
      whereArgs: ['%$termino%', '%$termino%', '%$termino%'],
      orderBy: 'nombre ASC',
    );
    return resultado.map((map) => Usuario.fromMap(map)).toList();
  }

  // Contar usuarios
  Future<int> contar() async {
    final db = await _dbHelper.database;
    final resultado = await db.rawQuery('SELECT COUNT(*) as total FROM
usuarios WHERE activo = 1');
    return Sqflite.firstIntValue(resultado) ?? 0;
  }

  // Contar usuarios por rol
  Future<int> contarPorRol(String rol) async {
    final db = await _dbHelper.database;
    final resultado = await db.rawQuery(
      'SELECT COUNT(*) as total FROM usuarios WHERE rol = ? AND activo = 1',
      [rol],
    );
    return Sqflite.firstIntValue(resultado) ?? 0;
  }

  // Obtener estadísticas de ventas por vendedor
  Future<Map<String, dynamic>> estadisticasVendedor(int vendedorId, DateTime
fechaInicio, DateTime fechaFin) async {
    final db = await _dbHelper.database;

    final resultado = await db.rawQuery('''
      SELECT
        COUNT(*) as total_ventas,
```

```
        COALESCE(SUM(total), 0) as total_vendido,
        COALESCE(AVG(total), 0) as promedio_venta
      FROM ventas
      WHERE vendedor_id = ?
        AND fecha BETWEEN ? AND ?
        AND estado = 'completada'
    ''', [vendedorId, fechaInicio.toIso8601String(),
fechaFin.toIso8601String()]);

    if (resultado.isNotEmpty) {
      return {
        'total_ventas': resultado.first['total_ventas'] ?? 0,
        'total_vendido': (resultado.first['total_vendido'] as
num?)?.toDouble() ?? 0.0,
        'promedio_venta': (resultado.first['promedio_venta'] as
num?)?.toDouble() ?? 0.0,
      };
    }

    return {
      'total_ventas': 0,
      'total_vendido': 0.0,
      'promedio_venta': 0.0,
    };
  }

  // Obtener historial de compras de un cliente
  Future<List<Map<String, dynamic>>> historialCliente(int clienteId) async {
    final db = await _dbHelper.database;

    return await db.rawQuery('''
      SELECT
        v.id,
        v.fecha,
        v.total,
        v.estado,
        ve.nombre as nombre_vendedor
      FROM ventas v
      LEFT JOIN usuarios ve ON v.vendedor_id = ve.id
      WHERE v.cliente_id = ?
      ORDER BY v.fecha DESC
    ''', [clienteId]);
  }

  // Obtener pedidos de un cliente
  Future<List<Map<String, dynamic>>> pedidosCliente(int clienteId) async {
    final db = await _dbHelper.database;

    return await db.rawQuery('''
      SELECT
        p.id,
        p.fecha_pedido,
        p.fecha_entrega,
        p.total,
```

```
        p.estado,
        v.nombre as nombre_vendedor
      FROM pedidos p
      LEFT JOIN usuarios v ON p.vendedor_id = v.id
      WHERE p.cliente_id = ?
      ORDER BY p.fecha_pedido DESC
    ''', [clienteId]);
  }

  // Validar credenciales (para cambio de password)
  Future<bool> validarCredenciales(int id, String passwordActual) async {
    final db = await _dbHelper.database;
    final hash = _hash(passwordActual);
    final resultado = await db.query(
      'usuarios',
      where: 'id = ? AND (password = ? OR password = ?)',
      whereArgs: [id, passwordActual, hash],
    );

    return resultado.isNotEmpty;
  }

  // Obtener último usuario registrado
  Future<Usuario?> obtenerUltimo() async {
    final db = await _dbHelper.database;
    final resultado = await db.query(
      'usuarios',
      orderBy: 'fecha_registro DESC',
      limit: 1,
    );

    if (resultado.isNotEmpty) {
      return Usuario.fromMap(resultado.first);
    }
    return null;
  }

  // Cambiar rol de usuario
  Future<int> cambiarRol(int id, String nuevoRol) async {
    final db = await _dbHelper.database;
    return await db.update(
      'usuarios',
      {'rol': nuevoRol},
      where: 'id = ?',
      whereArgs: [id],
    );
  }
}
```

9.6 **Venta DAO**

```
import 'package:sqflite/sqflite.dart';
  import '../database_helper.dart';
```

```dart
import '../models/venta.dart';
import '../models/detalle_venta.dart';

class VentaDao {
  final DatabaseHelper _dbHelper = DatabaseHelper.instance;

  // Crear venta con sus detalles (transacción)
  Future<int> crear(Venta venta, List<DetalleVenta> detalles) async {
    final db = await _dbHelper.database;
    int ventaId = 0;

    print('=== VentaDAO.crear ===');
    print('Insertando venta: ${venta.numeroVenta}');
    print('Mapa de venta: ${venta.toMap()}');

    await db.transaction((txn) async {
      try {
        // Insertar venta
        ventaId = await txn.insert('ventas', venta.toMap());
        print('✅ Venta insertada con ID: $ventaId');

        // Insertar detalles
        for (var i = 0; i < detalles.length; i++) {
          final detalle = detalles[i];
          final detalleConVentaId = DetalleVenta(
            ventaId: ventaId,
            productoId: detalle.productoId,
            cantidad: detalle.cantidad,
            precioUnitario: detalle.precioUnitario,
            subtotal: detalle.subtotal,
            descuento: detalle.descuento,
            total: detalle.total,
          );

          print('Insertando detalle $i: Producto ${detalle.productoId},
Cantidad ${detalle.cantidad}');
          await txn.insert('detalle_ventas', detalleConVentaId.toMap());

          // Actualizar stock del producto
          await txn.rawUpdate(
            'UPDATE productos SET stock = stock - ? WHERE id = ?',
            [detalle.cantidad, detalle.productoId],
          );
        }
        print('✅ Todos los detalles insertados correctamente');
      } catch (e) {
        print('❌ Error en transacción: $e');
        rethrow;
      }
    });

    return ventaId;
  }
```

```
    // Obtener venta por ID con detalles
    Future<Venta?> obtenerPorId(int id) async {
      final db = await _dbHelper.database;

      // Obtener venta con información de cliente y vendedor
      final ventaMaps = await db.rawQuery('''
        SELECT v.*,
               c.nombre as nombre_cliente,
               ve.nombre as nombre_vendedor
        FROM ventas v
        LEFT JOIN usuarios c ON v.cliente_id = c.id
        LEFT JOIN usuarios ve ON v.vendedor_id = ve.id
        WHERE v.id = ?
      ''', [id]);

      if (ventaMaps.isEmpty) return null;

      final venta = Venta.fromMap(ventaMaps.first);

      // Obtener detalles
      final detallesMaps = await db.rawQuery('''
        SELECT dv.*,
               p.nombre as nombre_producto,
               p.codigo as codigo_producto
        FROM detalle_ventas dv
        INNER JOIN productos p ON dv.producto_id = p.id
        WHERE dv.venta_id = ?
      ''', [id]);

      venta.detalles = detallesMaps.map((map) =>
DetalleVenta.fromMap(map)).toList();

      return venta;
    }

    // Obtener todas las ventas
    Future<List<Venta>> obtenerTodas() async {
      final db = await _dbHelper.database;
      final resultado = await db.rawQuery('''
        SELECT v.*,
               c.nombre as nombre_cliente,
               ve.nombre as nombre_vendedor
        FROM ventas v
        LEFT JOIN usuarios c ON v.cliente_id = c.id
        LEFT JOIN usuarios ve ON v.vendedor_id = ve.id
        ORDER BY v.fecha DESC
      ''');

      return resultado.map((map) => Venta.fromMap(map)).toList();
    }

    // Obtener ventas por fecha
    Future<List<Venta>> obtenerPorFecha(DateTime fechaInicio, DateTime
fechaFin) async {
```

```
  final db = await _dbHelper.database;
  final resultado = await db.rawQuery('''
    SELECT v.*,
          c.nombre as nombre_cliente,
          ve.nombre as nombre_vendedor
    FROM ventas v
    LEFT JOIN usuarios c ON v.cliente_id = c.id
    LEFT JOIN usuarios ve ON v.vendedor_id = ve.id
    WHERE v.fecha BETWEEN ? AND ?
    ORDER BY v.fecha DESC
  ''', [fechaInicio.toIso8601String(), fechaFin.toIso8601String()]);

  return resultado.map((map) => Venta.fromMap(map)).toList();
}

// Obtener ventas por vendedor
Future<List<Venta>> obtenerPorVendedor(int vendedorId) async {
  final db = await _dbHelper.database;
  final resultado = await db.rawQuery('''
    SELECT v.*,
          c.nombre as nombre_cliente,
          ve.nombre as nombre_vendedor
    FROM ventas v
    LEFT JOIN usuarios c ON v.cliente_id = c.id
    LEFT JOIN usuarios ve ON v.vendedor_id = ve.id
    WHERE v.vendedor_id = ?
    ORDER BY v.fecha DESC
  ''', [vendedorId]);

  return resultado.map((map) => Venta.fromMap(map)).toList();
}

// Actualizar estado de venta
Future<int> actualizarEstado(int id, String estado) async {
  final db = await _dbHelper.database;
  return await db.update(
    'ventas',
    {'estado': estado},
    where: 'id = ?',
    whereArgs: [id],
  );
}

// Cancelar venta (devolver stock)
Future<void> cancelar(int id) async {
  final db = await _dbHelper.database;

  await db.transaction((txn) async {
    // Obtener detalles de la venta
    final detalles = await txn.query(
      'detalle_ventas',
      where: 'venta_id = ?',
      whereArgs: [id],
    );
```

```
        // Devolver stock
        for (var detalle in detalles) {
          await txn.rawUpdate(
            'UPDATE productos SET stock = stock + ? WHERE id = ?',
            [detalle['cantidad'], detalle['producto_id']],
          );
        }

        // Actualizar estado
        await txn.update(
          'ventas',
          {'estado': 'cancelada'},
          where: 'id = ?',
          whereArgs: [id],
        );
      });
    }

    // Obtener total de ventas por período
    Future<double> obtenerTotalPorPeriodo(DateTime inicio, DateTime fin)
async {
      final db = await _dbHelper.database;
      final resultado = await db.rawQuery('''
        SELECT SUM(total) as total
        FROM ventas
        WHERE fecha BETWEEN ? AND ? AND estado = 'completada'
      ''', [inicio.toIso8601String(), fin.toIso8601String()]);

      return (resultado.first['total'] as num?)?.toDouble() ?? 0.0;
    }

    // Contar ventas
    Future<int> contar() async {
      final db = await _dbHelper.database;
      final resultado = await db.rawQuery('SELECT COUNT(*) as total FROM
ventas');
      return Sqflite.firstIntValue(resultado) ?? 0;
    }
    // Obtener ventas de hoy
  Future<List<Venta>> obtenerVentasHoy() async {
    final db = await _dbHelper.database;
    final ahora = DateTime.now();
    final fechaHoy = '${ahora.year}-${ahora.month.toString().padLeft(2,
'0')}-${ahora.day.toString().padLeft(2, '0')}';

    print('=== VentaDAO.obtenerVentasHoy ===');
    print('Buscando ventas del día: $fechaHoy');

    final resultado = await db.rawQuery('''
      SELECT v.*,
            c.nombre as nombre_cliente,
            ve.nombre as nombre_vendedor
      FROM ventas v
```

```
      LEFT JOIN usuarios c ON v.cliente_id = c.id
      LEFT JOIN usuarios ve ON v.vendedor_id = ve.id
      WHERE DATE(v.fecha) = ?
      ORDER BY v.fecha DESC
    ''', [fechaHoy]);

    print('Ventas encontradas en BD: ${resultado.length}');

    // Mapear cada venta y cargar sus detalles
    List<Venta> ventas = [];
    for (var map in resultado) {
      final venta = Venta.fromMap(map);

      print('Venta: ${venta.numeroVenta}, Vendedor ID: ${venta.vendedorId},
Total: ${venta.total}');

      // Cargar detalles de la venta
      final detallesMaps = await db.rawQuery('''
        SELECT dv.*,
               p.nombre as nombre_producto,
               p.codigo as codigo_producto
        FROM detalle_ventas dv
        LEFT JOIN productos p ON dv.producto_id = p.id
        WHERE dv.venta_id = ?
      ''', [venta.id]);

      venta.detalles = detallesMaps.map((map) =>
DetalleVenta.fromMap(map)).toList();
      print('  - Detalles cargados: ${venta.detalles.length}');

      ventas.add(venta);
    }

    print('✅ Total de ventas retornadas: ${ventas.length}');
    return ventas;
  }

  // Obtener total de ventas de hoy
  Future<Map<String, dynamic>> obtenerEstadisticasHoy() async {
    final db = await _dbHelper.database;
    final ahora = DateTime.now();
    final fechaHoy = '${ahora.year}-${ahora.month.toString().padLeft(2,
'0')}-${ahora.day.toString().padLeft(2, '0')}';

    final resultado = await db.rawQuery('''
      SELECT
        COUNT(*) as cantidad_ventas,
        COALESCE(SUM(CASE WHEN estado = 'completada' THEN total ELSE 0 END),
0) as total_ventas,
        COALESCE(SUM(CASE WHEN estado = 'completada' THEN 1 ELSE 0 END), 0)
as ventas_completadas,
        COALESCE(SUM(CASE WHEN estado = 'pendiente' THEN 1 ELSE 0 END), 0) as
ventas_pendientes,
        COALESCE(SUM(CASE WHEN estado = 'cancelada' THEN 1 ELSE 0 END), 0) as
```

```
ventas_canceladas
      FROM ventas
      WHERE DATE(fecha) = ?
    ''', [fechaHoy]);

    return resultado.first;
  }
  }
```

10 Modelo de Datos

10.1 Cliente.dart

```
import 'persona_base.dart';

class Cliente implements PersonaBase {
  @override
  final int? id;
  @override
  final String nombre;
  @override
  final String? telefono;
  @override
  final String? email;
  @override
  final String? direccion;
  @override
  final DateTime? fechaCreacion;
  @override
  final bool activo;

  Cliente({
    this.id,
    required this.nombre,
    this.telefono,
    this.email,
    this.direccion,
    this.fechaCreacion,
    this.activo = true,
  });

  Cliente copyWith({
  int? id,
  String? nombre,
  String? telefono,
  String? email,
  String? direccion,
  DateTime? fechaCreacion,
  bool? activo,
}) {
  return Cliente(
    id: id ?? this.id,
    nombre: nombre ?? this.nombre,
    telefono: telefono ?? this.telefono,
```

```
    email: email ?? this.email,
    direccion: direccion ?? this.direccion,
    fechaCreacion: fechaCreacion ?? this.fechaCreacion,
    activo: activo ?? this.activo,
  );
}

  Map<String, dynamic> toMap() {
    return {
      'id': id,
      'nombre': nombre,
      'telefono': telefono,
      'email': email,
      'direccion': direccion,
      'fecha_creacion':
          fechaCreacion?.toIso8601String() ??
DateTime.now().toIso8601String(),
      'activo': activo ? 1 : 0,
    };
  }

  factory Cliente.fromMap(Map<String, dynamic> map) {
    return Cliente(
      id: map['id'],
      nombre: map['nombre'] ?? '',
      telefono: map['telefono'],
      email: map['email'],
      direccion: map['direccion'],
      fechaCreacion: map['fecha_creacion'] != null
          ? DateTime.parse(map['fecha_creacion'])
          : null,
      activo: map['activo'] == 1,
    );
  }
}
```

10.2 **DetallePedido.dart**

```
class DetallePedido {
  final int? id;
  final int? pedidoId;
  final int productoId;
  final int cantidad;
  final double precioUnitario;
  final double subtotal;
  final double descuento;
  final double total;

  String? nombreProducto;
  String? codigoProducto;

  DetallePedido({
    this.id,
    this.pedidoId,
    required this.productoId,
```

```
    required this.cantidad,
    required this.precioUnitario,
    double? subtotal,
    this.descuento = 0,
    double? total,
    this.nombreProducto,
    this.codigoProducto,
  }) : subtotal = subtotal ?? (cantidad * precioUnitario),
       total = total ?? ((cantidad * precioUnitario) - descuento);

  Map<String, dynamic> toMap() {
    return {
      'id': id,
      'pedido_id': pedidoId,
      'producto_id': productoId,
      'cantidad': cantidad,
      'precio_unitario': precioUnitario,
      'subtotal': subtotal,
      'descuento': descuento,
      'total': total,
    };
  }

  factory DetallePedido.fromMap(Map<String, dynamic> map) {
    return DetallePedido(
      id: map['id'],
      pedidoId: map['pedido_id'],
      productoId: map['producto_id'],
      cantidad: map['cantidad'],
      precioUnitario: (map['precio_unitario'] as num).toDouble(),
      subtotal: (map['subtotal'] as num).toDouble(),
      descuento: (map['descuento'] is num)
          ? (map['descuento'] as num).toDouble()
          : 0.0,
      total: (map['total'] as num).toDouble(),
      nombreProducto: map['nombre_producto'],
      codigoProducto: map['codigo_producto'],
    );
  }
}
```

10.3 **DetalleVenta.dart**

```
class DetalleVenta {
  final int? id;
  final int ventaId;
  final int productoId;
  final int cantidad;
  final double precioUnitario;
  final double subtotal;
  final double descuento;
  final double total;
  String? nombreProducto;
  String? codigoProducto;
```

```
  DetalleVenta({
    this.id,
    required this.ventaId,
    required this.productoId,
    required this.cantidad,
    required this.precioUnitario,
    required this.subtotal,
    this.descuento = 0,
    required this.total,
    this.nombreProducto,
    this.codigoProducto,
  });

  Map<String, dynamic> toMap() {
    return {
      'id': id,
      'venta_id': ventaId,
      'producto_id': productoId,
      'cantidad': cantidad,
      'precio_unitario': precioUnitario,
      'subtotal':  subtotal,
      'descuento': descuento,
      'total': total,
    };
  }

  factory DetalleVenta.fromMap(Map<String, dynamic> map) {
    return DetalleVenta(
      id: map['id'],
      ventaId: map['venta_id'],
      productoId: map['producto_id'],
      cantidad: map['cantidad'],
      precioUnitario: (map['precio_unitario'] as num).toDouble(),
      subtotal: (map['subtotal'] as num).toDouble(),
      descuento: (map['descuento'] ?? 0).toDouble(),
      total: (map['total'] as num).toDouble(),
      nombreProducto: map['nombre_producto'],
      codigoProducto: map['codigo_producto'],
    );
  }
}
```

10.4 **PedidoEstado.dart**

```
enum PedidoEstado {
  pendientePago,
  pagado,
  procesando,
  enviado,
  entregado,
  cancelado,
}
```

```
/// Extension para obtener el nombre legible del estado
extension PedidoEstadoExtension on PedidoEstado {
  String get nombre {
    switch (this) {
      case PedidoEstado.pendientePago:
        return 'Pendiente de Pago';
      case PedidoEstado.pagado:
        return 'Pagado';
      case PedidoEstado.procesando:
        return 'Procesando';
      case PedidoEstado.enviado:
        return 'Enviado';
      case PedidoEstado.entregado:
        return 'Entregado';
      case PedidoEstado.cancelado:
        return 'Cancelado';
    }
  }
}
```

10.5 **Pedido.dart**

```
import 'detalle_pedido.dart';
import 'pedido_estado.dart';

class Pedido {
  final int? id;
  final String numeroPedido;
  final DateTime fechaPedido;
  final DateTime? fechaEntrega;
  final int clienteId;
  final int? vendedorId;
  final double subtotal;
  final double descuento;
  final double impuesto;
  final double total;

  /// Cambio: ahora se usa un ENUM en lugar de String
  final PedidoEstado estado;

  final String? direccionEntrega;
  final String? observaciones;

  String? nombreCliente;
  String? nombreVendedor;
  List<DetallePedido>? detalles;

  Pedido({
    this.id,
    required this.numeroPedido,
    required this.fechaPedido,
    this.fechaEntrega,
    required this.clienteId,
    this.vendedorId,
```

```
  required this.subtotal,
  this.descuento = 0,
  this.impuesto = 0,
  required this.total,
  this.estado = PedidoEstado.pendientePago,
  this.direccionEntrega,
  this.observaciones,
  this.nombreCliente,
  this.nombreVendedor,
  this.detalles,
});

Map<String, dynamic> toMap() {
  return {
    'id': id,
    'numero_pedido':  numeroPedido,
    'fecha_pedido': fechaPedido.toIso8601String(),
    'fecha_entrega': fechaEntrega?.toIso8601String(),
    'cliente_id': clienteId,
    'vendedor_id': vendedorId,
    'subtotal': subtotal,
    'descuento': descuento,
    'impuesto': impuesto,
    'total': total,

    /// Guardamos el estado como texto (pendiente, entregado, etc.)
    'estado': estado.name,

    'direccion_entrega': direccionEntrega,
    'observaciones': observaciones,
  };
}

factory Pedido.fromMap(Map<String, dynamic> map) {
  return Pedido(
    id: map['id'],
    numeroPedido: map['numero_pedido'] ?? '',
    fechaPedido: DateTime.parse(map['fecha_pedido']),
    fechaEntrega: map['fecha_entrega'] != null
        ? DateTime.parse(map['fecha_entrega'])
        : null,
    clienteId: map['cliente_id'],
    vendedorId: map['vendedor_id'],
    subtotal: (map['subtotal'] as num).toDouble(),
    descuento: (map['descuento'] ?? 0).toDouble(),
    impuesto: (map['impuesto'] ?? 0).toDouble(),
    total: (map['total'] as num).toDouble(),

    /// Convertimos el String a un enum
    estado: PedidoEstado.values.firstWhere(
      (e) => e.name == (map['estado'] ?? 'pendiente'),
      orElse: () => PedidoEstado.pendientePago,
    ),
```

```
      direccionEntrega: map['direccion_entrega'],
      observaciones: map['observaciones'],
      nombreCliente: map['nombre_cliente'],
      nombreVendedor: map['nombre_vendedor'],
    );
  }
}
```

10.6 **PersonaBase.dart**

```
abstract class PersonaBase {
  int? get id;
  String get nombre;
  String? get telefono;
  String? get email;
  String? get direccion;
  DateTime? get fechaCreacion;
  bool get activo;
}
```

10.7 **Producto.dart**

```
import 'dart:io';

class Producto {
  final int? id;
  final String codigo;
  final String nombre;
  final String? descripcion;
  final String categoria;
  final double precioCompra;
  final double precioVenta;
  final int stock;
  final int stockMinimo;
  final String? proveedor;
  final String? imagenUrl;
  final DateTime? fechaCreacion;
  final DateTime? fechaActualizacion;
  final bool activo;
  final File? imagenLocal;
  final String? informacionMexicana;

  Producto({
    this.id,
    required this.codigo,
    required this.nombre,
    this.descripcion,
    required this.categoria,
    required this.precioCompra,
    required this.precioVenta,
    required this.stock,
    required this.stockMinimo,
    this.proveedor,
    this.imagenUrl,
```

```
    this.fechaCreacion,
    this.fechaActualizacion,
    this.activo = true,
    this.imagenLocal,
    this.informacionMexicana,
  });

  double get margenGanancia => precioVenta - precioCompra;
  double get porcentajeMargen =>
      precioCompra > 0 ? (margenGanancia / precioCompra) * 100 : 0;
  bool get bajoEnStock => stock <= stockMinimo;
  double get valorInventario => stock * precioCompra;
  bool get tieneImagen => imagenUrl != null && imagenUrl!.isNotEmpty;

  Map<String, dynamic> toMap() {
    return {
      'id': id,
      'codigo': codigo,
      'nombre': nombre,
      'descripcion': descripcion,
      'categoria': categoria,
      'precio_compra': precioCompra,
      'precio_venta': precioVenta,
      'stock': stock,
      'stock_minimo': stockMinimo,
      'proveedor': proveedor,
      'imagen_url': imagenUrl,
      'fecha_creacion':
          fechaCreacion?.toIso8601String() ??
DateTime.now().toIso8601String(),
      'fecha_actualizacion': fechaActualizacion?.toIso8601String(),
      'activo': activo, // Usar boolean directo para Supabase
      'informacion_mexicana': informacionMexicana,
    };
  }

  factory Producto.fromMap(Map<String, dynamic> map) {
    bool activoValue = true;
    if (map['activo'] != null) {
      if (map['activo'] is bool) {
        activoValue = map['activo'] as bool;
      } else if (map['activo'] is int) {
        activoValue = map['activo'] == 1;
      } else if (map['activo'] is String) {
        activoValue = map['activo'] == 'true' || map['activo'] == '1';
      }
    }

    return Producto(
      id: map['id'],
      codigo: map['codigo'] ?? '',
      nombre: map['nombre'] ?? '',
      descripcion: map['descripcion'],
      categoria: map['categoria'] ?? '',
```

```
      precioCompra: (map['precio_compra'] as num?)?.toDouble() ?? 0.0,
      precioVenta: (map['precio_venta'] as num?)?.toDouble() ?? 0.0,
      stock: map['stock'] ?? 0,
      stockMinimo: map['stock_minimo'] ?? 0,
      proveedor: map['proveedor'],
      imagenUrl: map['imagen_url'],
      fechaCreacion: map['fecha_creacion'] != null
          ? DateTime.tryParse(map['fecha_creacion'].toString())
          : null,
      fechaActualizacion: map['fecha_actualizacion'] != null
          ? DateTime.tryParse(map['fecha_actualizacion'].toString())
          : null,
      activo: activoValue,
      imagenLocal: null,
      informacionMexicana: map['informacion_mexicana'],
    );
  }
}
```

10.8 **Usuario.dart**

```
import 'persona_base.dart';

class Usuario implements PersonaBase {
  @override
  final int? id;
  @override
  final String nombre;
  @override
  final String email;
  final String password;
  final String rol;
  @override
  final String? telefono;
  @override
  final String? direccion;
  @override
  final DateTime? fechaCreacion;
  final DateTime? fechaActualizacion;
  @override
  final bool activo;

  Usuario({
    this.id,
    required this.nombre,
    required this.email,
    required this.password,
    required this.rol,
    this.telefono,
    this.direccion,
    this.fechaCreacion,
    this.fechaActualizacion,
    this.activo = true,
  });
```

```
Usuario copyWith({
  int? id,
  String? nombre,
  String? email,
  String? password,
  String? rol,
  String? telefono,
  String? direccion,
  DateTime? fechaCreacion,
  DateTime? fechaActualizacion,
  bool? activo,
}) {
  return Usuario(
    id: id ?? this.id,
    nombre: nombre ?? this.nombre,
    email: email ?? this.email,
    password: password ?? this.password,
    rol: rol ?? this.rol,
    telefono: telefono ?? this.telefono,
    direccion: direccion ?? this.direccion,
    fechaCreacion: fechaCreacion ?? this.fechaCreacion,
    fechaActualizacion: fechaActualizacion ?? this.fechaActualizacion,
    activo: activo ?? this.activo,
  );
}

Map<String, dynamic> toMap() {
  return {
    'id': id,
    'nombre': nombre,
    'email': email,
    'password': password,
    'rol': rol,
    'telefono': telefono,
    'direccion': direccion,
    'fecha_creacion': (fechaCreacion ?? DateTime.now()).toIso8601String(),
    'fecha_actualizacion': fechaActualizacion?.toIso8601String(),
    'activo': activo, // Usar boolean directo para Supabase
  };
}

factory Usuario.fromMap(Map<String, dynamic> map) {
  bool activoValue = true;
  if (map['activo'] != null) {
    if (map['activo'] is bool) {
      activoValue = map['activo'] as bool;
    } else if (map['activo'] is int) {
      activoValue = map['activo'] == 1;
    } else if (map['activo'] is String) {
      activoValue = map['activo'] == 'true' || map['activo'] == '1';
    }
  }
```

```
    return Usuario(
      id: map['id'],
      nombre: map['nombre'] ?? '',
      email: map['email'] ?? '',
      password: map['password'] ?? '',
      rol: map['rol'] ?? 'usuario',
      telefono: map['telefono'],
      direccion: map['direccion'],
      fechaCreacion: map['fecha_creacion'] != null
          ? DateTime.tryParse(map['fecha_creacion'].toString())
          : null,
      fechaActualizacion: map['fecha_actualizacion'] != null
          ? DateTime.tryParse(map['fecha_actualizacion'].toString())
          : null,
      activo: activoValue,
    );
  }
}
```

10.9 **Venta.dart**

```
import 'detalle_venta.dart';

class Venta {
  final int? id;
  final String numeroVenta;
  final DateTime fecha;
  final int? clienteId;
  final int? vendedorId;
  final double subtotal;
  final double descuento;
  final double impuesto;
  final double total;
  final String metodoPago;
  final String estado;
  final String? observaciones;

  String? nombreCliente;
  String? nombreVendedor;

  List<DetalleVenta> detalles;

  Venta({
    this.id,
    required this.numeroVenta,
    required this.fecha,
    this.clienteId,
    this.vendedorId, // ahora es opcional
    required this.subtotal,
    this.descuento = 0,
    this.impuesto = 0,
    required this.total,
    required this.metodoPago,
    this.estado = 'completada',
```

```
    this.observaciones,
    this.nombreCliente,
    this.nombreVendedor,
    List<DetalleVenta>? detalles,
  }) : detalles = detalles ?? [];

  Map<String, dynamic> toMap() {
    return {
      'id': id,
      'numero_venta': numeroVenta,
      'fecha': fecha.toIso8601String(),
      'cliente_id': clienteId,
      'vendedor_id': vendedorId,
      'subtotal': subtotal,
      'descuento': descuento,
      'impuesto': impuesto,
      'total': total,
      'metodo_pago': metodoPago,
      'estado': estado,
      'observaciones': observaciones,
    };
  }

  factory Venta.fromMap(Map<String, dynamic> map) {
    return Venta(
      id: map['id'],
      numeroVenta: map['numero_venta'] ?? '',
      fecha: map['fecha'] != null ? DateTime.parse(map['fecha']) :
DateTime.now(),
      clienteId: map['cliente_id'],
      vendedorId: map['vendedor_id'],
      subtotal: (map['subtotal'] as num?)?.toDouble() ?? 0.0,
      descuento: (map['descuento'] as num?)?.toDouble() ?? 0.0,
      impuesto: (map['impuesto'] as num?)?.toDouble() ?? 0.0,
      total: (map['total'] as num?)?.toDouble() ?? 0.0,
      metodoPago: map['metodo_pago'] ?? '',
      estado: map['estado'] ?? 'completada',
      observaciones: map['observaciones'],
      nombreCliente: map['nombre_cliente'],
      nombreVendedor: map['nombre_vendedor'],
      detalles: [],
    );
  }
}
```

11 UI/UX Y COMPONENTES

11.1 App Theme Completo

11.1.1 Tiene el tema oscuro y claro con colores personalizados al estilo de Azteca Fest.

```
import 'package:aztecafest1/core/constants.dart';
import 'package:flutter/material.dart';

class AppTheme {
  // ======================================
  // TEMA OSCURO (Estilo Admin Dashboard)
  // ======================================
  static ThemeData get temaOscuro {
    return ThemeData(
      brightness: Brightness.dark,
      primaryColor: AppConstants.colorPrimario,
      scaffoldBackgroundColor: AppConstants.fondoOscuro,
      colorScheme: const ColorScheme.dark(
        primary: AppConstants.colorPrimario,
        secondary: AppConstants.colorSecundario,
        error: AppConstants.colorError,
        surface: AppConstants.fondoTarjeta,
        background: AppConstants.fondoOscuro,
      ),

      // AppBar
      appBarTheme: const AppBarTheme(
        backgroundColor: AppConstants.fondoMedio,
        foregroundColor: AppConstants.textoBlanco,
        elevation: 0,
        centerTitle: true,
        titleTextStyle: TextStyle(
          fontSize: AppConstants.fuenteGrande,
          fontWeight: FontWeight.bold,
          color: AppConstants.textoBlanco,
        ),
      ),

      // Tarjetas
      cardTheme: CardThemeData(
        color: AppConstants.fondoTarjeta,
        elevation: 2,
        margin: const EdgeInsets.symmetric(
          horizontal: AppConstants.espaciadoMedio,
          vertical: AppConstants.espaciadoPequeno,
        ),
        shape: RoundedRectangleBorder(
          borderRadius:
BorderRadius.circular(AppConstants.bordeRedondeadoMedio),
        ),
      ),
```

```
      // Botones elevados
      elevatedButtonTheme: ElevatedButtonThemeData(
        style: ElevatedButton.styleFrom(
          backgroundColor: AppConstants.colorPrimario,
          foregroundColor: Colors.white,
          padding: const EdgeInsets.symmetric(
            horizontal: AppConstants.espaciadoGrande,
            vertical: AppConstants.espaciadoMedio,
          ),
          shape: RoundedRectangleBorder(
            borderRadius:
BorderRadius.circular(AppConstants.bordeRedondeadoMedio),
          ),
          elevation: 2,
        ),
      ),

      // Campos de texto
      inputDecorationTheme: InputDecorationTheme(
        filled: true,
        fillColor: AppConstants.fondoInput,
        border: OutlineInputBorder(
          borderRadius:
BorderRadius.circular(AppConstants.bordeRedondeadoMedio),
          borderSide: const BorderSide(color: AppConstants.textoGrisOscuro),
        ),
        enabledBorder: OutlineInputBorder(
          borderRadius:
BorderRadius.circular(AppConstants.bordeRedondeadoMedio),
          borderSide: const BorderSide(color: AppConstants.textoGrisOscuro),
        ),
        focusedBorder: OutlineInputBorder(
          borderRadius:
BorderRadius.circular(AppConstants.bordeRedondeadoMedio),
          borderSide: const BorderSide(color: AppConstants.colorPrimario,
width: 2),
        ),
        errorBorder: OutlineInputBorder(
          borderRadius:
BorderRadius.circular(AppConstants.bordeRedondeadoMedio),
          borderSide: const BorderSide(color: AppConstants.colorError),
        ),
        contentPadding: const EdgeInsets.symmetric(
          horizontal: AppConstants.espaciadoMedio,
          vertical: AppConstants.espaciadoMedio,
        ),
        hintStyle: const TextStyle(color: AppConstants.textoGrisMedio),
        labelStyle: const TextStyle(color: AppConstants.textoGrisClaro),
      ),

      // Texto
      textTheme: const TextTheme(
        displayLarge: TextStyle(
          fontSize: AppConstants.fuenteTitulo,
```

```
        fontWeight: FontWeight.bold,
        color: AppConstants.textoBlanco,
      ),
      titleLarge: TextStyle(
        fontSize: AppConstants.fuenteGrande,
        fontWeight: FontWeight.w600,
        color: AppConstants.textoBlanco,
      ),
      bodyLarge: TextStyle(
        fontSize: AppConstants.fuenteNormal,
        color: AppConstants.textoGrisClaro,
      ),
      bodyMedium: TextStyle(
        fontSize: AppConstants.fuenteNormal,
        color: AppConstants.textoGrisMedio,
      ),
    ),
  );
}

// ======================================
// TEMA CLARO (Para compatibilidad)
// ======================================
static ThemeData get temaClaro {
  return ThemeData(
    primaryColor: AppConstants.colorPrimario,
    scaffoldBackgroundColor: AppConstants.fondoClaro,
    colorScheme: ColorScheme.light(
      primary: AppConstants.colorPrimario,
      secondary: AppConstants.colorSecundario,
      error: AppConstants.colorError,
    ),

    // AppBar
    appBarTheme: const AppBarTheme(
      backgroundColor: AppConstants.colorPrimario,
      foregroundColor: Colors.white,
      elevation: 2,
      centerTitle: true,
      titleTextStyle: TextStyle(
        fontSize: AppConstants.fuenteGrande,
        fontWeight: FontWeight.bold,
        color: Colors.white,
      ),
    ),

    // Tarjetas
    cardTheme: CardThemeData(
      color: AppConstants.fondoTarjeta,
      elevation: 2,
      margin: const EdgeInsets.symmetric(
        horizontal: AppConstants.espaciadoMedio,
        vertical: AppConstants.espaciadoPequeno,
      ),
```

```
        shape: RoundedRectangleBorder(
          borderRadius:
BorderRadius.circular(AppConstants.bordeRedondeadoMedio),
        ),
      ),

      // Botones elevados
      elevatedButtonTheme: ElevatedButtonThemeData(
        style: ElevatedButton.styleFrom(
          backgroundColor: AppConstants.colorPrimario,
          foregroundColor: Colors.white,
          padding: const EdgeInsets.symmetric(
            horizontal: AppConstants.espaciadoGrande,
            vertical: AppConstants.espaciadoMedio,
          ),
          shape: RoundedRectangleBorder(
            borderRadius:
BorderRadius.circular(AppConstants.bordeRedondeadoMedio),
          ),
          elevation: 2,
        ),
      ),

      // Campos de texto
      inputDecorationTheme: InputDecorationTheme(
        filled: true,
        fillColor: Colors.white,
        border: OutlineInputBorder(
          borderRadius:
BorderRadius.circular(AppConstants.bordeRedondeadoMedio),
          borderSide: const BorderSide(color: AppConstants.textoClaro),
        ),
        enabledBorder: OutlineInputBorder(
          borderRadius:
BorderRadius.circular(AppConstants.bordeRedondeadoMedio),
          borderSide: const BorderSide(color: AppConstants.textoClaro),
        ),
        focusedBorder: OutlineInputBorder(
          borderRadius:
BorderRadius.circular(AppConstants.bordeRedondeadoMedio),
          borderSide: const BorderSide(color: AppConstants.colorPrimario,
width: 2),
        ),
        errorBorder: OutlineInputBorder(
          borderRadius:
BorderRadius.circular(AppConstants.bordeRedondeadoMedio),
          borderSide: const BorderSide(color: AppConstants.colorError),
        ),
        contentPadding: const EdgeInsets.symmetric(
          horizontal: AppConstants.espaciadoMedio,
          vertical: AppConstants.espaciadoMedio,
        ),
      ),
```

```
      // Texto
      textTheme: const TextTheme(
        displayLarge: TextStyle(
          fontSize: AppConstants.fuenteTitulo,
          fontWeight: FontWeight.bold,
          color: AppConstants.textoOscuro,
        ),
        titleLarge: TextStyle(
          fontSize: AppConstants.fuenteGrande,
          fontWeight: FontWeight.w600,
          color: AppConstants.textoOscuro,
        ),
        bodyLarge: TextStyle(
          fontSize: AppConstants.fuenteNormal,
          color: AppConstants.textoOscuro,
        ),
        bodyMedium: TextStyle(
          fontSize: AppConstants.fuenteNormal,
          color: AppConstants.textoMedio,
        ),
      ),
    );
  }
}
```

11.2 **Widgets Personalizados**

Componentes reutilizables como ProductoCard (tarjeta de producto con imagen, nombre, precio, botón de agregar al carrito).

```
// ui/widgets/producto_card.dart
import 'dart:io';
import 'package:flutter/material.dart';
import '../../data/models/producto.dart';
import
'package:font_awesome_flutter/font_awesome_flutter.dart';

class ProductoCard extends StatelessWidget {
  final Producto producto;
  final VoidCallback? onTap;
  final VoidCallback? onAddToCart;
  final bool isClientView; // true si es vista de cliente

  const ProductoCard({
    Key? key,
```

```
    required this.producto,
    this.onTap,
    this.onAddToCart,
    this.isClientView = false,
  }) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Card(
      elevation: 2,
      shape: RoundedRectangleBorder(
        borderRadius: BorderRadius.circular(12),
      ),
      child: InkWell(
        onTap: onTap,
        borderRadius: BorderRadius.circular(12),
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            // Imagen del producto
            _buildImagenProducto(),

            // Información del producto
            Padding(
              padding: const EdgeInsets.all(12),
              child: Column(
                crossAxisAlignment:
CrossAxisAlignment.start,
                children: [
                  // Nombre del producto
                  Text(
                    producto.nombre,
```

```
                    style: const TextStyle(
                      fontSize: 16,
                      fontWeight: FontWeight.bold,
                    ),
                    maxLines: 2,
                    overflow: TextOverflow.ellipsis,
                  ),
                  const SizedBox(height: 4),

                  // Categoría
                  Container(
                    padding: const EdgeInsets.symmetric(
                      horizontal: 8,
                      vertical: 4,
                    ),
                    decoration: BoxDecoration(
                      color:
_getCategoriaColor(producto.categoria).withValues(alpha:
0.2),
                      borderRadius:
BorderRadius.circular(6),
                    ),
                    child: Text(
                      producto.categoria,
                      style: TextStyle(
                        fontSize: 11,
                        fontWeight: FontWeight.w600,
                        color:
_getCategoriaColor(producto.categoria),
                      ),
                    ),
                  ),
                  const SizedBox(height: 8),
```

```
                  // Descripción (si existe y no es vista de
cliente en grid)
                  if (producto.descripcion != null &&
                      producto.descripcion!.isNotEmpty &&
                      !isClientView)
                    Padding(
                      padding: const EdgeInsets.only(bottom:
8),
                      child: Text(
                        producto.descripcion!,
                        style: TextStyle(
                          fontSize: 12,
                          color: Colors.grey.shade600,
                        ),
                        maxLines: 2,
                        overflow: TextOverflow.ellipsis,
                      ),
                    ),

                  // Precio
                  Row(
                    mainAxisAlignment:
MainAxisAlignment.spaceBetween,
                    children: [
                      Text(

'\$${producto.precioVenta.toStringAsFixed(2)}',
                        style: const TextStyle(
                          fontSize: 20,
                          fontWeight: FontWeight.bold,
                          color: Colors.green,
                        ),
```

```
                        ),

                        // Stock badge
                        if (!isClientView)
                          Container(
                            padding: const
EdgeInsets.symmetric(
                              horizontal: 8,
                              vertical: 4,
                            ),
                            decoration: BoxDecoration(
                              color: producto.bajoEnStock
                                  ?
 Colors.red.withValues(alpha: 0.1)
                                  :
Colors.blue.withValues(alpha: 0.1),
                              borderRadius:
BorderRadius.circular(6),
                            ),
                            child: Row(
                              mainAxisSize: MainAxisSize.min,
                              children: [
                                Icon(
                                  producto.bajoEnStock
                                      ? Icons.warning_amber
                                      : Icons.inventory,
                                  size: 14,
                                  color: producto.bajoEnStock
                                      ? Colors.red
                                      : Colors.blue,
                                ),
                                const SizedBox(width: 4),
                                Text(
```

```
                                '${producto.stock}',
                                style: TextStyle(
                                  fontSize: 12,
                                  fontWeight:
FontWeight.bold,
                                  color:
producto.bajoEnStock
                                      ? Colors.red
                                      : Colors.blue,
                                ),
                              ),
                            ],
                          ),
                        ),
                      ],
                    ),

                    // Botón agregar al carrito (solo vista
cliente)
                    if (isClientView && onAddToCart != null)
...[
                      const SizedBox(height: 8),
                      SizedBox(
                        width: double.infinity,
                        child: ElevatedButton.icon(
                          onPressed: producto.stock > 0 ?
onAddToCart : null,
                          icon: const
Icon(Icons.add_shopping_cart, size: 18),
                          label: Text(
                            producto.stock > 0
                                ? 'Agregar al Carrito'
                                : 'Sin Stock',
```

```
                        ),
                        style: ElevatedButton.styleFrom(
                          padding: const
EdgeInsets.symmetric(vertical: 8),
                        ),
                      ),
                    ),
                  ],
                ],
              ),
            ),
          ],
        ),
      ),
    );
  }

  Widget _buildImagenProducto() {
    return Container(
      height: 180,
      width: double.infinity,
      decoration: BoxDecoration(
        color: Colors.grey.shade100,
        borderRadius: const BorderRadius.only(
          topLeft: Radius.circular(12),
          topRight: Radius.circular(12),
        ),
      ),
      child: ClipRRect(
        borderRadius: const BorderRadius.only(
          topLeft: Radius.circular(12),
          topRight: Radius.circular(12),
```

```
        ),
        child: producto.tieneImagen
            ? Stack(
                fit: StackFit.expand,
                children: [
                  _buildImage(),
                  // Overlay con categoría
                  Positioned(
                    top: 8,
                    right: 8,
                    child: Container(
                      padding: const EdgeInsets.symmetric(
                        horizontal: 8,
                        vertical: 4,
                      ),
                        decoration: BoxDecoration(
                          color:
Colors.black.withValues(alpha: 0.6),
                          borderRadius:
BorderRadius.circular(6),
                        ),
                      child: Text(
                        producto.categoria,
                        style: const TextStyle(
                          color: Colors.white,
                          fontSize: 11,
                          fontWeight: FontWeight.bold,
                        ),
                      ),
                    ),
                  ),
                  // Badge de stock bajo
```

```
                    if (producto.bajoEnStock && !isClientView)
                      Positioned(
                        top: 8,
                        left: 8,
                        child: Container(
                          padding: const EdgeInsets.symmetric(
                            horizontal: 8,
                            vertical: 4,
                          ),
                          decoration: BoxDecoration(
                            color: Colors.red,
                            borderRadius:
BorderRadius.circular(6),
                          ),
                          child: const Row(
                            mainAxisSize: MainAxisSize.min,
                            children: [
                              Icon(
                                Icons.warning_amber,
                                color: Colors.white,
                                size: 14,
                              ),
                              SizedBox(width: 4),
                              Text(
                                'Stock Bajo',
                                style: TextStyle(
                                  color: Colors.white,
                                  fontSize: 11,
                                  fontWeight: FontWeight.bold,
                                ),
                              ),
                            ],
```

```
                        ),
                      ),
                    ),
                  ],
                )
              : _buildPlaceholderImagen(),
      ),
    );
  }

  Widget _buildImage() {
    if (producto.imagenUrl != null &&
producto.imagenUrl!.isNotEmpty) {
      // Si es una URL de red (Supabase)
      if (producto.imagenUrl!.startsWith('http')) {
        return Image.network(
          producto.imagenUrl!,
          fit: BoxFit.cover,
          loadingBuilder: (context, child, loadingProgress)
{
            if (loadingProgress == null) return child;
            return Center(
              child: CircularProgressIndicator(
                value: loadingProgress.expectedTotalBytes !=
null
                    ? loadingProgress.cumulativeBytesLoaded
/
                        loadingProgress.expectedTotalBytes!
                    : null,
                strokeWidth: 2,
              ),
            );
          },
```

```
        errorBuilder: (context, error, stackTrace) {
          return _buildPlaceholderImagen();
        },
      );
    } else {
      // Es un archivo local
      return Image.file(
        File(producto.imagenUrl!),
        fit: BoxFit.cover,
        errorBuilder: (context, error, stackTrace) {
          return _buildPlaceholderImagen();
        },
      );
    }
  }
  return _buildPlaceholderImagen();
}

Widget _buildPlaceholderImagen() {
  return Center(
    child: Column(
      mainAxisAlignment: MainAxisAlignment.center,
      children: [
        Icon(
          _getCategoriaIcon(producto.categoria),
          size: 64,
          color: Colors.grey.shade400,
        ),
        const SizedBox(height: 8),
        Text(
          producto.categoria,
          style: TextStyle(
```

```
            color: Colors.grey.shade600,
            fontSize: 14,
            fontWeight: FontWeight.w600,
          ),
        ),
      ],
    ),
  );
}

Color _getCategoriaColor(String categoria) {
  switch (categoria) {
    case 'Bebidas':
      return Colors.blue;
    case 'Comida':
      return Colors.orange;
    case 'Snacks':
      return Colors.purple;
    case 'Postres':
      return Colors.pink;
    default:
      return Colors.grey;
  }
}

IconData _getCategoriaIcon(String categoria) {
  switch (categoria) {
    case 'Bebidas':
      return Icons.local_drink;
    case 'Comida':
      return Icons.restaurant;
    case 'Snacks':
```

```
        return Icons.fastfood;
      case 'Postres':
        return Icons.cake;
      case 'Salchipapas':
        return FontAwesomeIcons.hotdog;
      case 'Sándwiches':
      case 'Sandwiches':
        return Icons.lunch_dining;
      default:
        return Icons.inventory_2;
    }
  }
}
```

Animated Add Button.dart

```
import 'package:flutter/material.dart';

class AnimatedAddButton extends StatefulWidget {
  final VoidCallback onPressed;
  final bool isLoading;
  final String text;

  // ignore: use_super_parameters
  const AnimatedAddButton({
    Key? key,
    required this.onPressed,
    this.isLoading = false,
    this.text = "Agregar",
  }) : super(key: key);

  @override
  State<AnimatedAddButton> createState() => _AnimatedAddButtonState();
}

class _AnimatedAddButtonState extends State<AnimatedAddButton>
    with SingleTickerProviderStateMixin {
  late AnimationController _controller;
  late Animation<double> _scaleAnimation;
  late Animation<double> _rotationAnimation;

  @override
  void initState() {
    super.initState();
    _controller = AnimationController(
```

```
      duration: const Duration(milliseconds: 300),
      vsync: this,
    );
    _scaleAnimation = Tween<double>(begin: 1.0, end: 0.95).animate(
      CurvedAnimation(parent: _controller, curve: Curves.easeInOut),
    );
    _rotationAnimation = Tween<double>(begin: 0.0, end: 0.1).animate(
      CurvedAnimation(parent: _controller, curve: Curves.easeInOut),
    );
  }

  @override
  void dispose() {
    _controller.dispose();
    super.dispose();
  }

  void _onTapDown(TapDownDetails details) {
    _controller.forward();
  }

  void _onTapUp(TapUpDetails details) {
    _controller.reverse();
  }

  void _onTapCancel() {
    _controller.reverse();
  }

  @override
  Widget build(BuildContext context) {
    return GestureDetector(
      onTapDown: _onTapDown,
      onTapUp: _onTapUp,
      onTapCancel: _onTapCancel,
      child: ScaleTransition(
        scale: _scaleAnimation,
        child: RotationTransition(
          turns: _rotationAnimation,
          child: ElevatedButton(
            onPressed: widget.isLoading ? null : widget.onPressed,
            style: ElevatedButton.styleFrom(
              backgroundColor: const Color(0xFFE74C3C), // Rojo cálido
mexicano
              foregroundColor: Colors.white,
              padding: const EdgeInsets.symmetric(horizontal: 16, vertical:
12),
              shape: RoundedRectangleBorder(
                borderRadius: BorderRadius.circular(8),
              ),
              elevation: 4,
              shadowColor: const Color(0xFFE74C3C).withOpacity(0.4),
            ),
            child: widget.isLoading
```

```
                ? const SizedBox(
                    width: 20,
                    height: 20,
                    child: CircularProgressIndicator(
                      strokeWidth: 2,
                      valueColor:
AlwaysStoppedAnimation<Color>(Colors.white),
                    ),
                  )
                : Row(
                    mainAxisSize: MainAxisSize.min,
                    children: [
                      const Icon(
                        Icons.add_shopping_cart,
                        size: 18,
                        color: Colors.white,
                      ),
                      const SizedBox(width: 8),
                      Text(
                        widget.text,
                        style: const TextStyle(
                          fontSize: 14,
                          fontWeight: FontWeight.bold,
                        ),
                      ),
                    ],
                  ),
          ),
        ),
      ),
    );
  }
}
```

CartBadge.dart

```
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import '../../services/carrito_service.dart';

class CartBadge extends StatefulWidget {
  final VoidCallback onTap;

  const CartBadge({Key? key, required this.onTap}) : super(key: key);

  @override
  State<CartBadge> createState() => _CartBadgeState();
}

class _CartBadgeState extends State<CartBadge> with
SingleTickerProviderStateMixin {
  late AnimationController _animationController;
  late Animation<double> _animation;

  @override
```

```
  void initState() {
    super.initState();
    _animationController = AnimationController(
      duration: const Duration(milliseconds: 300),
      vsync: this,
    );
    _animation = Tween<double>(begin: 1.0, end: 1.3).animate(
      CurvedAnimation(parent: _animationController, curve:
Curves.elasticOut),
    );
  }

  @override
  void dispose() {
    _animationController.dispose();
    super.dispose();
  }

  @override
  Widget build(BuildContext context) {
    return Consumer<CarritoService>(
      builder: (context, carrito, child) {
        // Solo animar si hay items en el carrito
        if (carrito.cantidadTotal > 0) {
          _animationController.forward().then((_) {
            _animationController.reverse();
          });
        }

        return Stack(
          clipBehavior: Clip.none,
          children: [
            IconButton(
              icon: const Icon(
                Icons.shopping_cart_outlined,
                color: Colors.white,
                size: 28,
              ),
              onPressed: widget.onTap,
              tooltip: 'Carrito',
            ),
            if (carrito.cantidadTotal > 0)
              Positioned(
                right: 0,
                top: 0,
                child: ScaleTransition(
                  scale: _animation,
                  child: Container(
                    padding: const EdgeInsets.all(6),
                    decoration: BoxDecoration(
                      color: const Color(0xFFE74C3C), // Rojo cálido mexicano
                      borderRadius: BorderRadius.circular(12),
                      border: Border.all(
                        color: Colors.white,
```

```
                        width: 2,
                      ),
                      boxShadow: [
                        BoxShadow(
                          color: const Color(0xFFE74C3C).withOpacity(0.5),
                          blurRadius: 8,
                          offset: const Offset(0, 2),
                        ),
                      ],
                    ),
                    constraints: const BoxConstraints(
                      minWidth: 24,
                      minHeight: 24,
                    ),
                    child: Text(
                      '${carrito.cantidadTotal}',
                      style: const TextStyle(
                        color: Colors.white,
                        fontSize: 12,
                        fontWeight: FontWeight.bold,
                      ),
                      textAlign: TextAlign.center,
                    ),
                  ),
                ),
              ),
            ),
          ],
        );
      },
    );
  }
}
```

CustomButton.dart

```
import 'package:flutter/material.dart';
import '../../core/constants.dart';

class CustomButton extends StatelessWidget {
  final String texto;
  final VoidCallback? onPressed;
  final bool cargando;
  final Color? color;
  final Color? colorTexto;
  final IconData? icono;
  final bool expandir;

  const CustomButton({
    Key? key,
    required this.texto,
    this.onPressed,
    this.cargando = false,
    this.color,
    this.colorTexto,
    this.icono,
```

```
    this.expandir = true,
  }) : super(key: key);

  @override
  Widget build(BuildContext context) {
    Widget boton = ElevatedButton(
      onPressed: cargando ? null : onPressed,
      style: ElevatedButton.styleFrom(
        backgroundColor: color ?? AppConstants.colorPrimario,
        foregroundColor: colorTexto ?? Colors.white,
        padding: const EdgeInsets.symmetric(
          horizontal: AppConstants.espaciadoGrande,
          vertical: AppConstants.espaciadoMedio,
        ),
      ),
      child: cargando
          ? const SizedBox(
              height: 20,
              width: 20,
              child: CircularProgressIndicator(
                strokeWidth: 2,
                valueColor: AlwaysStoppedAnimation<Color>(Colors.white),
              ),
            )
          : Row(
              mainAxisSize: MainAxisSize.min,
              children: [
                if (icono != null) ...[
                  Icon(icono, size: 20),
                  const SizedBox(width: AppConstants.espaciadoPequeno),
                ],
                Text(texto),
              ],
            ),
    );

    return expandir ? SizedBox(width: double.infinity, child: boton) : boton;
  }
}
```

CustomImput.dart

```
// widgets/custom_input.dart
import 'package:flutter/material.dart';
import '../../core/constants.dart';

class CustomInput extends StatelessWidget {
  final String label;
  final String? hint;
  final TextEditingController? controller;
  final String? Function(String?)? validator;
  final TextInputType? keyboardType;
  final bool obscureText;
  final IconData? prefixIcon;
  final IconData? suffixIcon;
```

```
final Widget? suffixIconWidget;
final VoidCallback? onSuffixIconPressed;
final int? maxLines;
final bool enabled;
final TextInputAction? textInputAction;
final Function(String)? onChanged;
final Function(String)? onSubmitted;

const CustomInput({
  Key? key,
  required this.label,
  this.hint,
  this.controller,
  this.validator,
  this.keyboardType,
  this.obscureText = false,
  this.prefixIcon,
  this.suffixIcon,
  this.suffixIconWidget,
  this.onSuffixIconPressed,
  this.maxLines = 1,
  this.enabled = true,
  this.textInputAction,
  this.onChanged,
  this.onSubmitted,
}) : super(key: key);

@override
Widget build(BuildContext context) {
  return Column(
    crossAxisAlignment: CrossAxisAlignment.start,
    children: [
      Text(
        label,
        style: const TextStyle(
          fontSize: AppConstants.fuenteMediana,
          fontWeight: FontWeight.w500,
        ),
      ),
      const SizedBox(height: 8),
      TextFormField(
        controller: controller,
        validator: validator,
        keyboardType: keyboardType,
        obscureText: obscureText,
        maxLines: obscureText ? 1 : maxLines,
        enabled: enabled,
        textInputAction: textInputAction,
        onChanged: onChanged,
        onFieldSubmitted: onSubmitted,
        decoration: InputDecoration(
          hintText: hint,
          hintStyle: TextStyle(
            color: AppConstants.textoMedio,
```

```
),
prefixIcon: prefixIcon != null
    ? Icon(prefixIcon, color: AppConstants.colorPrimario)
    : null,
suffixIcon: suffixIconWidget ??
    (suffixIcon != null
        ? IconButton(
            icon: Icon(suffixIcon),
            onPressed: onSuffixIconPressed,
          )
        : null),
filled: true,
fillColor: enabled ? Colors.grey[100] : Colors.grey[200],
border: OutlineInputBorder(
  borderRadius: BorderRadius.circular(
    AppConstants.bordeRedondeadoMedio,
  ),
  borderSide: BorderSide.none,
),
enabledBorder: OutlineInputBorder(
  borderRadius: BorderRadius.circular(
    AppConstants.bordeRedondeadoMedio,
  ),
  borderSide: BorderSide(
    color: Colors.grey[300]!,
    width: 1,
  ),
),
focusedBorder: OutlineInputBorder(
  borderRadius: BorderRadius.circular(
    AppConstants.bordeRedondeadoMedio,
  ),
  borderSide: BorderSide(
    color: AppConstants.colorPrimario,
    width: 2,
  ),
),
errorBorder: OutlineInputBorder(
  borderRadius: BorderRadius.circular(
    AppConstants.bordeRedondeadoMedio,
  ),
  borderSide: BorderSide(
    color: AppConstants.colorError,
    width: 1,
  ),
),
focusedErrorBorder: OutlineInputBorder(
  borderRadius: BorderRadius.circular(
    AppConstants.bordeRedondeadoMedio,
  ),
  borderSide: BorderSide(
    color: AppConstants.colorError,
    width: 2,
  ),
```

```
            ),
            contentPadding: const EdgeInsets.symmetric(
              horizontal: 16,
              vertical: 16,
            ),
          ),
        ),
      ],
    );
  }
}
```

ListItem.dart

```
import 'package:flutter/material.dart';
import '../../core/constants.dart';

class ListItem extends StatelessWidget {
  final String titulo;
  final String? subtitulo;
  final String? detalle;
  final IconData? icono;
  final Color? colorIcono;
  final VoidCallback? onTap;
  final Widget? trailing;

  const ListItem({
    Key? key,
    required this.titulo,
    this.subtitulo,
    this.detalle,
    this.icono,
    this.colorIcono,
    this.onTap,
    this.trailing,
  }) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Card(
      child: ListTile(
        leading: icono != null
            ? CircleAvatar(
                backgroundColor: colorIcono ?? AppConstants.colorPrimario,
                child: Icon(icono, color: Colors.white),
              )
            : null,
        title: Text(
          titulo,
          style: const TextStyle(
            fontWeight: FontWeight.w600,
            fontSize: AppConstants.fuenteNormal,
          ),
        ),
        subtitle: subtitulo != null || detalle != null
```

```
            ? Column(
                crossAxisAlignment: CrossAxisAlignment.start,
                children: [
                  if (subtitulo != null) ...[
                    const SizedBox(height: 4),
                    Text(subtitulo!),
                  ],
                  if (detalle != null) ...[
                    const SizedBox(height: 4),
                    Text(
                      detalle!,
                      style: const TextStyle(
                        color: AppConstants.textoMedio,
                        fontSize: AppConstants.fuentePequena,
                      ),
                    ),
                  ],
                ],
              )
            : null,
        trailing: trailing ?? const Icon(Icons.chevron_right),
        onTap: onTap,
      ),
    );
  }
}
```

ProductFeedbackOverlay.dart

```
import 'package:flutter/material.dart';

class ProductFeedbackOverlay extends StatefulWidget {
  final Widget child;

  const ProductFeedbackOverlay({
    Key? key,
    required this.child,
  }) : super(key: key);

  @override
  ProductFeedbackOverlayState createState() => ProductFeedbackOverlayState();
}

class ProductFeedbackOverlayState extends State<ProductFeedbackOverlay>
    with TickerProviderStateMixin {
  late AnimationController _controller;
  late Animation<double> _opacityAnimation;
  late Animation<Offset> _slideAnimation;
  String _feedbackText = '¡Producto agregado!';

  @override
  void initState() {
    super.initState();
    _controller = AnimationController(
      duration: const Duration(milliseconds: 1500),
```

```
      vsync: this,
    );
    _opacityAnimation = Tween<double>(begin: 0.0, end: 1.0).animate(
      CurvedAnimation(
        parent: _controller,
        curve: const Interval(0.0, 0.5, curve: Curves.easeOut),
      ),
    );
    _slideAnimation = Tween<Offset>(
      begin: const Offset(0, 1),
      end: Offset.zero,
    ).animate(
      CurvedAnimation(
        parent: _controller,
        curve: const Interval(0.0, 0.5, curve: Curves.elasticOut),
      ),
    );
  }

  @override
  void dispose() {
    _controller.dispose();
    super.dispose();
  }

  void showFeedback([String? text]) {
    if (text != null) {
      setState(() => _feedbackText = text);
    }
    _controller.forward(from: 0).then((_) {
      Future.delayed(const Duration(milliseconds: 800), () {
        if (mounted) {
          _controller.reverse();
        }
      });
    });
  }

  @override
  Widget build(BuildContext context) {
    // Obtener el ancho de la pantalla para hacer el mensaje responsivo
    final screenWidth = MediaQuery.of(context).size.width;
    final isMobile = screenWidth < 600;

    // Ajustar padding y tamaños según el dispositivo
    final horizontalPadding = isMobile ? 12.0 : 24.0;
    final verticalPadding = isMobile ? 12.0 : 16.0;
    final iconSize = isMobile ? 20.0 : 24.0;
    final fontSize = isMobile ? 14.0 : 16.0;
    final spacing = isMobile ? 8.0 : 12.0;

    // El contenedor ocupará el 90% del ancho en móviles, máximo 350px en
tablets/desktop
    final maxWidth = isMobile ? screenWidth * 0.9 : 350.0;
```

```
return Stack(
  children: [
    widget.child,
    Positioned.fill(
      child: IgnorePointer(
        child: FadeTransition(
          opacity: _opacityAnimation,
          child: SlideTransition(
            position: _slideAnimation,
            child: Align(
              alignment: Alignment.bottomCenter,
              child: Container(
                margin: EdgeInsets.all(isMobile ? 12 : 16),
                padding: EdgeInsets.symmetric(
                  horizontal: horizontalPadding,
                  vertical: verticalPadding,
                ),
                constraints: BoxConstraints(maxWidth: maxWidth),
                decoration: BoxDecoration(
                  color: const Color(0xFF27AE60),
                  borderRadius: BorderRadius.circular(12),
                  boxShadow: [
                    BoxShadow(
                      color: Colors.black.withOpacity(0.2),
                      blurRadius: 12,
                      offset: const Offset(0, 4),
                    ),
                  ],
                ),
                child: Row(
                  mainAxisSize: MainAxisSize.min,
                  children: [
                    Icon(
                      Icons.check_circle,
                      color: Colors.white,
                      size: iconSize,
                    ),
                    SizedBox(width: spacing),
                    Flexible(
                      child: Text(
                        _feedbackText,
                        style: TextStyle(
                          color: Colors.white,
                          fontSize: fontSize,
                          fontWeight: FontWeight.bold,
                        ),
                        overflow: TextOverflow.visible,
                        maxLines: 2,
                        textAlign: TextAlign.center,
                      ),
                    ),
                  ],
                ),
```

```
                ),
              ),
            ),
          ),
        ),
      ),
    ],
  );
 }
}
ResponsiveWidgets

import 'package:flutter/material.dart';
import '../../utils/responsive_util.dart';

/// 🎨 ResponsiveButton - Botón adaptativo para todas las plataformas
class ResponsiveButton extends StatelessWidget {
  final String label;
  final VoidCallback onPressed;
  final bool isLoading;
  final Color? backgroundColor;
  final Color? textColor;
  final IconData? icon;
  final bool isOutlined;
  final bool fullWidth;

  const ResponsiveButton({
    Key? key,
    required this.label,
    required this.onPressed,
    this.isLoading = false,
    this.backgroundColor,
    this.textColor,
    this.icon,
    this.isOutlined = false,
    this.fullWidth = true,
  }) : super(key: key);

  @override
  Widget build(BuildContext context) {
    final sizes = ResponsiveUtil.getSizes(context);
    final bgColor = backgroundColor ?? const Color(0xFFFF6B6B);
    final txtColor = textColor ?? Colors.white;

    final button = isOutlined
        ? OutlinedButton(
            onPressed: isLoading ? null : onPressed,
            style: OutlinedButton.styleFrom(
              side: BorderSide(color: bgColor),
              padding: EdgeInsets.symmetric(
                horizontal: sizes.padding,
                vertical: sizes.padding / 2,
              ),
              shape: RoundedRectangleBorder(
```

```
              borderRadius: BorderRadius.circular(sizes.borderRadius),
            ),
          ),
          child: _buildChild(sizes, bgColor, txtColor),
        )
      : ElevatedButton(
          onPressed: isLoading ? null : onPressed,
          style: ElevatedButton.styleFrom(
            backgroundColor: bgColor,
            disabledBackgroundColor: Colors.grey[600],
            padding: EdgeInsets.symmetric(
              horizontal: sizes.padding,
              vertical: sizes.padding / 2,
            ),
            shape: RoundedRectangleBorder(
              borderRadius: BorderRadius.circular(sizes.borderRadius),
            ),
          ),
          child: _buildChild(sizes, bgColor, txtColor),
        );

  if (fullWidth) {
    return SizedBox(
      width: double.infinity,
      height: sizes.buttonHeight,
      child: button,
    );
  }

  return button;
}

Widget _buildChild(ResponsiveSizes sizes, Color bgColor, Color txtColor) {
  if (isLoading) {
    return SizedBox(
      height: sizes.iconSizeSmall,
      width: sizes.iconSizeSmall,
      child: CircularProgressIndicator(
        strokeWidth: 2,
        valueColor: AlwaysStoppedAnimation<Color>(txtColor),
      ),
    );
  }

  if (icon != null) {
    return Row(
      mainAxisSize: MainAxisSize.min,
      children: [
        Icon(icon, color: txtColor, size: sizes.iconSize),
        SizedBox(width: sizes.spacingSmall),
        Text(
          label,
          style: TextStyle(
            color: txtColor,
```

```
              fontSize: sizes.fontSize,
              fontWeight: FontWeight.bold,
            ),
          ),
        ],
      );
    }

    return Text(
      label,
      style: TextStyle(
        color: txtColor,
        fontSize: sizes.fontSize,
        fontWeight: FontWeight.bold,
      ),
    );
  }
}

/// 📱 ResponsiveCard - Tarjeta adaptativa
class ResponsiveCard extends StatelessWidget {
  final Widget child;
  final double? width;
  final double? height;
  final EdgeInsets? padding;
  final Color? backgroundColor;
  final double? elevation;
  final VoidCallback? onTap;

  const ResponsiveCard({
    Key? key,
    required this.child,
    this.width,
    this.height,
    this.padding,
    this.backgroundColor,
    this.elevation,
    this.onTap,
  }) : super(key: key);

  @override
  Widget build(BuildContext context) {
    final sizes = ResponsiveUtil.getSizes(context);

    return GestureDetector(
      onTap: onTap,
      child: Card(
        color: backgroundColor ?? ResponsiveUtil.getCardColor(context),
        elevation: elevation ?? 4,
        shape: RoundedRectangleBorder(
          borderRadius: BorderRadius.circular(sizes.borderRadius),
        ),
        child: Container(
          width: width,
```

```
          height: height,
          padding: padding ?? EdgeInsets.all(sizes.padding),
          child: child,
        ),
      ),
    );
  }
}

/// 🔤 ResponsiveText - Texto adaptativo
class ResponsiveText extends StatelessWidget {
  final String text;
  final double? fontSize;
  final FontWeight? fontWeight;
  final Color? color;
  final TextAlign? textAlign;
  final int? maxLines;
  final TextOverflow? overflow;
  final bool isTitle;
  final bool isHeading;
  final bool isSubheading;

  const ResponsiveText(
    this.text, {
    Key? key,
    this.fontSize,
    this.fontWeight,
    this.color,
    this.textAlign,
    this.maxLines,
    this.overflow,
    this.isTitle = false,
    this.isHeading = false,
    this.isSubheading = false,
  }) : super(key: key);

  @override
  Widget build(BuildContext context) {
    final sizes = ResponsiveUtil.getSizes(context);

    double finalFontSize = fontSize ?? sizes.fontSize;
    if (isTitle) finalFontSize = sizes.fontSizeTitle;
    if (isHeading) finalFontSize = sizes.fontSizeHeading;
    if (isSubheading) finalFontSize = sizes.fontSizeSubheading;

    return Text(
      text,
      style: TextStyle(
        fontSize: finalFontSize,
        fontWeight: fontWeight ?? FontWeight.normal,
        color: color ?? ResponsiveUtil.getTextColor(context),
      ),
      textAlign: textAlign,
      maxLines: maxLines,
```

```
      overflow: overflow,
    );
  }
}

/// 📏 ResponiveContainer - Contenedor adaptativo
class ResponsiveContainer extends StatelessWidget {
  final Widget child;
  final double? maxWidth;
  final EdgeInsets? padding;
  final Alignment alignment;

  const ResponsiveContainer({
    Key? key,
    required this.child,
    this.maxWidth,
    this.padding,
    this.alignment = Alignment.center,
  }) : super(key: key);

  @override
  Widget build(BuildContext context) {
    final sizes = ResponsiveUtil.getSizes(context);

    return Center(
      child: ConstrainedBox(
        constraints: BoxConstraints(
          maxWidth: maxWidth ?? sizes.containerMaxWidth,
        ),
        child: Padding(
          padding: padding ?? EdgeInsets.all(sizes.padding),
          child: child,
        ),
      ),
    );
  }
}
```

ScrollableList.dart

```
import 'package:flutter/material.dart';
import '../../core/constants.dart';

class ScrollableList<T> extends StatelessWidget {
  final List<T> items;
  final Widget Function(BuildContext, T) itemBuilder;
  final bool cargando;
  final String? mensajeVacio;
  final Future<void> Function()? onRefresh;

  const ScrollableList({
    Key? key,
    required this.items,
    required this.itemBuilder,
    this.cargando = false,
```

```
    this.mensajeVacio,
    this.onRefresh,
  }) : super(key: key);

  @override
  Widget build(BuildContext context) {
    if (cargando) {
      return const Center(child: CircularProgressIndicator());
    }

    if (items.isEmpty) {
      return Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            Icon(
              Icons.inbox_outlined,
              size: 64,
              color: AppConstants.textoClaro,
            ),
            const SizedBox(height: AppConstants.espaciadoMedio),
            Text(
              mensajeVacio ?? AppConstants.mensajeSinDatos,
              style: const TextStyle(
                color: AppConstants.textoMedio,
                fontSize: AppConstants.fuenteNormal,
              ),
            ),
          ],
        ),
      );
    }

    Widget lista = ListView.builder(
      itemCount: items.length,
      padding: const EdgeInsets.all(AppConstants.espaciadoPequeno),
      itemBuilder: (context, index) => itemBuilder(context, items[index]),
    );

    if (onRefresh != null) {
      return RefreshIndicator(
        onRefresh: onRefresh!,
        child: lista,
      );
    }

    return lista;
  }
}
```

11.3 **Pantallas por Rol**

Admin:

Admindashboard.dart

```
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import '../../core/app_routes.dart';
import '../../services/auth_service.dart';
import '../../data/models/usuario.dart';
import 'admin_productos_tab.dart';
import 'admin_ventas_tab.dart';
import 'admin_pedidos_tab.dart';
import 'admin_usuarios_tab.dart';
import 'admin_reportes_tab.dart';

/// 🎯 PANEL DE ADMINISTRACIÓN
/// Solo para usuarios con rol ADMIN
/// Gestiona: Productos, Ventas, Pedidos, Usuarios y Reportes
class AdminDashboard extends StatefulWidget {
  final Usuario? usuario;

  const AdminDashboard({
    super.key,
    this.usuario,
  });

  @override
  State<AdminDashboard> createState() => _AdminDashboardState();
}

class _AdminDashboardState extends State<AdminDashboard> {
  int _indiceSeleccionado = 0;

  // 📋 Tabs del panel de administración
  final List<Widget> _tabs = const [
    AdminProductosTab(),
    AdminVentasTab(),
    AdminPedidosTab(),
    AdminUsuariosTab(),
    AdminReportesTab(),
  ];

  Map<String, dynamic> _getResponsiveSizes(double width) {
    if (width > 800) {
      return {
        'titleSize': 22.0,
        'iconSize': 26.0,
        'fontSize': 16.0,
        'avatarSize': 44.0,
        'padding': 16.0,
      };
    } else if (width > 600) {
      return {
        'titleSize': 20.0,
        'iconSize': 24.0,
```

```
      'fontSize': 15.0,
      'avatarSize': 40.0,
      'padding': 14.0,
    };
  } else {
    return {
      'titleSize': 18.0,
      'iconSize': 22.0,
      'fontSize': 14.0,
      'avatarSize': 36.0,
      'padding': 12.0,
    };
  }
}

@override
Widget build(BuildContext context) {
  final authService = context.watch<AuthService>();
  final usuario = widget.usuario ?? authService.usuarioActual;

  final width = MediaQuery.of(context).size.width;
  final sizes = _getResponsiveSizes(width);
  final isDesktop = width > 800;

  return Scaffold(
    backgroundColor: const Color(0xFF0A0A0F),
    appBar: _buildAppBar(usuario, isDesktop, sizes),
    body: _tabs[_indiceSeleccionado],
    bottomNavigationBar: _buildBottomNavBar(sizes),
  );
}

// ========================================
// APP BAR
// ========================================
PreferredSizeWidget _buildAppBar(
  Usuario? usuario,
  bool  isDesktop,
  Map<String, dynamic> sizes,
) {
  return AppBar(
    backgroundColor: const Color(0xFF0A0A0F),
    elevation: 0,
    automaticallyImplyLeading: false,
    toolbarHeight: 70,
    title: Row(
      children: [
        // Logo
        Container(
          width: sizes['avatarSize'],
          height: sizes['avatarSize'],
          decoration: BoxDecoration(
            gradient: const LinearGradient(
              colors: [Color(0xFFFF073A), Color(0xFFFF8E53)],
```

```
        ),
        borderRadius: BorderRadius.circular(12),
        boxShadow: [
          BoxShadow(
            color: Theme.of(context)
                .colorScheme
                .primary
                .withValues(alpha: 0.3),
            blurRadius: 8,
            offset: const Offset(0, 2),
          ),
        ],
      ),
      child: ClipRRect(
        borderRadius: BorderRadius.circular(12),
        child: Image.asset(
          'lib/assets/images/favicon.ico',
          fit: BoxFit.contain,
          errorBuilder: (context, error, stackTrace) {
            return const Icon(
              Icons.festival_rounded,
              color: Colors.white,
              size: 24,
            );
          },
        ),
      ),
    ),
    const SizedBox(width: 12),

    // Título
    if (isDesktop)
      ShaderMask(
        shaderCallback: (bounds) => const LinearGradient(
          colors: [Color(0xFFFF073A), Color(0xFFFF8E53)],
        ).createShader(bounds),
        child: Text(
          'Azteca Fest - Panel Admin',
          style: TextStyle(
            fontSize: sizes['titleSize'],
            fontWeight: FontWeight.bold,
            color: Colors.white,
            letterSpacing: 0.5,
          ),
        ),
      ),
  ],
),
actions: [
  // Badge de ADMIN
  Container(
    margin: const EdgeInsets.only(right: 12),
    padding: const EdgeInsets.symmetric(horizontal: 12, vertical: 6),
    decoration: BoxDecoration(
```

```
          gradient: const LinearGradient(
            colors: [Color(0xFFFF073A), Color(0xFFFF8E53)],
          ),
          borderRadius: BorderRadius.circular(20),
        ),
        child: const Text(
          'ADMIN',
          style: TextStyle(
            color: Colors.white,
            fontSize: 12,
            fontWeight: FontWeight.bold,
          ),
        ),
      ),

      // Menú de perfil
      _buildMenuPerfil(usuario, sizes),

      SizedBox(width: sizes['padding']),
    ],
  );
}

// ========================================
// MENÚ DE PERFIL
// ========================================
Widget _buildMenuPerfil(Usuario? usuario, Map<String, dynamic> sizes) {
  final authService = context.read<AuthService>();

  return Container(
    margin: const EdgeInsets.only(right: 12),
    decoration: BoxDecoration(
      color: const Color(0xFF0A0A0F),
      borderRadius: BorderRadius.circular(12),
      border: Border.all(
        color: const Color(0xFFFF073A).withValues(alpha: 0.3),
      ),
    ),
    child: PopupMenuButton<String>(
      icon: Icon(
        Icons.account_circle_rounded,
        color: Colors.white,
        size: sizes['iconSize'] + 4,
      ),
      color: const Color(0xFF0A0A0F),
      offset: const Offset(0, 55),
      shape: RoundedRectangleBorder(
        borderRadius: BorderRadius.circular(16),
        side: BorderSide(color: Colors.white.withValues(alpha: 0.1)),
      ),
      elevation: 8,
      onSelected: (value) {
        final u = usuario ?? authService.usuario;
        switch (value) {
```

```
            case 'perfil':
              if (u != null) {
                Navigator.pushNamed(
                  context,
                  AppRoutes.perfilUsuario,
                  arguments: u,
                );
              }
              break;
            case 'configuracion':
              if (u != null) {
                Navigator.pushNamed(
                  context,
                  AppRoutes.configuracionUsuarios,
                  arguments: u,
                );
              }
              break;
case 'logout':
              _mostrarDialogoCerrarSesion(authService);
              break;
          }
        },
        itemBuilder: (context) => [
          // Header del usuario
          _buildMenuHeader(usuario, sizes),

          // Mi Perfil
          _buildMenuItem(
            'perfil',
            'Mi Perfil',
            Icons.person_rounded,
            const Color(0xFFFF073A),
            sizes,
          ),

          // Configuración
          _buildMenuItem(
            'configuracion',
            'Configuración',
            Icons.settings_rounded,
            const Color(0xFFFF073A),
            sizes,
          ),

          // Divider
          PopupMenuItem(
            enabled: false,
            child: Divider(
              color: Colors.white.withOpacity(0.1),
              height: 1,
            ),
          ),
```

```
          // Cerrar Sesión
          _buildMenuItem(
            'logout',
            'Cerrar Sesión',
            Icons.logout_rounded,
            const Color(0xFFFF073A),
            sizes,
            isDanger: true,
          ),
        ],
      ),
    );
  }

  PopupMenuItem<String> _buildMenuHeader(
    Usuario? usuario,
    Map<String, dynamic> sizes,
  ) {
    return PopupMenuItem(
      enabled: false,
      child: Container(
        padding: const EdgeInsets.symmetric(vertical: 8),
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            Row(
              children: [
                Container(
                  decoration: BoxDecoration(
                    gradient: const LinearGradient(
                      colors: [Color(0xFFFF073A), Color(0xFFFF8E53)],
                    ),
                    shape: BoxShape.circle,
                    boxShadow: [
                      BoxShadow(
                        color: const Color(0xFFFF073A).withOpacity(0.4),
                        blurRadius: 8,
                      ),
                    ],
                  ),
                  child: CircleAvatar(
                    backgroundColor: Colors.transparent,
                    radius: 28,
                    child: Text(
                      (usuario != null && usuario.nombre.isNotEmpty ?
usuario.nombre[0] : 'A').toUpperCase(),
                      style: const TextStyle(
                        color: Colors.white,
                        fontWeight: FontWeight.bold,
                        fontSize: 24,
                      ),
                    ),
                  ),
                ),
```

```
              const SizedBox(width: 14),
              Expanded(
                child: Column(
                  crossAxisAlignment: CrossAxisAlignment.start,
                  children: [
                    Text(
                      usuario?.nombre ?? 'Administrador',
                      style: const TextStyle(
                        color: Colors.white,
                        fontWeight: FontWeight.bold,
                        fontSize: 17,
                      ),
                      overflow: TextOverflow.ellipsis,
                    ),
                    const SizedBox(height: 3),
                    Text(
                      usuario?.email ?? '',
                      style: TextStyle(
                        color: Colors.grey[500],
                        fontSize: 13,
                      ),
                      overflow: TextOverflow.ellipsis,
                    ),
                  ],
                ),
              ),
            ],
          ),
          const SizedBox(height: 16),
          Divider(color: Colors.white.withValues(alpha: 0.1), height: 1),
        ],
      ),
    ),
  );
}

PopupMenuItem<String> _buildMenuItem(
  String value,
  String label,
  IconData icon,
  Color color,
  Map<String, dynamic> sizes, {
  bool isDanger = false,
}) {
  return PopupMenuItem(
    value: value,
    child: Container(
      padding: const EdgeInsets.symmetric(vertical: 4),
      child: Row(
        children: [
          Container(
            padding: const EdgeInsets.all(8),
            decoration: BoxDecoration(
              color: isDanger
```

```
                    ? color.withValues(alpha: 0.15)
                    : const Color(0xFF0A0A0F),
                borderRadius: BorderRadius.circular(10),
              ),
              child: Icon(
                icon,
                color: color,
                size: sizes['iconSize'],
              ),
            ),
            const SizedBox(width: 14),
            Text(
              label,
              style: TextStyle(
                fontSize: sizes['fontSize'],
                color: isDanger ? color : Colors.white,
                fontWeight: isDanger ? FontWeight.w600 : FontWeight.w500,
              ),
            ),
          ],
        ),
      ),
    );
  }

  // ========================================
  // BOTTOM NAVIGATION BAR
  // ========================================
  Widget _buildBottomNavBar(Map<String, dynamic> sizes) {
    return Container(
      decoration: BoxDecoration(
        color: const Color(0xFF1A1A1A),
        border: Border(
          top: BorderSide(color: Colors.white.withValues(alpha: 0.1), width:
1),
        ),
        boxShadow: [
          BoxShadow(
            color: Colors.black.withValues(alpha: 0.3),
            blurRadius: 10,
            offset: const Offset(0, -3),
          ),
        ],
      ),
      child: BottomNavigationBar(
        currentIndex: _indiceSeleccionado,
        type: BottomNavigationBarType.fixed,
        backgroundColor: Colors.transparent,
        selectedItemColor: Theme.of(context).colorScheme.primary,
        unselectedItemColor: Colors.grey[600],
        selectedFontSize: sizes['fontSize'] - 1,
        unselectedFontSize: sizes['fontSize'] - 2,
        iconSize: sizes['iconSize'],
        selectedLabelStyle: const TextStyle(fontWeight: FontWeight.w600),
```

```
      elevation: 0,
      onTap: (index) {
        setState(() => _indiceSeleccionado = index);
      },
      items: [
        _buildNavItem(
          icon: Icons.inventory_2_outlined,
          activeIcon: Icons.inventory_2_rounded,
          label: 'Productos',
          index: 0,
        ),
        _buildNavItem(
          icon: Icons.shopping_cart_outlined,
          activeIcon: Icons.shopping_cart_rounded,
          label: 'Ventas',
          index: 1,
        ),
        _buildNavItem(
          icon: Icons.assignment_outlined,
          activeIcon: Icons.assignment_rounded,
          label: 'Pedidos',
          index: 2,
        ),
        _buildNavItem(
          icon: Icons.people_outlined,
          activeIcon: Icons.people_rounded,
          label: 'Usuarios',
          index: 3,
        ),
        _buildNavItem(
          icon: Icons.analytics_outlined,
          activeIcon: Icons.analytics_rounded,
          label: 'Reportes',
          index: 4,
        ),
      ],
    ),
  );
}

BottomNavigationBarItem _buildNavItem({
  required IconData icon,
  required IconData activeIcon,
  required String label,
  required int index,
}) {
  final isSelected = _indiceSeleccionado == index;

  return BottomNavigationBarItem(
    icon: Container(
      padding: const EdgeInsets.symmetric(vertical: 6, horizontal: 16),
      decoration: isSelected
          ? BoxDecoration(
              gradient: LinearGradient(
```

```
                  colors: [
                    const Color(0xFFFF073A).withValues(alpha: 0.2),
                    const Color(0xFFFF073A).withValues(alpha: 0.2),
                  ],
                ),
                borderRadius: BorderRadius.circular(12),
              )
            : null,
        child: Icon(icon),
      ),
      activeIcon: Container(
        padding: const EdgeInsets.symmetric(vertical: 6, horizontal: 16),
        decoration: BoxDecoration(
          gradient: LinearGradient(
            colors: [
              const Color(0xFFFF073A).withValues(alpha: 0.2),
              const Color(0xFFFF073A).withValues(alpha: 0.2),
            ],
          ),
          borderRadius: BorderRadius.circular(12),
        ),
        child: Icon(activeIcon),
      ),
      label: label,
    );
  }

  // ======================================
  // DIÁLOGOS
  // ======================================
  // (eliminado) _mostrarConfiguracion: ya no se usa; navegación abre
pantalla de configuración

  void _mostrarDialogoCerrarSesion(AuthService authService) {
    showDialog(
      context: context,
      builder: (context) => AlertDialog(
        backgroundColor: const Color(0xFF0A0A0F),
        shape: RoundedRectangleBorder(
          borderRadius: BorderRadius.circular(20),
          side: BorderSide(color: Colors.white.withValues(alpha: 0.1)),
        ),
        title: Row(
          children: [
            Container(
              padding: const EdgeInsets.all(10),
              decoration: BoxDecoration(
                color: const Color(0xFFFF073A).withValues(alpha: 0.2),
                borderRadius: BorderRadius.circular(12),
              ),
              child: const Icon(
                Icons.logout_rounded,
                color: Color(0xFFFF073A),
                size: 28,
```

```
      ),
    ),
    const SizedBox(width: 14),
    const Text(
      'Cerrar Sesión',
      style: TextStyle(
        color: Colors.white,
        fontWeight: FontWeight.bold,
      ),
    ),
  ],
),
content: Padding(
  padding: const EdgeInsets.only(top: 8),
  child: Text(
    '¿Estás seguro de que deseas cerrar sesión?',
    style: TextStyle(color: Colors.grey[400], fontSize: 15),
  ),
),
actions: [
  TextButton(
    onPressed: () => Navigator.pop(context),
    child: Text(
      'Cancelar',
      style: TextStyle(
        color: Colors.grey[400],
        fontSize: 15,
        fontWeight: FontWeight.w500,
      ),
    ),
  ),
  Container(
    decoration: BoxDecoration(
      gradient: const LinearGradient(
        colors: [Color(0xFFFF073A), Color(0xFFFF8E53)],
      ),
      borderRadius: BorderRadius.circular(12),
      boxShadow: [
        BoxShadow(
          color: const Color(0xFFFF073A).withValues(alpha: 0.4),
          blurRadius: 8,
          offset: const Offset(0, 3),
        ),
      ],
    ),
    child: ElevatedButton(
      onPressed: () async {
        Navigator.pop(context);
        authService.cerrarSesion();
        if (context.mounted) {
          Navigator.pushNamedAndRemoveUntil(
            context,
            AppRoutes.login,
            (route) => false,
```

```
                );
              }
            },
            style: ElevatedButton.styleFrom(
              backgroundColor: Colors.transparent,
              shadowColor: Colors.transparent,
              padding: const EdgeInsets.symmetric(
                horizontal: 24,
                vertical: 12,
              ),
              shape: RoundedRectangleBorder(
                borderRadius: BorderRadius.circular(12),
              ),
            ),
            child: const Text(
              'Cerrar Sesión',
              style: TextStyle(
                color: Colors.white,
                fontWeight: FontWeight.bold,
                fontSize: 15,
              ),
            ),
          ),
        ),
      ),
    ],
  ),
);
  }
}
```

Admin_pedidos_tab.dart

```
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import '../../viewmodels/pedido_viewmodel.dart';
import '../../viewmodels/usuario_viewmodel.dart';
import '../../data/models/pedido.dart';
import '../../data/models/pedido_estado.dart';

class AdminPedidosTab extends StatefulWidget {
  const AdminPedidosTab({Key? key}) : super(key: key);

  @override
  State<AdminPedidosTab> createState() => _AdminPedidosTabState();
}

class _AdminPedidosTabState extends State<AdminPedidosTab> {
  PedidoEstado? _filtroEstadoSeleccionado;
  final TextEditingController _busquedaController = TextEditingController();
  bool _mostrarFiltros = false;

  // Cache MediaQuery values
  late double _screenWidth;

  @override
```

```
  void initState() {
    super.initState();
    WidgetsBinding.instance.addPostFrameCallback((_) {
      if (mounted) {
        context.read<PedidoViewModel>().cargarPedidos();
      }
    });
  }

  @override
  void didChangeDependencies() {
    super.didChangeDependencies();
    _screenWidth = MediaQuery.of(context).size.width;
  }

  @override
  void dispose() {
    _busquedaController.dispose();
    super.dispose();
  }

  // Update helper methods to use cached values
  bool get _isMobile => _screenWidth < 600;

  bool get _isTablet => _screenWidth >= 600 && _screenWidth < 1024;

  EdgeInsets get _responsivePadding {
    if (_screenWidth < 480) return const EdgeInsets.all(10);
    if (_screenWidth < 768) return const EdgeInsets.all(14);
    if (_screenWidth < 1024) return const EdgeInsets.all(18);
    return const EdgeInsets.all(24);
  }

  double _getResponsiveFontSize(double base) {
    if (_screenWidth < 480) return base * 0.85;
    if (_screenWidth < 768) return base * 0.92;
    return base;
  }

  // Helper para verificar si el estado es final
  bool _esEstadoFinal(String estado) {
    final estadoLower = estado.toLowerCase();
    return estadoLower == 'cancelado' || estadoLower == 'completado' ||
estadoLower == 'entregado';
  }

  @override
  Widget build(BuildContext context) {
    return Consumer<PedidoViewModel>(
      builder: (context, viewModel, child) {
        return Container(
          color: const Color(0xFF0F0F0F),
          child: Column(
            children: [
```

```
            _buildEstadisticasHeader(viewModel),
            _buildBarraBusqueda(viewModel),
            if (_mostrarFiltros) _buildFiltrosExpandibles(viewModel),
            const SizedBox(height: 8),
            Expanded(
              child: _buildListaPedidos(viewModel),
            ),
          ],
        ),
      );
    },
  );
}

// ================================
//   ESTADÍSTICAS HEADER - RESPONSIVO
// ================================
Widget _buildEstadisticasHeader(PedidoViewModel viewModel) {
  final isMobile = _isMobile;
  final padding = _responsivePadding;

  return Container(
    margin: padding,
    padding: padding,
    decoration: BoxDecoration(
      gradient: LinearGradient(
        colors: [
          const Color(0xFF1A1A1A),
          const Color(0xFF2A2A2A),
        ],
        begin: Alignment.topLeft,
        end: Alignment.bottomRight,
      ),
      borderRadius: BorderRadius.circular(16),
      border: Border.all(
        color: Colors.white.withOpacity(0.1),
      ),
      boxShadow: [
        BoxShadow(
          color: Colors.black.withOpacity(0.3),
          blurRadius: 10,
          offset: const Offset(0, 4),
        ),
      ],
    ),
    child: Column(
      children: [
        Row(
          children: [
            Container(
              padding: EdgeInsets.all(isMobile ? 8 : 10),
              decoration: BoxDecoration(
                gradient: const LinearGradient(
                  colors: [Color(0xFFFF073A), Color(0xFFFF8E53)],
```

```
                    ),
                    borderRadius: BorderRadius.circular(12),
                  ),
                  child: Icon(
                    Icons.assignment_rounded,
                    color: Colors.white,
                    size: isMobile ? 20 : 24,
                  ),
                ),
                const SizedBox(width: 12),
                Expanded(
                  child: Text(
                    'Gestión de Pedidos',
                    style: TextStyle(
                      color: Colors.white,
                      fontSize: _getResponsiveFontSize(20),
                      fontWeight: FontWeight.bold,
                      letterSpacing: 0.5,
                    ),
                    maxLines: 1,
                    overflow: TextOverflow.ellipsis,
                  ),
                ),
                IconButton(
                  onPressed: () => viewModel.cargarPedidos(),
                  icon: Icon(
                    Icons.refresh_rounded,
                    color: Colors.white70,
                    size: isMobile ? 20 : 24,
                  ),
                  tooltip: 'Actualizar',
                ),
              ],
            ),
            SizedBox(height: isMobile ? 12 : 20),
          ],
        ),
      );
    }

    // ================================
    //   BARRA DE BÚSQUEDA - RESPONSIVO
    // ================================
    Widget _buildBarraBusqueda(PedidoViewModel viewModel) {
      final isMobile = _isMobile;
      final padding = _responsivePadding;

      return Container(
        margin: EdgeInsets.symmetric(horizontal: padding.left),
        padding: EdgeInsets.all(_screenWidth < 480 ? 10 : (isMobile ? 12 :
14)),
        decoration: BoxDecoration(
          color: const Color(0xFF1A1A1A),
          borderRadius: BorderRadius.circular(12),
```

```
      border: Border.all(
        color: Colors.white.withOpacity(0.1),
      ),
    ),
    child: isMobile
        ? Column(
            children: [
              _buildSearchField(),
              const SizedBox(height: 10),
              SizedBox(
                width: double.infinity,
                child: _buildFilterButton(),
              ),
            ],
          )
        : Row(
            children: [
              Expanded(child: _buildSearchField()),
              const SizedBox(width: 12),
              _buildFilterButton(),
            ],
          ),
  );
}

Widget _buildSearchField() {
  final isMobile = _isMobile;

  return TextField(
    controller: _busquedaController,
    style: TextStyle(
      color: Colors.white,
      fontSize: _getResponsiveFontSize(14),
    ),
    decoration: InputDecoration(
      hintText: _screenWidth < 300
          ? 'Buscar...'
          : (isMobile
              ? 'Buscar pedido...'
              : 'Buscar por número de pedido o cliente...'),
      hintStyle: TextStyle(
        color: Colors.grey[600],
        fontSize: _getResponsiveFontSize(13),
      ),
      prefixIcon: Icon(
        Icons.search,
        color: Colors.grey[600],
        size: _screenWidth < 480 ? 20 : 22,
      ),
      border: OutlineInputBorder(
        borderRadius: BorderRadius.circular(10),
        borderSide: BorderSide(color: Colors.grey[800]!),
      ),
      enabledBorder: OutlineInputBorder(
```

```
        borderRadius: BorderRadius.circular(10),
        borderSide: BorderSide(color: Colors.grey[800]!),
      ),
      focusedBorder: OutlineInputBorder(
        borderRadius: BorderRadius.circular(10),
        borderSide: const BorderSide(color: Color(0xFFFF073A)),
      ),
      filled: true,
      fillColor: const Color(0xFF0F0F0F),
      contentPadding: EdgeInsets.symmetric(
        horizontal: _screenWidth < 480 ? 10 : (isMobile ? 12 : 16),
        vertical: _screenWidth < 480 ? 8 : (isMobile ? 10 : 12),
      ),
      suffixIcon: _busquedaController.text.isNotEmpty
          ? IconButton(
              icon: Icon(
                Icons.clear,
                color: Colors.grey[600],
                size: _screenWidth < 480 ? 18 : 20,
              ),
              onPressed: () {
                _busquedaController.clear();
                setState(() {});
              },
            )
          : null,
    ),
    onChanged: (value) => setState(() {}),
  );
}

Widget _buildFilterButton() {
  final isMobile = _isMobile;

  return Container(
    decoration: BoxDecoration(
      gradient: _mostrarFiltros
          ? const LinearGradient(
              colors: [Color(0xFFFF073A), Color(0xFFFF8E53)],
            )
          : null,
      color: _mostrarFiltros ? null : const Color(0xFF2A2A2A),
      borderRadius: BorderRadius.circular(10),
      border: Border.all(
        color: _mostrarFiltros
            ? Colors.transparent
            : Colors.white.withOpacity(0.1),
      ),
    ),
    child: isMobile
        ? TextButton.icon(
            onPressed: () {
              setState(() {
                _mostrarFiltros = !_mostrarFiltros;
```

```
                });
              },
              icon: Icon(
                _mostrarFiltros ? Icons.filter_alt :
Icons.filter_alt_outlined,
                color: Colors.white,
              ),
              label: const Text(
                'Filtros',
                style: TextStyle(color: Colors.white),
              ),
            )
          : IconButton(
              onPressed: () {
                setState(() {
                  _mostrarFiltros = !_mostrarFiltros;
                });
              },
              icon: Icon(
                _mostrarFiltros ? Icons.filter_alt :
Icons.filter_alt_outlined,
                color: Colors.white,
              ),
              tooltip: 'Filtros',
            ),
    );
  }

  // ================================
  //   FILTROS EXPANDIBLES - RESPONSIVO
  // ================================
  Widget _buildFiltrosExpandibles(PedidoViewModel viewModel) {
    final padding = _responsivePadding;

    return Container(
      margin: EdgeInsets.fromLTRB(padding.left, 5, padding.right, 0),
      padding: padding,
      decoration: BoxDecoration(
        color: const Color(0xFF1A1A1A),
        borderRadius: BorderRadius.circular(12),
        border: Border.all(
          color: Colors.white.withOpacity(0.1),
        ),
      ),
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          Row(
            children: [
              Icon(
                Icons.filter_list_rounded,
                color: Colors.grey[400],
                size: 20,
              ),
```

```
            const SizedBox(width: 8),
            Text(
              'Filtrar por Estado',
              style: TextStyle(
                color: Colors.grey[400],
                fontWeight: FontWeight.w600,
                fontSize: _getResponsiveFontSize(13),
              ),
            ),
          ],
        ),
        const SizedBox(height: 12),
        Wrap(
          spacing: 8,
          runSpacing: 8,
          children: [
            _buildChipFiltro('Todos', null, viewModel),
            _buildChipFiltro('Pendiente', 'pendiente', viewModel),
            _buildChipFiltro('Pagado', 'pagado', viewModel),
            _buildChipFiltro('Procesando', 'procesando', viewModel),
            _buildChipFiltro('Enviado', 'enviado', viewModel),
            _buildChipFiltro('Entregado', 'entregado', viewModel),
            _buildChipFiltro('Cancelado', 'cancelado', viewModel),
          ],
        ),
      ],
    ),
  );
}

Widget _buildChipFiltro(
  String label,
  String? estado,
  PedidoViewModel viewModel,
) {
  final estadoEnum = estado != null ? _stringToEstado(estado) : null;
  final isSelected = _filtroEstadoSeleccionado == estadoEnum;
  final isMobile = _isMobile;

  Color chipColor;
  IconData chipIcon;

  switch (estado) {
    case 'pendiente':
      chipColor = const Color(0xFFFFA726);
      chipIcon = Icons.pending_actions_rounded;
      break;
    case 'pagado':
      chipColor = const Color(0xFF4CAF50);
      chipIcon = Icons.payment_rounded;
      break;
    case 'procesando':
      chipColor = const Color(0xFF4A9EFF);
      chipIcon = Icons.autorenew_rounded;
```

```
        break;
      case 'enviado':
        chipColor = const Color(0xFF9C27B0);
        chipIcon = Icons.local_shipping_rounded;
        break;
      case 'entregado':
        chipColor = const Color(0xFF66BB6A);
        chipIcon = Icons.check_circle_rounded;
        break;
      case 'completado':
        chipColor = const Color(0xFF2E7D32);
        chipIcon = Icons.verified_rounded;
        break;
      case 'cancelado':
        chipColor = const Color(0xFFFF073A);
        chipIcon = Icons.cancel_rounded;
        break;
      default:
        chipColor = Colors.grey;
        chipIcon = Icons.list_rounded;
    }

    return InkWell(
      onTap: () {
        setState(() {
          _filtroEstadoSeleccionado = estadoEnum;
        });
        if (estadoEnum == null) {
          viewModel.cargarPedidos();
        } else {
          viewModel.cargarPorEstado(estadoEnum);
        }
      },
      borderRadius: BorderRadius.circular(20),
      child: Container(
        padding: EdgeInsets.symmetric(
          horizontal: isMobile ? 12 : 16,
          vertical: isMobile ? 8 : 10,
        ),
        decoration: BoxDecoration(
          gradient: isSelected
              ? LinearGradient(
                  colors: [chipColor, chipColor.withOpacity(0.7)],
                )
              : null,
          color: isSelected ? null : chipColor.withOpacity(0.15),
          borderRadius: BorderRadius.circular(20),
          border: Border.all(
            color: isSelected ? Colors.transparent :
chipColor.withOpacity(0.3),
            width: 1.1,
          ),
        ),
        child: Row(
```

```
        mainAxisSize: MainAxisSize.min,
        children: [
          Icon(
            chipIcon,
            size: isMobile ? 16 : 18,
            color: isSelected ? Colors.white : chipColor,
          ),
          const SizedBox(width: 6),
          Text(
            label,
            style: TextStyle(
              color: isSelected ? Colors.white : chipColor,
              fontWeight: isSelected ? FontWeight.bold : FontWeight.w600,
              fontSize: isMobile ? 12 : 13,
            ),
          ),
          if (isSelected) ...[
            const SizedBox(width: 4),
            Icon(
              Icons.check,
              size: isMobile ? 14 : 16,
              color: Colors.white,
            ),
          ],
        ],
      ),
    ),
  );
}

// ================================
//   LISTA DE PEDIDOS
// ================================
Widget _buildListaPedidos(PedidoViewModel viewModel) {
  if (viewModel.cargando) {
    return Center(
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          Container(
            padding: const EdgeInsets.all(20),
            decoration: BoxDecoration(
              color: const Color(0xFF1A1A1A),
              borderRadius: BorderRadius.circular(16),
              border: Border.all(
                color: Colors.white.withOpacity(0.1),
              ),
            ),
            child: const CircularProgressIndicator(
              color: Color(0xFFFF073A),
            ),
          ),
          const SizedBox(height: 16),
          Text(
```

```
          'Cargando pedidos...',
          style: TextStyle(
            color: Colors.grey[400],
            fontSize: 14,
          ),
        ),
      ],
    ),
  );
}

if (viewModel.error != null) {
  return Center(
    child: Container(
      margin: const EdgeInsets.all(20),
      padding: const EdgeInsets.all(24),
      decoration: BoxDecoration(
        color: const Color(0xFF1A1A1A),
        borderRadius: BorderRadius.circular(16),
        border: Border.all(
          color: const Color(0xFFFF073A).withOpacity(0.3),
        ),
      ),
      child: Column(
        mainAxisSize: MainAxisSize.min,
        children: [
          Container(
            padding: const EdgeInsets.all(16),
            decoration: BoxDecoration(
              color: const Color(0xFFFF073A).withOpacity(0.15),
              shape: BoxShape.circle,
            ),
            child: const Icon(
              Icons.error_outline_rounded,
              size: 48,
              color: Color(0xFFFF073A),
            ),
          ),
          const SizedBox(height: 16),
          const Text(
            'Error al cargar pedidos',
            style: TextStyle(
              color: Colors.white,
              fontSize: 18,
              fontWeight: FontWeight.bold,
            ),
          ),
          const SizedBox(height: 8),
          Text(
            viewModel.error!,
            style: TextStyle(
              color: Colors.grey[400],
              fontSize: 14,
            ),
```

```
              textAlign: TextAlign.center,
            ),
            const SizedBox(height: 20),
            ElevatedButton.icon(
              onPressed: () => viewModel.cargarPedidos(),
              icon: const Icon(Icons.refresh_rounded),
              label: const Text('Reintentar'),
              style: ElevatedButton.styleFrom(
                backgroundColor: const Color(0xFFFF073A),
                foregroundColor: Colors.white,
                padding: const EdgeInsets.symmetric(
                  horizontal: 24,
                  vertical: 12,
                ),
                shape: RoundedRectangleBorder(
                  borderRadius: BorderRadius.circular(10),
                ),
              ),
            ),
          ],
        ),
      ),
    );
  }

  List<Pedido> pedidosFiltrados = viewModel.pedidos;
  if (_busquedaController.text.isNotEmpty) {
    final busqueda = _busquedaController.text.toLowerCase();
    pedidosFiltrados = pedidosFiltrados.where((pedido) {
      return pedido.numeroPedido.toLowerCase().contains(busqueda) ||
          (pedido.nombreCliente?.toLowerCase().contains(busqueda)  ??
false);
    }).toList();
  }

  if (pedidosFiltrados.isEmpty) {
    return Center(
      child: Container(
        margin: const EdgeInsets.all(20),
        padding: const EdgeInsets.all(24),
        decoration: BoxDecoration(
          color: const Color(0xFF1A1A1A),
          borderRadius: BorderRadius.circular(16),
          border: Border.all(
            color: Colors.white.withOpacity(0.1),
          ),
        ),
        child: Column(
          mainAxisSize: MainAxisSize.min,
          children: [
            Container(
              padding: const EdgeInsets.all(16),
              decoration: BoxDecoration(
                color: Colors.grey.withOpacity(0.1),
```

```
                shape: BoxShape.circle,
              ),
              child: Icon(
                Icons.inbox_rounded,
                size: 48,
                color: Colors.grey[600],
              ),
            ),
            const SizedBox(height: 16),
            Text(
              'No hay pedidos',
              style: TextStyle(
                fontSize: 18,
                color: Colors.grey[400],
                fontWeight: FontWeight.w600,
              ),
            ),
            const SizedBox(height: 8),
            Text(
              _busquedaController.text.isNotEmpty
                  ? 'No se encontraron pedidos con esos criterios'
                  : 'Aún no hay pedidos en el sistema',
              style: TextStyle(
                color: Colors.grey[600],
                fontSize: 14,
              ),
              textAlign: TextAlign.center,
            ),
          ],
        ),
      ),
    );
  }

  final padding = _responsivePadding;

  return RefreshIndicator(
    onRefresh: () async {
      await viewModel.cargarPedidos();
    },
    color: const Color(0xFFFF073A),
    backgroundColor: const Color(0xFF1A1A1A),
    child: ListView.builder(
      padding: padding,
      itemCount: pedidosFiltrados.length,
      itemBuilder: (context, index) {
        final pedido = pedidosFiltrados[index];
        return _buildPedidoCard(pedido, viewModel);
      },
    ),
  );
}

// ================================
```

```
  //   CARD DE PEDIDO - RESPONSIVO
  // ================================
  Widget _buildPedidoCard(Pedido pedido, PedidoViewModel viewModel) {
    final isMobile = _isMobile;
    final estadoPedido = _obtenerEstadoString(pedido);
    final mostrarAcciones = !_esEstadoFinal(estadoPedido);

    return Container(
      margin: EdgeInsets.only(bottom: _screenWidth < 480 ? 10 : (isMobile ?
12 : 16)),
      decoration: BoxDecoration(
        color: const Color(0xFF1A1A1A),
        borderRadius: BorderRadius.circular(16),
        border: Border.all(
          color: Colors.white.withOpacity(0.1),
        ),
        boxShadow: [
          BoxShadow(
            color: Colors.black.withOpacity(0.2),
            blurRadius: 8,
            offset: const Offset(0, 2),
          ),
        ],
      ),
      child: InkWell(
        onTap: () => _mostrarDetallePedido(pedido, viewModel),
        borderRadius: BorderRadius.circular(16),
        child: Padding(
          padding: EdgeInsets.all(_screenWidth < 480 ? 10 : (isMobile ? 12 :
16)),
          child: Column(
            crossAxisAlignment: CrossAxisAlignment.start,
            children: [
              isMobile
                  ? Column(
                      crossAxisAlignment: CrossAxisAlignment.start,
                      children: [
                        Row(
                          children: [
                            Container(
                              padding: EdgeInsets.all(_screenWidth < 480 ? 6
: 8),
                              decoration: BoxDecoration(
                                gradient: const LinearGradient(
                                  colors: [
                                    Color(0xFFFF073A),
                                    Color(0xFFFF8E53)
                                  ],
                                ),
                                borderRadius: BorderRadius.circular(10),
                              ),
                              child: Icon(
                                Icons.receipt_long_rounded,
                                color: Colors.white,
```

```
                    size: _screenWidth < 480 ? 16 : 18,
                  ),
                ),
                const SizedBox(width: 10),
                Expanded(
                  child: Column(
                    crossAxisAlignment: CrossAxisAlignment.start,
                    children: [
                      Text(
                        pedido.numeroPedido,
                        style: TextStyle(
                          fontWeight: FontWeight.bold,
                          fontSize: _getResponsiveFontSize(14),
                          color: Colors.white,
                        ),
                        maxLines: 1,
                        overflow: TextOverflow.ellipsis,
                      ),
                      Text(
                        _formatearFecha(pedido.fechaPedido),
                        style: TextStyle(
                          color: Colors.grey[600],
                          fontSize: _getResponsiveFontSize(11),
                        ),
                      ),
                    ],
                  ),
                ),
              ],
            ),
            const SizedBox(height: 12),
            _buildEstadoChip(estadoPedido),
          ],
        )
      : Row(
          mainAxisAlignment: MainAxisAlignment.spaceBetween,
          children: [
            Expanded(
              child: Row(
                children: [
                  Container(
                    padding: const EdgeInsets.all(8),
                    decoration: BoxDecoration(
                      gradient: const LinearGradient(
                        colors: [
                          Color(0xFFFF073A),
                          Color(0xFFFF8E53)
                        ],
                      ),
                      borderRadius: BorderRadius.circular(10),
                    ),
                    child: const Icon(
                      Icons.receipt_long_rounded,
                      color: Colors.white,
```

```
                                    size: 20,
                                  ),
                                ),
                                const SizedBox(width: 12),
                                Expanded(
                                  child: Column(
                                    crossAxisAlignment:
CrossAxisAlignment.start,
                                    children: [
                                      Text(
                                        pedido.numeroPedido,
                                        style: const TextStyle(
                                          fontWeight: FontWeight.bold,
                                          fontSize: 16,
                                          color: Colors.white,
                                          letterSpacing: 0.5,
                                        ),
                                        maxLines: 1,
                                        overflow: TextOverflow.ellipsis,
                                      ),
                                      const SizedBox(height: 2),
                                      Text(
                                        _formatearFecha(pedido.fechaPedido),
                                        style: TextStyle(
                                          color: Colors.grey[600],
                                          fontSize: 12,
                                        ),
                                      ),
                                    ],
                                  ),
                                ),
                              ],
                            ),
                          ),
                          _buildEstadoChip(estadoPedido),
                        ],
                      ),

                  const SizedBox(height: 16),

                  Divider(
                    color: Colors.white.withOpacity(0.1),
                    height: 1,
                  ),

                  const SizedBox(height: 16),

                  isMobile
                      ? Column(
                          crossAxisAlignment: CrossAxisAlignment.start,
                          children: [
                            _buildInfoItem(
                              Icons.person_rounded,
                              'Cliente',
```

```
              pedido.nombreCliente ?? 'Cliente desconocido',
            ),
            if (pedido.nombreVendedor != null) ...[
              SizedBox(height: _screenWidth < 480 ? 8 : 12),
              _buildInfoItem(
                Icons.badge_rounded,
                'Vendedor',
                pedido.nombreVendedor!,
              ),
            ],
          ],
        )
      : Row(
          children: [
            Expanded(
              child: _buildInfoItem(
                Icons.person_rounded,
                'Cliente',
                pedido.nombreCliente ?? 'Cliente desconocido',
              ),
            ),
            if (pedido.nombreVendedor != null)
              Expanded(
                child: _buildInfoItem(
                  Icons.badge_rounded,
                  'Vendedor',
                  pedido.nombreVendedor!,
                ),
              ),
          ],
        ),

SizedBox(height: _screenWidth < 480 ? 12 : 16),

Row(
  children: [
    Expanded(
      child: Container(
        padding: EdgeInsets.symmetric(
          horizontal: _screenWidth < 480 ? 12 : 16,
          vertical: _screenWidth < 480 ? 8 : 10,
        ),
        decoration: BoxDecoration(
          gradient: LinearGradient(
            colors: [
              const Color(0xFF66BB6A).withOpacity(0.2),
              const Color(0xFF4CAF50).withOpacity(0.2),
            ],
          ),
          borderRadius: BorderRadius.circular(10),
          border: Border.all(
            color: const Color(0xFF66BB6A).withOpacity(0.3),
          ),
        ),
```

```
          child: Row(
            mainAxisSize: MainAxisSize.min,
            children: [
              Icon(
                Icons.attach_money_rounded,
                color: const Color(0xFF66BB6A),
                size: _screenWidth < 480 ? 18 : 20,
              ),
              const SizedBox(width: 8),
              Text(
                '\$${pedido.total.toStringAsFixed(2)}',
                style: TextStyle(
                  fontWeight: FontWeight.bold,
                  fontSize: _getResponsiveFontSize(16),
                  color: const Color(0xFF66BB6A),
                ),
              ),
            ],
          ),
        ),
      ),
    ],
  ),

  if (mostrarAcciones) ...[
    const SizedBox(height: 16),
    Divider(
      color: Colors.white.withOpacity(0.1),
      height: 1,
    ),
    const SizedBox(height: 16),
    isMobile
        ? _buildActionButtonsGrid(pedido, viewModel)
        : Row(
            mainAxisAlignment: MainAxisAlignment.end,
            children: [
              _buildActionButton(
                icon: Icons.visibility_rounded,
                color: const Color(0xFF4A9EFF),
                tooltip: 'Ver detalles',
                onPressed: () =>
                    _mostrarDetallePedido(pedido, viewModel),
              ),
              const SizedBox(width: 8),
              _buildActionButton(
                icon: Icons.edit_rounded,
                color: const Color(0xFFFFA726),
                tooltip: 'Cambiar estado',
                onPressed: () =>
                    _mostrarCambiarEstado(pedido, viewModel),
              ),
              const SizedBox(width: 8),
              _buildActionButton(
                icon: Icons.person_add_rounded,
```

```
                          color: const Color(0xFF9C27B0),
                          tooltip: 'Asignar vendedor',
                          onPressed: () =>
                              _mostrarAsignarVendedor(pedido, viewModel),
                        ),
                        const SizedBox(width: 8),
                        _buildActionButton(
                          icon: Icons.cancel_rounded,
                          color: const Color(0xFFFF073A),
                          tooltip: 'Cancelar pedido',
                          onPressed: () =>
                              _confirmarCancelar(pedido, viewModel),
                        ),
                      ],
                    ),
                  ],
                ),
              ),
            ),
          ),
        );
      }

      Widget _buildActionButtonsGrid(Pedido pedido, PedidoViewModel viewModel) {
        final actions = <Map<String, dynamic>>[
          {
            'icon': Icons.visibility_rounded,
            'color': const Color(0xFF4A9EFF),
            'tooltip': 'Ver',
            'onPressed': () => _mostrarDetallePedido(pedido, viewModel),
          },
          {
            'icon': Icons.edit_rounded,
            'color': const Color(0xFFFFA726),
            'tooltip': 'Estado',
            'onPressed': () => _mostrarCambiarEstado(pedido, viewModel),
          },
          {
            'icon': Icons.person_add_rounded,
            'color': const Color(0xFF9C27B0),
            'tooltip': 'Asignar',
            'onPressed': () => _mostrarAsignarVendedor(pedido, viewModel),
          },
          {
            'icon': Icons.cancel_rounded,
            'color': const Color(0xFFFF073A),
            'tooltip': 'Cancelar',
            'onPressed': () => _confirmarCancelar(pedido, viewModel),
          },
        ];

        return GridView.builder(
          shrinkWrap: true,
          physics: const NeverScrollableScrollPhysics(),
```

```
    gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(
      crossAxisCount: 2,
      crossAxisSpacing: 8,
      mainAxisSpacing: 8,
      childAspectRatio: _screenWidth < 480 ? 2.5 : 3,
    ),
    itemCount: actions.length,
    itemBuilder: (context, index) {
      final action = actions[index];
      return _buildActionButtonExpanded(
        icon: action['icon'],
        color: action['color'],
        tooltip: action['tooltip'],
        onPressed: action['onPressed'],
      );
    },
  );
}

Widget _buildActionButtonExpanded({
  required IconData icon,
  required Color color,
  required String tooltip,
  required VoidCallback onPressed,
}) {
  return InkWell(
    onTap: onPressed,
    borderRadius: BorderRadius.circular(10),
    child: Container(
      padding: EdgeInsets.symmetric(
        horizontal: _screenWidth < 480 ? 8 : 12,
        vertical: _screenWidth < 480 ? 6 : 8,
      ),
      decoration: BoxDecoration(
        color: color.withOpacity(0.15),
        borderRadius: BorderRadius.circular(10),
        border: Border.all(
          color: color.withOpacity(0.3),
        ),
      ),
      child: Row(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          Icon(icon, color: color, size: _screenWidth < 480 ? 16 : 18),
          const SizedBox(width: 6),
          Flexible(
            child: Text(
              tooltip,
              style: TextStyle(
                color: color,
                fontSize: _screenWidth < 480 ? 11 : 12,
                fontWeight: FontWeight.w600,
              ),
              overflow: TextOverflow.ellipsis,
```

```
              maxLines: 1,
            ),
          ),
        ],
      ),
    ),
  );
}

Widget _buildInfoItem(IconData icon, String label, String value) {
  final isMobile = _isMobile;

  return Row(
    children: [
      Icon(icon, size: isMobile ? 16 : 18, color: Colors.grey[600]),
      const SizedBox(width: 8),
      Expanded(
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            Text(
              label,
              style: TextStyle(
                color: Colors.grey[600],
                fontSize: isMobile ? 10 : 11,
                fontWeight: FontWeight.w500,
              ),
            ),
            const SizedBox(height: 2),
            Text(
              value,
              style: TextStyle(
                color: Colors.white,
                fontSize: isMobile ? 12 : 13,
                fontWeight: FontWeight.w600,
              ),
              overflow: TextOverflow.ellipsis,
            ),
          ],
        ),
      ),
    ],
  );
}

Widget _buildActionButton({
  required IconData icon,
  required Color color,
  required String tooltip,
  required VoidCallback onPressed,
}) {
  return Tooltip(
    message: tooltip,
    child: InkWell(
```

```
      onTap: onPressed,
      borderRadius: BorderRadius.circular(10),
      child: Container(
        padding: EdgeInsets.all(_screenWidth < 768 ? 8 : 10),
        decoration: BoxDecoration(
          color: color.withOpacity(0.15),
          borderRadius: BorderRadius.circular(10),
          border: Border.all(
            color: color.withOpacity(0.3),
          ),
        ),
        child: Icon(
          icon,
          color: color,
          size: _screenWidth < 768 ? 18 : 20,
        ),
      ),
    ),
  );
}

Widget _buildEstadoChip(String estado) {
  Color color;
  IconData icon;

  switch (estado.toLowerCase()) {
    case 'pendiente':
      color = const Color(0xFFFFA726);
      icon = Icons.pending_actions_rounded;
      break;
    case 'pagado':
      color = const Color(0xFF4CAF50);
      icon = Icons.payment_rounded;
      break;
    case 'procesando':
      color = const Color(0xFF4A9EFF);
      icon = Icons.autorenew_rounded;
      break;
    case 'enviado':
      color = const Color(0xFF9C27B0);
      icon = Icons.local_shipping_rounded;
      break;
    case 'entregado':
      color = const Color(0xFF66BB6A);
      icon = Icons.check_circle_rounded;
      break;
    case 'completado':
      color = const Color(0xFF2E7D32);
      icon = Icons.verified_rounded;
      break;
    case 'cancelado':
      color = const Color(0xFFFF073A);
      icon = Icons.cancel_rounded;
      break;
```

```
      default:
        color = Colors.grey;
        icon = Icons.help_outline_rounded;
    }

    return Container(
      padding: EdgeInsets.symmetric(
        horizontal: _screenWidth < 480 ? 10 : 12,
        vertical: _screenWidth < 480 ? 6 : 8,
      ),
      decoration: BoxDecoration(
        gradient: LinearGradient(
          colors: [color, color.withOpacity(0.7)],
        ),
        borderRadius: BorderRadius.circular(20),
      ),
      child: Row(
        mainAxisSize: MainAxisSize.min,
        children: [
          Icon(icon, color: Colors.white, size: _screenWidth < 480 ? 14 :
16),
          const SizedBox(width: 6),
          Text(
            estado.toUpperCase(),
            style: TextStyle(
              color: Colors.white,
              fontWeight: FontWeight.bold,
              fontSize: _screenWidth < 480 ? 10 : 12,
              letterSpacing: 0.5,
            ),
          ),
        ],
      ),
    );
  }

  // ================================
  //   DIÁLOGOS - RESPONSIVOS
  // ================================

  void _mostrarDetallePedido(Pedido pedido, PedidoViewModel viewModel) async
{
    final pedidoDetalle = await viewModel.obtenerPedidoDetalle(pedido.id!);
    if (!mounted || pedidoDetalle == null) return;

    final dialogWidth = _screenWidth > 600 ? 500.0 : _screenWidth * 0.9;

    if (!mounted) return;

    showDialog(
      context: context,
      builder: (dialogContext) => Dialog(
        backgroundColor: Colors.transparent,
        child: Container(
```

```
constraints: BoxConstraints(maxWidth: dialogWidth),
decoration: BoxDecoration(
  color: const Color(0xFF1A1A1A),
  borderRadius: BorderRadius.circular(20),
  border: Border.all(
    color: Colors.white.withOpacity(0.1),
  ),
  boxShadow: [
    BoxShadow(
      color: Colors.black.withOpacity(0.5),
      blurRadius: 20,
    ),
  ],
),
child: Column(
  mainAxisSize: MainAxisSize.min,
  children: [
    Container(
      padding: const EdgeInsets.all(20),
      decoration: const BoxDecoration(
        gradient: LinearGradient(
          colors: [Color(0xFFFF073A), Color(0xFFFF8E53)],
        ),
        borderRadius: BorderRadius.only(
          topLeft: Radius.circular(20),
          topRight: Radius.circular(20),
        ),
      ),
      child: Row(
        children: [
          const Icon(
            Icons.receipt_long_rounded,
            color: Colors.white,
            size: 28,
          ),
          const SizedBox(width: 12),
          Expanded(
            child: Column(
              crossAxisAlignment: CrossAxisAlignment.start,
              children: [
                Text(
                  'Detalle del Pedido',
                  style: TextStyle(
                    color: Colors.white,
                    fontSize: _getResponsiveFontSize(18),
                    fontWeight: FontWeight.bold,
                  ),
                ),
                Text(
                  pedido.numeroPedido,
                  style: TextStyle(
                    color: Colors.white.withOpacity(0.9),
                    fontSize: _getResponsiveFontSize(14),
                  ),
```

```
            ),
          ],
        ),
      ),
      IconButton(
        onPressed: () => Navigator.pop(dialogContext),
        icon: const Icon(Icons.close, color: Colors.white),
      ),
    ],
  ),
),
Flexible(
  child: SingleChildScrollView(
    padding: _responsivePadding,
    child: Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
        _buildDetalleSeccion(
          'Información General',
          Icons.info_rounded,
          [
            _buildDetalleRow(
                'Cliente', pedido.nombreCliente ?? 'N/A'),
            _buildDetalleRow('Vendedor',
                pedido.nombreVendedor ?? 'Sin asignar'),
            _buildDetalleRow(
                'Fecha', _formatearFecha(pedido.fechaPedido)),
          ],
        ),
        const SizedBox(height: 20),
        _buildDetalleSeccion(
          'Estado del Pedido',
          Icons.flag_rounded,
          [
            Center(
                child: _buildEstadoChip(
                    _obtenerEstadoString(pedido))),
          ],
        ),
        const SizedBox(height: 20),
        _buildDetalleSeccion(
          'Productos',
          Icons.shopping_bag_rounded,
          [
            if (pedidoDetalle.detalles != null)
              ...pedidoDetalle.detalles!.map(
                (detalle) => Container(
                  margin: const EdgeInsets.only(bottom: 12),
                  padding: const EdgeInsets.all(12),
                  decoration: BoxDecoration(
                    color: const Color(0xFF0F0F0F),
                    borderRadius: BorderRadius.circular(10),
                    border: Border.all(
                      color: Colors.white.withOpacity(0.1),
```

```
                                    ),
                                  ),
                                  child: Row(
                                    mainAxisAlignment:
                                        MainAxisAlignment.spaceBetween,
                                    children: [
                                      Expanded(
                                        child: Column(
                                          crossAxisAlignment:
                                              CrossAxisAlignment.start,
                                          children: [
                                            Text(
                                              detalle.nombreProducto ??
                                                  'Producto',
                                              style: TextStyle(
                                                color: Colors.white,
                                                fontWeight: FontWeight.w600,
                                                fontSize:
_getResponsiveFontSize(14),
                                              ),
                                            ),
                                            const SizedBox(height: 4),
                                            Text(
                                              'Cantidad: ${detalle.cantidad}',
                                              style: TextStyle(
                                                color: Colors.grey[500],
                                                fontSize:
_getResponsiveFontSize(12),
                                              ),
                                            ),
                                          ],
                                        ),
                                      ),
                                      Text(
'\$${detalle.subtotal.toStringAsFixed(2)}',
                                        style: TextStyle(
                                          color: const Color(0xFF66BB6A),
                                          fontWeight: FontWeight.bold,
                                          fontSize: _getResponsiveFontSize(16),
                                        ),
                                      ),
                                    ],
                                  ),
                                ),
                              ),
                            ),
                          ],
                        ),
                        const SizedBox(height: 20),
                        Container(
                          padding: const EdgeInsets.all(16),
                          decoration: BoxDecoration(
                            gradient: LinearGradient(
                              colors: [
```

```
                            const Color(0xFF66BB6A).withOpacity(0.2),
                            const Color(0xFF4CAF50).withOpacity(0.2),
                          ],
                        ),
                        borderRadius: BorderRadius.circular(12),
                        border: Border.all(
                          color: const Color(0xFF66BB6A).withOpacity(0.3),
                        ),
                      ),
                      child: Row(
                        mainAxisAlignment: MainAxisAlignment.spaceBetween,
                        children: [
                          Text(
                            'TOTAL',
                            style: TextStyle(
                              color: Colors.white,
                              fontWeight: FontWeight.bold,
                              fontSize: _getResponsiveFontSize(18),
                            ),
                          ),
                          Text(
                            '\$${pedido.total.toStringAsFixed(2)}',
                            style: TextStyle(
                              color: const Color(0xFF66BB6A),
                              fontWeight: FontWeight.bold,
                              fontSize: _getResponsiveFontSize(24),
                            ),
                          ),
                        ],
                      ),
                    ),
                  ),
                ],
              ),
            ),
          ),
        ),
      ),
    );
  }

  Widget _buildDetalleSeccion(
    String titulo,
    IconData icon,
    List<Widget> contenido,
  ) {
    final isMobile = _isMobile;

    return Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
        Row(
          children: [
```

```
            Container(
              padding: const EdgeInsets.all(8),
              decoration: BoxDecoration(
                gradient: const LinearGradient(
                  colors: [Color(0xFFFF073A), Color(0xFFFF8E53)],
                ),
                borderRadius: BorderRadius.circular(8),
              ),
              child: Icon(icon, color: Colors.white, size: isMobile ? 16 :
18),
            ),
            const SizedBox(width: 10),
            Text(
              titulo,
              style: TextStyle(
                color: Colors.white,
                fontWeight: FontWeight.bold,
                fontSize: _getResponsiveFontSize(16),
              ),
            ),
          ],
        ),
        const SizedBox(height: 12),
        ...contenido,
      ],
    );
  }

  Widget _buildDetalleRow(String label, String valor) {
    final isMobile = _isMobile;

    return Padding(
      padding: const EdgeInsets.only(bottom: 8),
      child: Row(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          SizedBox(
            width: isMobile ? 80 : 100,
            child: Text(
              '$label:',
              style: TextStyle(
                color: Colors.grey[500],
                fontSize: _getResponsiveFontSize(14),
              ),
            ),
          ),
          Expanded(
            child: Text(
              valor,
              style: TextStyle(
                color: Colors.white,
                fontWeight: FontWeight.w500,
                fontSize: _getResponsiveFontSize(14),
              ),
```

```
          ),
        ),
      ],
    ),
  );
}

void _mostrarCambiarEstado(Pedido pedido, PedidoViewModel viewModel) {
  final estados = [
    PedidoEstado.pendientePago,
    PedidoEstado.pagado,
    PedidoEstado.procesando,
    PedidoEstado.enviado,
    PedidoEstado.entregado,
  ];
  final estadoActual = pedido.estado;
  PedidoEstado? estadoSeleccionado = estadoActual;

  final dialogWidth = _screenWidth > 600 ? 450.0 : _screenWidth * 0.92;
  final isMobile = _screenWidth < 600;

  showDialog(
    context: context,
    builder: (dialogContext) => Dialog(
      backgroundColor: Colors.transparent,
      child: Container(
        constraints: BoxConstraints(maxWidth: dialogWidth),
        decoration: BoxDecoration(
          color: const Color(0xFF1A1A1A),
          borderRadius: BorderRadius.circular(20),
          border: Border.all(
            color: Colors.white.withOpacity(0.1),
          ),
        ),
        child: Column(
          mainAxisSize: MainAxisSize.min,
          children: [
            Container(
              padding: EdgeInsets.all(isMobile ? 14 : 20),
              decoration: const BoxDecoration(
                gradient: LinearGradient(
                  colors: [Color(0xFFFF073A), Color(0xFFFF8E53)],
                ),
                borderRadius: BorderRadius.only(
                  topLeft: Radius.circular(20),
                  topRight: Radius.circular(20),
                ),
              ),
              child: Row(
                children: [
                  Icon(
                    Icons.edit_rounded,
                    color: Colors.white,
                    size: isMobile ? 20 : 24,
```

```
      ),
      const SizedBox(width: 12),
      Expanded(
        child: Text(
          'Cambiar Estado',
          style: TextStyle(
            color: Colors.white,
            fontSize: isMobile ? 16 : 18,
            fontWeight: FontWeight.bold,
          ),
        ),
      ),
    ],
  ),
),
Flexible(
  child: SingleChildScrollView(
    padding: EdgeInsets.all(isMobile ? 14 : 20),
    child: StatefulBuilder(
      builder: (context, setDialogState) => Column(
        children: estados.map((estado) {
          final isSelected = estadoSeleccionado == estado;
          Color color;
          IconData icon;

          switch (estado) {
            case PedidoEstado.pendientePago:
              color = const Color(0xFFFFA726);
              icon = Icons.pending_actions_rounded;
              break;
            case PedidoEstado.pagado:
              color = const Color(0xFF4CAF50);
              icon = Icons.payment_rounded;
              break;
            case PedidoEstado.procesando:
              color = const Color(0xFF4A9EFF);
              icon = Icons.autorenew_rounded;
              break;
            case PedidoEstado.enviado:
              color = const Color(0xFF9C27B0);
              icon = Icons.local_shipping_rounded;
              break;
            case PedidoEstado.entregado:
              color = const Color(0xFF66BB6A);
              icon = Icons.check_circle_rounded;
              break;
            case PedidoEstado.cancelado:
              color = Colors.red;
              icon = Icons.cancel_rounded;
              break;
          }

          return Container(
            margin: const EdgeInsets.only(bottom: 10),
```

```
                            decoration: BoxDecoration(
                              gradient: isSelected
                                  ? LinearGradient(
                                      colors: [
                                        color.withOpacity(0.2),
                                        color.withOpacity(0.1)
                                      ],
                                    )
                                  : null,
                              color: isSelected ? null : const
Color(0xFF0F0F0F),
                              borderRadius: BorderRadius.circular(12),
                              border: Border.all(
                                color: isSelected
                                    ? color
                                    : Colors.white.withOpacity(0.1),
                                width: isSelected ? 2 : 1,
                              ),
                            ),
                            child: RadioListTile<PedidoEstado>(
                              contentPadding: EdgeInsets.symmetric(
                                horizontal: isMobile ? 12 : 16,
                                vertical: isMobile ? 4 : 8,
                              ),
                              title: Row(
                                children: [
                                  Icon(icon,
                                      color: color, size: isMobile ? 18 : 22),
                                  const SizedBox(width: 10),
                                  Expanded(
                                    child: Text(
                                      estado.nombre.toUpperCase(),
                                      style: TextStyle(
                                        color: isSelected ? color :
Colors.white,
                                        fontWeight: isSelected
                                            ? FontWeight.bold
                                            : FontWeight.w500,
                                        fontSize: isMobile ? 13 : 14,
                                      ),
                                    ),
                                  ),
                                ],
                              ),
                              value: estado,
                              groupValue: estadoSeleccionado,
                              activeColor: color,
                              onChanged: (value) {
                                setDialogState(() {
                                  estadoSeleccionado = value;
                                });
                              },
                            ),
                          );
```

```
        }).toList(),
      ),
    ),
  ),
),
Padding(
  padding: EdgeInsets.all(isMobile ? 14 : 20),
  child: isMobile
      ? Column(
          children: [
            SizedBox(
              width: double.infinity,
              height: 44,
              child: TextButton(
                onPressed: () => Navigator.pop(dialogContext),
                style: TextButton.styleFrom(
                  backgroundColor: const Color(0xFF2A2A2A),
                  shape: RoundedRectangleBorder(
                    borderRadius: BorderRadius.circular(10),
                  ),
                ),
                child: const Text(
                  'Cancelar',
                  style: TextStyle(
                    color: Colors.white,
                    fontWeight: FontWeight.w600,
                    fontSize: 14,
                  ),
                ),
              ),
            ),
            const SizedBox(height: 10),
            SizedBox(
              width: double.infinity,
              height: 44,
              child: Container(
                decoration: BoxDecoration(
                  gradient: const LinearGradient(
                    colors: [
                      Color(0xFFFF6B6B),
                      Color(0xFFFF8E53)
                    ],
                  ),
                  borderRadius: BorderRadius.circular(10),
                ),
                child: ElevatedButton(
                  onPressed: () async {
                    if (estadoSeleccionado != null &&
                        estadoSeleccionado != estadoActual) {
                      final success =
                          await viewModel.actualizarEstado(
                        pedido.id!,
                        estadoSeleccionado!,
                      );
```

```
                    if (mounted) {
                      Navigator.pop(dialogContext);

                      if (success) {
                        await viewModel.cargarPedidos();

                        if (mounted) {
                          ScaffoldMessenger.of(context)
                              .showSnackBar(
                            SnackBar(
                              content: const Row(
                                children: [
                                  Icon(Icons.check_circle,
                                      color: Colors.white,
                                      size: 20),
                                  SizedBox(width: 12),
                                  Expanded(
                                    child: Text(
                                      'Estado actualizado',
                                      style: TextStyle(
                                          fontSize: 14),
                                    ),
                                  ),
                                ],
                              ),
                              backgroundColor:
                                  const Color(0xFF66BB6A),
                              behavior:
                                  SnackBarBehavior.floating,
                              shape: RoundedRectangleBorder(
                                borderRadius:

BorderRadius.circular(10),

                              ),
                            ),
                          );
                        }
                      }
                    } else {
                      Navigator.pop(dialogContext);
                    }
                  },
                  style: ElevatedButton.styleFrom(
                    backgroundColor: Colors.transparent,
                    shadowColor: Colors.transparent,
                    shape: RoundedRectangleBorder(
                      borderRadius: BorderRadius.circular(10),
                    ),
                  ),
                  child: const Text(
                    'Guardar',
                    style: TextStyle(
```

```
                                    color: Colors.white,
                                    fontWeight: FontWeight.bold,
                                    fontSize: 14,
                                  ),
                                ),
                              ),
                            ),
                          ),
                        ],
                      )
                    : Row(
                        children: [
                          Expanded(
                            child: SizedBox(
                              height: 40,
                              child: TextButton(
                                onPressed: () =>
Navigator.pop(dialogContext),
                                style: TextButton.styleFrom(
                                  backgroundColor: const Color(0xFF2A2A2A),
                                  shape: RoundedRectangleBorder(
                                    borderRadius: BorderRadius.circular(10),
                                  ),
                                ),
                                child: const Text(
                                  'Cancelar',
                                  style: TextStyle(
                                    color: Colors.white,
                                    fontWeight: FontWeight.w600,
                                  ),
                                ),
                              ),
                            ),
                          ),
                          const SizedBox(width: 12),
                          Expanded(
                            child: SizedBox(
                              height: 44,
                              child: Container(
                                decoration: BoxDecoration(
                                  gradient: const LinearGradient(
                                    colors: [
                                      Color(0xFFFF6B6B),
                                      Color(0xFFFF8E53)
                                    ],
                                  ),
                                  borderRadius: BorderRadius.circular(10),
                                ),
                                child: ElevatedButton(
                                  onPressed: () async {
                                    if (estadoSeleccionado != null &&
                                        estadoSeleccionado != estadoActual) {
                                      final success =
                                          await viewModel.actualizarEstado(
```

```
                                    pedido.id!,
                                    estadoSeleccionado!,
                                  );

                                  if (mounted) {
                                    Navigator.pop(dialogContext);

                                    if (success) {
                                      await viewModel.cargarPedidos();

                                      if (mounted) {
                                        ScaffoldMessenger.of(context)
                                            .showSnackBar(
                                          SnackBar(
                                            content: const Row(
                                              children: [
                                                Icon(Icons.check_circle,
                                                    color: Colors.white),
                                                SizedBox(width: 12),
                                                Text(
                                                    'Estado actualizado
correctamente'),
                                              ],
                                            ),
                                            backgroundColor:
                                                const Color(0xFF66BB6A),
                                            behavior:

SnackBarBehavior.floating,
                                            shape:

RoundedRectangleBorder(
                                              borderRadius:

BorderRadius.circular(10),
                                            ),
                                          ),
                                        );
                                      }
                                    }
                                  }
                                } else {
                                  Navigator.pop(dialogContext);
                                }
                              },
                              style: ElevatedButton.styleFrom(
                                backgroundColor: Colors.transparent,
                                shadowColor: Colors.transparent,
                                shape: RoundedRectangleBorder(
                                  borderRadius:
BorderRadius.circular(10),
                                ),
                              ),
                              child: const Text(
                                'Guardar',
```

```
                                    style: TextStyle(
                                      color: Colors.white,
                                      fontWeight: FontWeight.bold,
                                    ),
                                  ),
                                ),
                              ),
                            ),
                          ),
                        ],
                      ),
                    ),
                  ],
                ),
              ),
            ),
          ),
        );
      }

  void _mostrarAsignarVendedor(Pedido pedido, PedidoViewModel viewModel)
async {
    final usuarioViewModel = context.read<UsuarioViewModel>();
    await usuarioViewModel.cargarVendedores();
    if (!mounted) return;

    final dialogWidth = _screenWidth > 600 ? 400.0 : _screenWidth * 0.9;
    final isMobile = _isMobile;

    if (!mounted) return;

    showDialog(
      context: context,
      builder: (dialogContext) => Dialog(
        backgroundColor: Colors.transparent,
        child: Container(
          constraints: BoxConstraints(
            maxWidth: dialogWidth,
            maxHeight: MediaQuery.of(dialogContext).size.height * 0.7,
          ),
          decoration: BoxDecoration(
            color: const Color(0xFF1A1A1A),
            borderRadius: BorderRadius.circular(20),
            border: Border.all(
              color: Colors.white.withOpacity(0.1),
            ),
          ),
          child: Column(
            mainAxisSize: MainAxisSize.min,
            children: [
              Container(
                padding: EdgeInsets.all(isMobile ? 16 : 20),
                decoration: const BoxDecoration(
                  gradient: LinearGradient(
                    colors: [Color(0xFF9C27B0), Color(0xFFBA68C8)],
```

```
      ),
      borderRadius: BorderRadius.only(
        topLeft: Radius.circular(20),
        topRight: Radius.circular(20),
      ),
    ),
    child: Row(
      children: [
        Icon(
          Icons.person_add_rounded,
          color: Colors.white,
          size: isMobile ? 20 : 24,
        ),
        const SizedBox(width: 12),
        Text(
          'Asignar Vendedor',
          style: TextStyle(
            color: Colors.white,
            fontSize: _getResponsiveFontSize(18),
            fontWeight: FontWeight.bold,
          ),
        ),
      ],
    ),
  ),
),
Flexible(
  child: Consumer<UsuarioViewModel>(
    builder: (context, uvm, _) {
      if (uvm.cargando) {
        return const Center(
          child: Padding(
            padding: EdgeInsets.all(40),
            child: CircularProgressIndicator(
              color: Color(0xFF9C27B0),
            ),
          ),
        );
      }

      if (uvm.usuarios.isEmpty) {
        return Center(
          child: Padding(
            padding: const EdgeInsets.all(40),
            child: Column(
              mainAxisSize: MainAxisSize.min,
              children: [
                Icon(
                  Icons.person_off_rounded,
                  size: 48,
                  color: Colors.grey[600],
                ),
                const SizedBox(height: 16),
                Text(
                  'No hay vendedores disponibles',
```

```
            style: TextStyle(
              color: Colors.grey[400],
              fontSize: 14,
            ),
          ),
        ],
      ),
    ),
  );
}

return ListView.builder(
  shrinkWrap: true,
  padding: const EdgeInsets.all(12),
  itemCount: uvm.usuarios.length,
  itemBuilder: (context, index) {
    final vendedor = uvm.usuarios[index];
    return Container(
      margin: const EdgeInsets.only(bottom: 8),
      decoration: BoxDecoration(
        color: const Color(0xFF0F0F0F),
        borderRadius: BorderRadius.circular(12),
        border: Border.all(
          color: Colors.white.withOpacity(0.1),
        ),
      ),
      child: ListTile(
        contentPadding: EdgeInsets.symmetric(
          horizontal: isMobile ? 12 : 16,
          vertical: 8,
        ),
        leading: Container(
          padding: const EdgeInsets.all(10),
          decoration: BoxDecoration(
            gradient: const LinearGradient(
              colors: [
                Color(0xFF9C27B0),
                Color(0xFFBA68C8)
              ],
            ),
            borderRadius: BorderRadius.circular(10),
          ),
          child: Text(
            vendedor.nombre[0].toUpperCase(),
            style: TextStyle(
              color: Colors.white,
              fontWeight: FontWeight.bold,
              fontSize: _getResponsiveFontSize(18),
            ),
          ),
        ),
        title: Text(
          vendedor.nombre,
          style: TextStyle(
```

```
                                    color: Colors.white,
                                    fontWeight: FontWeight.w600,
                                    fontSize: _getResponsiveFontSize(14),
                                  ),
                                ),
                                subtitle: Text(
                                  vendedor.email,
                                  style: TextStyle(
                                    color: Colors.grey[500],
                                    fontSize: _getResponsiveFontSize(12),
                                  ),
                                ),
                                trailing: Icon(
                                  Icons.arrow_forward_ios_rounded,
                                  color: const Color(0xFF9C27B0),
                                  size: isMobile ? 16 : 18,
                                ),
                                onTap: () async {
                                  // Implementar asignación de vendedor
                                  // await viewModel.asignarVendedor(pedido.id!,
vendedor.id!);
                                  if (mounted) {
                                    Navigator.pop(dialogContext);
                                    ScaffoldMessenger.of(context).showSnackBar(
                                      SnackBar(
                                        content: Row(
                                          children: [
                                            const Icon(Icons.check_circle,
                                                color: Colors.white),
                                            const SizedBox(width: 12),
                                            Expanded(
                                              child: Text(
                                                'Vendedor ${vendedor.nombre}
asignado',
                                              ),
                                            ),
                                          ],
                                        ),
                                        backgroundColor: const Color(0xFF9C27B0),
                                        behavior: SnackBarBehavior.floating,
                                        shape: RoundedRectangleBorder(
                                          borderRadius:
BorderRadius.circular(10),
                                        ),
                                      ),
                                    );
                                  }
                                },
                              ),
                            );
                          },
                        );
                      },
                    ),
```

```
            ),
            Padding(
              padding: EdgeInsets.all(isMobile ? 16 : 20),
              child: SizedBox(
                width: double.infinity,
                child: TextButton(
                  onPressed: () => Navigator.pop(dialogContext),
                  style: TextButton.styleFrom(
                    padding: const EdgeInsets.symmetric(vertical: 14),
                    backgroundColor: const Color(0xFF2A2A2A),
                    shape: RoundedRectangleBorder(
                      borderRadius: BorderRadius.circular(10),
                    ),
                  ),
                  child: const Text(
                    'Cancelar',
                    style: TextStyle(
                      color: Colors.white,
                      fontWeight: FontWeight.w600,
                    ),
                  ),
                ),
              ),
            ),
          ],
        ),
      ),
    ),
  );
}

void _confirmarCancelar(Pedido pedido, PedidoViewModel viewModel) {
  final dialogWidth = _screenWidth > 600 ? 400.0 : _screenWidth * 0.9;
  final isMobile = _isMobile;

  showDialog(
    context: context,
    builder: (dialogContext) => Dialog(
      backgroundColor: Colors.transparent,
      child: Container(
        constraints: BoxConstraints(maxWidth: dialogWidth),
        decoration: BoxDecoration(
          color: const Color(0xFF1A1A1A),
          borderRadius: BorderRadius.circular(20),
          border: Border.all(
            color: const Color(0xFFFF073A).withOpacity(0.3),
          ),
        ),
        child: Column(
          mainAxisSize: MainAxisSize.min,
          children: [
            Container(
              padding: EdgeInsets.all(isMobile ? 16 : 20),
              decoration: const BoxDecoration(
```

```
                    gradient: LinearGradient(
                      colors: [
                        Color(0xFFEF5350),
                        Color(0xFFE57373),
                      ],
                    ),
                    borderRadius: BorderRadius.only(
                      topLeft: Radius.circular(20),
                      topRight: Radius.circular(20),
                    ),
                  ),
                  child: Row(
                    children: [
                      Icon(
                        Icons.warning_rounded,
                        color: Colors.white,
                        size: isMobile ? 24 : 28,
                      ),
                      const SizedBox(width: 12),
                      Text(
                        'Cancelar Pedido',
                        style: TextStyle(
                          color: Colors.white,
                          fontSize: _getResponsiveFontSize(18),
                          fontWeight: FontWeight.bold,
                        ),
                      ),
                    ],
                  ),
                ),
                Padding(
                  padding: EdgeInsets.all(isMobile ? 20 : 24),
                  child: Column(
                    children: [
                      Container(
                        padding: const EdgeInsets.all(16),
                        decoration: BoxDecoration(
                          color: const Color(0xFFFF073A).withOpacity(0.1),
                          borderRadius:  BorderRadius.circular(12),
                          border: Border.all(
                            color: const Color(0xFFFF073A).withOpacity(0.3),
                          ),
                        ),
                        child: Text(
                          '¿Estás seguro de cancelar el pedido
${pedido.numeroPedido}?',
                          style: TextStyle(
                            color: Colors.white,
                            fontSize: _getResponsiveFontSize(15),
                          ),
                          textAlign: TextAlign.center,
                        ),
                      ),
                      const SizedBox(height: 8),
```

```
      Text(
        'Esta acción no se puede deshacer',
        style: TextStyle(
          color: Colors.grey[500],
          fontSize: _getResponsiveFontSize(12),
        ),
      ),
    ],
  ),
),
Padding(
  padding: EdgeInsets.fromLTRB(
    isMobile ? 16 : 20,
    0,
    isMobile ? 16 : 20,
    isMobile ? 16 : 20,
  ),
  child: isMobile
      ? Column(
          children: [
            SizedBox(
              width: double.infinity,
              child: TextButton(
                onPressed: () => Navigator.pop(dialogContext),
                style: TextButton.styleFrom(
                  padding:
                      const EdgeInsets.symmetric(vertical: 14),
                  backgroundColor: const Color(0xFF2A2A2A),
                  shape: RoundedRectangleBorder(
                    borderRadius: BorderRadius.circular(10),
                  ),
                ),
                child: const Text(
                  'No, mantener',
                  style: TextStyle(
                    color: Colors.white,
                    fontWeight: FontWeight.w600,
                  ),
                ),
              ),
            ),
            const SizedBox(height: 12),
            SizedBox(
              width: double.infinity,
              child: Container(
                decoration: BoxDecoration(
                  gradient: const LinearGradient(
                    colors: [
                      Color(0xFFFF073A),
                      Color(0xFFFF073A)
                    ],
                  ),
                  borderRadius: BorderRadius.circular(10),
                ),
```

```
                                child: ElevatedButton(
                                  onPressed: () async {
                                    final success = await viewModel
                                        .cancelarPedido(pedido.id!);
                                    if (mounted) {
                                      Navigator.pop(dialogContext);
                                      if (success) {
                                        await viewModel.cargarPedidos();
                                        if (mounted) {
                                          ScaffoldMessenger.of(context)
                                              .showSnackBar(
                                            SnackBar(
                                              content: const Row(
                                                children: [
                                                  Icon(Icons.info,
                                                      color: Colors.white),
                                                  SizedBox(width: 12),
                                                  Text('Pedido cancelado'),
                                                ],
                                              ),
                                              backgroundColor:
                                                  const Color(0xFFFFA726),
                                              behavior:
SnackBarBehavior.floating,
                                              shape: RoundedRectangleBorder(
                                                borderRadius:
                                                    BorderRadius.circular(10),
                                              ),
                                            ),
                                          );
                                        }
                                      }
                                    }
                                  },
                                  style: ElevatedButton.styleFrom(
                                    backgroundColor: Colors.transparent,
                                    shadowColor: Colors.transparent,
                                    padding:
                                        const EdgeInsets.symmetric(vertical:
14),
                                    shape: RoundedRectangleBorder(
                                      borderRadius: BorderRadius.circular(10),
                                    ),
                                  ),
                                  child: const Text(
                                    'Sí, cancelar',
                                    style: TextStyle(
                                      color: Colors.white,
                                      fontWeight: FontWeight.bold,
                                    ),
                                  ),
                                ),
                              ),
                            ),
                          ),
```

```
        ],
      )
    : Row(
        children: [
          Expanded(
            child: TextButton(
              onPressed: () => Navigator.pop(dialogContext),
              style: TextButton.styleFrom(
                padding:
                    const EdgeInsets.symmetric(vertical: 14),
                backgroundColor: const Color(0xFF2A2A2A),
                shape: RoundedRectangleBorder(
                  borderRadius: BorderRadius.circular(10),
                ),
              ),
              child: const Text(
                'No, mantener',
                style: TextStyle(
                  color: Colors.white,
                  fontWeight: FontWeight.w600,
                ),
              ),
            ),
          ),
          const SizedBox(width: 12),
          Expanded(
            child: Container(
              decoration: BoxDecoration(
                gradient: const LinearGradient(
                  colors: [
                    Color(0xFFFF073A),
                    Color(0xFFFF073A)
                  ],
                ),
                borderRadius: BorderRadius.circular(10),
              ),
              child: ElevatedButton(
                onPressed: () async {
                  final success = await viewModel
                      .cancelarPedido(pedido.id!);
                  if (mounted) {
                    Navigator.pop(dialogContext);
                    if (success) {
                      await viewModel.cargarPedidos();
                      if (mounted) {
                        ScaffoldMessenger.of(context)
                            .showSnackBar(
                          SnackBar(
                            content: const Row(
                              children: [
                                Icon(Icons.info,
                                    color: Colors.white),
                                SizedBox(width: 12),
                                Text('Pedido cancelado'),
```

```
                                                  ],
                                                ),
                                                backgroundColor:
                                                    const Color(0xFFFFA726),
                                                behavior:
SnackBarBehavior.floating,
                                                shape: RoundedRectangleBorder(
                                                  borderRadius:
                                                      BorderRadius.circular(10),
                                                ),
                                              ),
                                            );
                                          }
                                        }
                                      }
                                    },
                                    style: ElevatedButton.styleFrom(
                                      backgroundColor: Colors.transparent,
                                      shadowColor: Colors.transparent,
                                      padding:
                                          const EdgeInsets.symmetric(vertical:
14),
                                      shape: RoundedRectangleBorder(
                                        borderRadius: BorderRadius.circular(10),
                                      ),
                                    ),
                                    child: const Text(
                                      'Sí, cancelar',
                                      style: TextStyle(
                                        color: Colors.white,
                                        fontWeight: FontWeight.bold,
                                      ),
                                    ),
                                  ),
                                ),
                              ),
                            ),
                          ],
                        ),
                      ),
                    ),
                  ),
                ),
              ),
            ],
          ),
        ),
      ),
    );
  }

  // ================================
  //   HELPERS
  // ================================

  String _formatearFecha(DateTime fecha) {
    final meses = [
      'Ene',
      'Feb',
```

```
      'Mar',
      'Abr',
      'May',
      'Jun',
      'Jul',
      'Ago',
      'Sep',
      'Oct',
      'Nov',
      'Dic'
    ];
    return '${fecha.day} ${meses[fecha.month - 1]} ${fecha.year}';
  }

  String _obtenerEstadoString(Pedido pedido) {
    return pedido.estado.nombre;
  }

  PedidoEstado _stringToEstado(String estado) {
    switch (estado.toLowerCase()) {
      case 'pendiente':
      case 'pendiente_pago':
      case 'pendientepago':
        return PedidoEstado.pendientePago;
      case 'pagado':
        return PedidoEstado.pagado;
      case 'procesando':
        return PedidoEstado.procesando;
      case 'enviado':
        return PedidoEstado.enviado;
      case 'entregado':
        return PedidoEstado.entregado;
      case 'cancelado':
        return PedidoEstado.cancelado;
      default:
        return PedidoEstado.pendientePago;
    }
  }
}

Admin_productos_tab.dart
import 'dart:io';
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import '../../viewmodels/producto_viewmodel.dart';
import '../../data/models/producto.dart';
import '../productos/producto_form_dialog.dart';

class AdminProductosTab extends StatefulWidget {
  const AdminProductosTab({Key? key}) : super(key: key);

  @override
  State<AdminProductosTab> createState() => _AdminProductosTabState();
}
```

```
class _AdminProductosTabState extends State<AdminProductosTab> {
  String? _categoriaFiltro;
  final TextEditingController _buscarController = TextEditingController();

  @override
  void initState() {
    super.initState();
    WidgetsBinding.instance.addPostFrameCallback((_) {
      context.read<ProductoViewModel>().cargarProductos();
    });
  }

  @override
  void dispose() {
    _buscarController.dispose();
    super.dispose();
  }

  int _getCrossAxisCount(double width) {
    if (width > 1400) return 5;
    if (width > 1100) return 4;
    if (width > 800) return 3;
    if (width > 500) return 2;
    return 1;
  }

  double _getChildAspectRatio(double width) {
    if (width > 1400) return 0.72;
    if (width > 1100) return 0.70;
    if (width > 800) return 0.68;
    if (width > 500) return 0.65;
    return 0.80;
  }

  Map<String, dynamic> _getResponsiveSizes(double width) {
    if (width > 1100) {
      return {
        'searchPadding': 20.0,
        'gridPadding': 20.0,
        'gridSpacing': 20.0,
        'cardPadding': 14.0,
        'productNameSize': 16.0,
        'priceSize': 19.0,
        'buttonHeight': 44.0,
        'buttonTextSize': 15.0,
        'iconSize': 20.0,
        'chipTextSize': 14.0,
        'chipPaddingH': 16.0,
        'chipPaddingV': 10.0,
      };
    } else if (width > 800) {
      return {
        'searchPadding': 18.0,
```

```
      'gridPadding': 16.0,
      'gridSpacing': 16.0,
      'cardPadding': 12.0,
      'productNameSize': 15.0,
      'priceSize': 18.0,
      'buttonHeight': 42.0,
      'buttonTextSize': 14.0,
      'iconSize': 19.0,
      'chipTextSize': 13.5,
      'chipPaddingH': 14.0,
      'chipPaddingV': 9.0,
    };
  } else if (width > 500) {
    return {
      'searchPadding': 14.0,
      'gridPadding': 12.0,
      'gridSpacing': 12.0,
      'cardPadding': 10.0,
      'productNameSize': 14.0,
      'priceSize': 17.0,
      'buttonHeight': 40.0,
      'buttonTextSize': 13.5,
      'iconSize': 18.0,
      'chipTextSize': 13.0,
      'chipPaddingH': 12.0,
      'chipPaddingV': 8.0,
    };
  } else {
    return {
      'searchPadding': 12.0,
      'gridPadding': 10.0,
      'gridSpacing': 10.0,
      'cardPadding': 10.0,
      'productNameSize': 14.0,
      'priceSize': 17.0,
      'buttonHeight': 44.0,
      'buttonTextSize': 14.0,
      'iconSize': 18.0,
      'chipTextSize': 13.0,
      'chipPaddingH': 12.0,
      'chipPaddingV': 8.0,
    };
  }
}

@override
Widget build(BuildContext context) {
  final width = MediaQuery.of(context).size.width;
  final sizes = _getResponsiveSizes(width);

  return Scaffold(
    backgroundColor: const Color(0xFF0A0A0F),
    body: Column(
      children: [
```

```
        _buildFiltros(sizes),
        Expanded(child: _buildListaProductos(sizes)),
      ],
    ),
    floatingActionButton: FloatingActionButton.extended(
      onPressed: _mostrarFormularioNuevo,
      backgroundColor: const Color(0xFFFF073A),
      icon: const Icon(Icons.add, color: Colors.white),
      label: const Text(
        'Nuevo Producto',
        style: TextStyle(color: Colors.white),
      ),
    ),
  );
}

Widget _buildFiltros(Map<String, dynamic> sizes) {
  return Container(
    padding: EdgeInsets.all(sizes['searchPadding']),
    color: const Color(0xFF0A0A0F),
    child: Column(
      children: [
        // Barra de búsqueda
        Container(
          decoration: BoxDecoration(
            color: Colors.white,
            borderRadius: BorderRadius.circular(30),
          ),
          child: TextField(
            controller: _buscarController,
            style: TextStyle(
              fontSize: sizes['buttonTextSize'],
              color: Colors.black87,
            ),
            decoration: InputDecoration(
              hintText: 'Buscar productos...',
              hintStyle: TextStyle(
                fontSize: sizes['buttonTextSize'],
                color: Colors.grey[600],
              ),
              prefixIcon: Icon(
                Icons.search,
                size: sizes['iconSize'],
                color: Colors.grey[700],
              ),
              border: InputBorder.none,
              contentPadding: EdgeInsets.symmetric(
                horizontal: 20,
                vertical: sizes['searchPadding'] - 2,
              ),
              suffixIcon: _buscarController.text.isNotEmpty
                  ? IconButton(
                      icon: Icon(
                        Icons.clear,
```

```
                        size: sizes['iconSize'],
                        color: Colors.grey[700],
                      ),
                      onPressed: () {
                        _buscarController.clear();
                        setState(() {});
                      },
                    )
                  : null,
            ),
            onChanged: (value) => setState(() {}),
          ),
        ),
        const SizedBox(height: 16),

        // Filtros de categoría
        SingleChildScrollView(
          scrollDirection: Axis.horizontal,
          child: Row(
            children: [
              _buildChipFiltro('Todos', null, sizes),
              _buildChipFiltro('Bebidas', 'Bebidas', sizes),
              _buildChipFiltro('Comida', 'Comida', sizes),
              _buildChipFiltro('Otros', 'Otros', sizes),
              const SizedBox(width: 8),
              ActionChip(
                avatar: Icon(
                  Icons.warning,
                  size: sizes['iconSize'] - 2,
                  color: Colors.white,
                ),
                label: Text(
                  'Bajo Stock',
                  style: TextStyle(
                    color: Colors.white,
                    fontSize: sizes['chipTextSize'],
                  ),
                ),
                backgroundColor: Colors.orange,
                padding: EdgeInsets.symmetric(
                  horizontal: sizes['chipPaddingH'],
                  vertical: sizes['chipPaddingV'],
                ),
                onPressed: () {
                  context.read<ProductoViewModel>().cargarBajoStock();
                },
              ),
            ],
          ),
        ),
      ],
    ),
  );
}
```

```
  Widget _buildChipFiltro(
      String label, String? categoria, Map<String, dynamic> sizes) {
    final isSeleccionado = _categoriaFiltro == categoria;

    return Padding(
      padding: const EdgeInsets.only(right: 10),
      child: FilterChip(
        label: Text(
          label,
          style: TextStyle(fontSize: sizes['chipTextSize']),
        ),
        selected: isSeleccionado,
        backgroundColor: const Color(0xFF0A0A0F),
        selectedColor: const Color(0xFFFF073A),
        checkmarkColor: Colors.white,
        labelStyle: TextStyle(
          color: isSeleccionado ? Colors.white : Colors.grey[400],
          fontWeight: isSeleccionado ? FontWeight.w600 : FontWeight.normal,
        ),
        side: BorderSide(
          color: isSeleccionado ? const Color(0xFFFF073A) :
Colors.grey[700]!,
        ),
        padding: EdgeInsets.symmetric(
          horizontal: sizes['chipPaddingH'],
          vertical: sizes['chipPaddingV'],
        ),
        visualDensity: VisualDensity.comfortable,
        onSelected: (selected) {
          setState(() {
            _categoriaFiltro = selected ? categoria : null;
          });

          if (categoria == null) {
            context.read<ProductoViewModel>().cargarProductos();
          } else {
            context.read<ProductoViewModel>().cargarPorCategoria(categoria);
          }
        },
      ),
    );
  }

  Widget _buildListaProductos(Map<String, dynamic> sizes) {
    final width = MediaQuery.of(context).size.width;

    return Consumer<ProductoViewModel>(
      builder: (context, viewModel, child) {
        if (viewModel.cargando) {
          return const Center(
            child: CircularProgressIndicator(color: Color(0xFFFF073A)),
          );
        }
```

```
if (viewModel.error != null) {
  return Center(
    child: Column(
      mainAxisAlignment: MainAxisAlignment.center,
      children: [
        const Icon(
          Icons.error_outline,
          size: 64,
          color: Color(0xFFFF073A),
        ),
        const SizedBox(height: 16),
        Text(
          viewModel.error!,
          style: const TextStyle(color: Colors.white),
        ),
        const SizedBox(height: 16),
        ElevatedButton(
          onPressed: () => viewModel.cargarProductos(),
          style: ElevatedButton.styleFrom(
            backgroundColor: const Color(0xFFFF073A),
          ),
          child: const Text('Reintentar'),
        ),
      ],
    ),
  );
}

var productos = viewModel.productos;

// Filtrar por búsqueda
if (_buscarController.text.isNotEmpty) {
  final termino = _buscarController.text.toLowerCase();
  productos = productos.where((p) {
    return p.nombre.toLowerCase().contains(termino) ||
        p.codigo.toLowerCase().contains(termino) ||
        (p.categoria.toLowerCase().contains(termino));
  }).toList();
}

if (productos.isEmpty) {
  return Center(
    child: Column(
      mainAxisAlignment: MainAxisAlignment.center,
      children: [
        Icon(
          Icons.inventory_2_outlined,
          size: 64,
          color: Colors.grey[600],
        ),
        const SizedBox(height: 16),
        Text(
          'No hay productos',
```

```
                style: TextStyle(
                  fontSize: 18,
                  color: Colors.grey[400],
                ),
              ),
              const SizedBox(height: 8),
              TextButton.icon(
                onPressed: _mostrarFormularioNuevo,
                icon: const Icon(Icons.add, color: Color(0xFFFF073A)),
                label: const Text(
                  'Agregar Producto',
                  style: TextStyle(color: Color(0xFFFF073A)),
                ),
              ),
            ],
          ),
        );
      }

      return GridView.builder(
        padding: EdgeInsets.all(sizes['gridPadding']),
        gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(
          crossAxisCount: _getCrossAxisCount(width),
          childAspectRatio: _getChildAspectRatio(width),
          crossAxisSpacing: sizes['gridSpacing'],
          mainAxisSpacing: sizes['gridSpacing'],
        ),
        itemCount: productos.length,
        itemBuilder: (context, index) {
          return _buildProductoCard(productos[index], sizes);
        },
      );
    },
  );
}

Widget _buildProductoCard(Producto producto, Map<String, dynamic> sizes) {
  final bajoStock = producto.bajoEnStock;

  return Card(
    elevation: 4,
    color: const Color(0xFF0A0A0F),
    shape: RoundedRectangleBorder(
      borderRadius: BorderRadius.circular(16),
    ),
    child: Column(
      crossAxisAlignment: CrossAxisAlignment.stretch,
      children: [
        // Imagen del producto
        Expanded(
          flex: 3,
          child: Stack(
            children: [
              ClipRRect(
```

```
        borderRadius: const BorderRadius.vertical(
          top: Radius.circular(16),
        ),
        child: producto.imagenUrl != null &&
                producto.imagenUrl!.isNotEmpty
            ? (producto.imagenUrl!.startsWith('http')
                ? Image.network(
                    producto.imagenUrl!,
                    fit: BoxFit.cover,
                    width: double.infinity,
                    errorBuilder: (context, error, stackTrace) {
                      return _buildImagenPlaceholder(sizes);
                    },
                  )
                : Image.file(
                    File(producto.imagenUrl!),
                    fit: BoxFit.cover,
                    width: double.infinity,
                    errorBuilder: (context, error, stackTrace) {
                      return _buildImagenPlaceholder(sizes);
                    },
                  ))
            : _buildImagenPlaceholder(sizes),
      ),
      if (bajoStock)
        Positioned(
          top: 8,
          right: 8,
          child: Container(
            padding: const EdgeInsets.symmetric(
              horizontal: 8,
              vertical: 4,
            ),
            decoration: BoxDecoration(
              color: Colors.orange,
              borderRadius: BorderRadius.circular(12),
            ),
            child: Text(
              'BAJO STOCK',
              style: TextStyle(
                color: Colors.white,
                fontSize: sizes['buttonTextSize'] - 4,
                fontWeight: FontWeight.bold,
              ),
            ),
          ),
        ),
    ],
  ),
),

// Información del producto
Expanded(
  flex: 2,
```

```
              child: Padding(
                padding: EdgeInsets.all(sizes['cardPadding']),
                child: Column(
                  crossAxisAlignment: CrossAxisAlignment.start,
                  mainAxisSize: MainAxisSize.min,
                  children: [
                    Flexible(
                      child: Text(
                        producto.nombre,
                        style: TextStyle(
                          fontSize: sizes['productNameSize'],
                          fontWeight: FontWeight.bold,
                          height: 1.2,
                          color: Colors.white,
                        ),
                        maxLines: 2,
                        overflow: TextOverflow.ellipsis,
                      ),
                    ),
                    const SizedBox(height: 4),

                    Text(
                      'Stock: ${producto.stock} uds',
                      style: TextStyle(
                        fontSize: sizes['buttonTextSize'] - 2,
                        color: Colors.grey[400],
                      ),
                    ),
                    const Spacer(),

                    Padding(
                      padding: const EdgeInsets.only(bottom: 6),
                      child: Text(
                        '\$${producto.precioVenta.toStringAsFixed(2)}',
                        style: TextStyle(
                          fontSize: sizes['priceSize'],
                          fontWeight: FontWeight.bold,
                          color: const Color(0xFFFFD700),
                        ),
                      ),
                    ),

                    // Botones de acción (ADMIN)
                    SizedBox(
                      width: double.infinity,
                      height: sizes['buttonHeight'],
                      child: Row(
                        children: [
                          Expanded(
                            child: ElevatedButton(
                              onPressed: () =>
_mostrarFormularioEditar(producto),
                              style: ElevatedButton.styleFrom(
                                backgroundColor: const Color(0xFFFF073A),
```

```
                              padding: EdgeInsets.symmetric(
                                horizontal: sizes['cardPadding'] - 2,
                              ),
                              shape: RoundedRectangleBorder(
                                borderRadius: BorderRadius.circular(10),
                              ),
                            ),
                            child: Icon(
                              Icons.edit,
                              size: sizes['iconSize'],
                              color: Colors.white,
                            ),
                          ),
                        ),
                        const SizedBox(width: 8),
                        Expanded(
                          child: ElevatedButton(
                            onPressed: () => _confirmarEliminar(producto),
                            style: ElevatedButton.styleFrom(
                              backgroundColor: const Color(0xFF1A1A1A),
                              side: const BorderSide(
                                color: Color(0xFFFF073A),
                                width: 1.5,
                              ),
                              padding: EdgeInsets.symmetric(
                                horizontal: sizes['cardPadding'] - 2,
                              ),
                              shape: RoundedRectangleBorder(
                                borderRadius: BorderRadius.circular(10),
                              ),
                            ),
                            child: Icon(
                              Icons.delete,
                              size:  sizes['iconSize'],
                              color: const Color(0xFFFF073A),
                            ),
                          ),
                        ),
                      ],
                    ),
                  ),
                ],
              ),
            ),
          ),
        ],
      ),
    );
  }

  Widget _buildImagenPlaceholder(Map<String, dynamic> sizes) {
    return Container(
      color: const Color(0xFF1A1A1A),
      width: double.infinity,
```

```
    child: Center(
      child: Icon(
        Icons.restaurant_menu,
        size: sizes['priceSize'] * 3,
        color: Colors.grey[700],
      ),
    ),
  );
}

Future<void> _mostrarFormularioNuevo() async {
  final producto = await showDialog<Producto>(
    context: context,
    builder: (context) => const ProductoFormDialog(),
  );

  if (producto != null && mounted) {
    final viewModel = context.read<ProductoViewModel>();
    final exito = await viewModel.crearProducto(
      producto,
      imagenFile: producto.imagenLocal,
    );

    if (exito && mounted) {
      ScaffoldMessenger.of(context).showSnackBar(
        const SnackBar(
          content: Text('Producto creado exitosamente'),
          backgroundColor: Colors.green,
        ),
      );
    } else if (viewModel.error != null && mounted) {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(
          content: Text(viewModel.error!),
          backgroundColor: Colors.red,
        ),
      );
    }
  }
}

Future<void> _mostrarFormularioEditar(Producto producto) async {
  final productoEditado = await showDialog<Producto>(
    context: context,
    builder: (context) => ProductoFormDialog(producto: producto),
  );

  if (productoEditado != null && mounted) {
    final viewModel = context.read<ProductoViewModel>();
    final exito = await viewModel.actualizarProducto(
      productoEditado,
      nuevaImagen: productoEditado.imagenLocal,
    );
```

```
    if (exito && mounted) {
      ScaffoldMessenger.of(context).showSnackBar(
        const SnackBar(
          content: Text('Producto actualizado exitosamente'),
          backgroundColor: Colors.green,
        ),
      );
    } else if (viewModel.error != null && mounted) {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(
          content: Text(viewModel.error!),
          backgroundColor: Colors.red,
        ),
      );
    }
  }
}

Future<void> _confirmarEliminar(Producto producto) async {
  final confirmar = await showDialog<bool>(
    context: context,
    builder: (context) => AlertDialog(
      backgroundColor: const Color(0xFF1A1A1A),
      title: const Text(
        'Eliminar Producto',
        style: TextStyle(color: Colors.white),
      ),
      content: Text(
        '¿Estás seguro de eliminar "${producto.nombre}"?',
        style: const TextStyle(color: Colors.white70),
      ),
      actions: [
        TextButton(
          onPressed: () => Navigator.pop(context, false),
          child: const Text('Cancelar'),
        ),
        ElevatedButton(
          onPressed: () => Navigator.pop(context, true),
          style: ElevatedButton.styleFrom(
            backgroundColor: const Color(0xFFFF073A),
          ),
          child: const Text('Eliminar'),
        ),
      ],
    ),
  );

  if (confirmar == true && mounted) {
    final viewModel = context.read<ProductoViewModel>();
    final exito = await viewModel.eliminarProducto(producto.id!);

    if (exito && mounted) {
      ScaffoldMessenger.of(context).showSnackBar(
        const SnackBar(
```

```
            content: Text('Producto eliminado'),
            backgroundColor: Colors.green,
          ),
        );
      } else if (viewModel.error != null && mounted) {
        ScaffoldMessenger.of(context).showSnackBar(
          SnackBar(
            content: Text(viewModel.error!),
            backgroundColor: Colors.red,
          ),
        );
      }
    }
  }
}
```

Admin_reportes_tab.dart

```
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import 'package:fl_chart/fl_chart.dart';
import 'package:intl/intl.dart';
import '../../viewmodels/venta_viewmodel.dart';
import '../../viewmodels/producto_viewmodel.dart';
import '../../viewmodels/usuario_viewmodel.dart';
import '../../data/models/venta.dart';

class AdminReportesTab extends StatefulWidget {
  const AdminReportesTab({Key? key}) : super(key: key);

  @override
  State<AdminReportesTab> createState() => _AdminReportesTabState();
}

class _AdminReportesTabState extends State<AdminReportesTab>
    with TickerProviderStateMixin {
  DateTime? _fechaInicio;
  DateTime? _fechaFin;
  AnimationController? _animationController;

  @override
  void initState() {
    super.initState();
    _animationController = AnimationController(
      vsync: this,
      duration: const Duration(milliseconds: 1500),
    );
    _animationController?.forward();

    WidgetsBinding.instance.addPostFrameCallback((_) {
      _cargarDatos();
    });
  }

  @override
```

```
void dispose() {
  _animationController?.dispose();
  super.dispose();
}

void _cargarDatos() {
  if (!mounted) return;

  final  ventaVM  =  context.read<VentaViewModel>();
  final productoVM = context.read<ProductoViewModel>();
  final usuarioVM = context.read<UsuarioViewModel>();

  ventaVM.cargarVentas();
  productoVM.cargarProductos();
  usuarioVM.cargarUsuarios();

  _animationController?.reset();
  _animationController?.forward();
}

Future<void> _seleccionarFecha(bool esInicio) async {
  final fecha = await showDatePicker(
    context: context,
    initialDate: DateTime.now(),
    firstDate: DateTime(2020),
    lastDate: DateTime.now(),
    builder: (context, child) {
      return Theme(
        data: ThemeData.dark().copyWith(
          colorScheme: const ColorScheme.dark(
            primary: Color(0xFFFF073A),
            onPrimary: Colors.white,
            surface: Color(0xFF0A0A0F),
            onSurface: Colors.white,
          ),
          dialogBackgroundColor: const Color(0xFF0A0A0F),
        ),
        child: child!,
      );
    },
  );

  if (fecha != null) {
    setState(() {
      if (esInicio) {
        _fechaInicio = fecha;
      } else {
        _fechaFin = fecha;
      }
    });
  }
}

@override
```

```
Widget build(BuildContext context) {
  final screenWidth = MediaQuery.of(context).size.width;
  final isMobile = screenWidth < 600;

  return Container(
    color: const Color(0xFF0F0F0F),
    child: SingleChildScrollView(
      padding: EdgeInsets.all(isMobile ? 12 : 16),
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          _buildHeader(),
          SizedBox(height: isMobile ? 12 : 16),
          _buildFiltrosFecha(),
          SizedBox(height: isMobile ? 12 : 16),
          _buildResumenGeneral(),
          SizedBox(height: isMobile ? 12 : 16),
          _buildEstadisticasUsuarios(),
          SizedBox(height: isMobile ? 12 : 16),
          _buildEstadisticasVentas(),
          SizedBox(height: isMobile ? 12 : 16),
          _buildGraficoVentasTendencia(),
          SizedBox(height: isMobile ? 12 : 16),
          _buildEstadisticasProductos(),
          SizedBox(height: isMobile ? 12 : 16),
          _buildGraficoTopProductos(),
        ],
      ),
    ),
  );
}

Widget _buildHeader() {
  return LayoutBuilder(
    builder: (context, constraints) {
      final isMobile = constraints.maxWidth < 600;

      return Container(
        padding: EdgeInsets.all(isMobile ? 16 : 20),
        decoration: BoxDecoration(
          gradient: const LinearGradient(
            colors: [Color(0xFFFF073A), Color(0xFFFF8E53)],
            begin: Alignment.topLeft,
            end: Alignment.bottomRight,
          ),
          borderRadius: BorderRadius.circular(16),
          boxShadow: [
            BoxShadow(
              color: const Color(0xFFFF073A).withValues(alpha: 0.3),
              blurRadius: 12,
              offset: const Offset(0, 4),
            ),
          ],
        ),
```

```
      child: Row(
        children: [
          Container(
            padding: EdgeInsets.all(isMobile ? 8 : 12),
            decoration: BoxDecoration(
              color: Colors.white.withValues(alpha: 0.2),
              borderRadius: BorderRadius.circular(12),
            ),
            child: Icon(
              Icons.analytics_rounded,
              color: Colors.white,
              size: isMobile ? 24 : 32,
            ),
          ),
          SizedBox(width: isMobile ? 12 : 16),
          Expanded(
            child: Column(
              crossAxisAlignment: CrossAxisAlignment.start,
              children: [
                Text(
                  'Reportes y Estadísticas',
                  style: TextStyle(
                    color: Colors.white,
                    fontSize: isMobile ? 18 : 22,
                    fontWeight: FontWeight.bold,
                    letterSpacing: 0.5,
                  ),
                ),
                const SizedBox(height: 4),
                Text(
                  'Panel de análisis de negocio',
                  style: TextStyle(
                    color: Colors.white70,
                    fontSize: isMobile ? 12 : 14,
                  ),
                ),
              ],
            ),
          ),
          IconButton(
            onPressed: _cargarDatos,
            icon: Icon(Icons.refresh_rounded,
                color: Colors.white, size: isMobile ? 20 : 24),
            tooltip: 'Actualizar datos',
            style: IconButton.styleFrom(
              backgroundColor: Colors.white.withValues(alpha: 0.2),
              padding: EdgeInsets.all(isMobile ? 8 : 12),
            ),
          ),
        ],
      ),
    );
  },
);
```

```
}

Widget _buildFiltrosFecha() {
  return LayoutBuilder(
    builder: (context, constraints) {
      final isMobile = constraints.maxWidth < 600;

      return Container(
        padding: EdgeInsets.all(isMobile ? 16 : 20),
        decoration: BoxDecoration(
          color: const Color(0xFF1A1A1A),
          borderRadius: BorderRadius.circular(16),
          border: Border.all(color: Colors.white.withValues(alpha: 0.1)),
          boxShadow: [
            BoxShadow(
              color: Colors.black.withValues(alpha: 0.2),
              blurRadius: 10,
              offset: const Offset(0, 4),
            ),
          ],
        ),
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            Row(
              children: [
                Container(
                  padding: const EdgeInsets.all(8),
                  decoration: BoxDecoration(
                    color: const Color(0xFFFF073A).withValues(alpha: 0.15),
                    borderRadius: BorderRadius.circular(8),
                  ),
                  child: Icon(
                    Icons.date_range_rounded,
                    color: const Color(0xFFFF073A),
                    size: isMobile ? 18 : 20,
                  ),
                ),
                SizedBox(width: isMobile ? 8 : 12),
                Flexible(
                  child: Text(
                    'Filtrar por Rango de Fechas',
                    style: TextStyle(
                      fontSize: isMobile ? 16 : 18,
                      fontWeight: FontWeight.bold,
                      color: Colors.white,
                    ),
                  ),
                ),
              ],
            ),
            SizedBox(height: isMobile ? 16 : 20),
            LayoutBuilder(
              builder: (context, constraints) {
```

```
      final isNarrow = constraints.maxWidth < 700;
      if (isNarrow) {
        return Column(
          children: [
            _buildBotonFecha(
              esInicio: true,
              fecha: _fechaInicio,
              label: 'Fecha Inicio',
              icon: Icons.calendar_today_rounded,
            ),
            const SizedBox(height: 12),
            _buildBotonFecha(
              esInicio: false,
              fecha: _fechaFin,
              label: 'Fecha Fin',
              icon: Icons.event_rounded,
            ),
          ],
        );
      }
      return Row(
        children: [
          Expanded(
            child: _buildBotonFecha(
              esInicio: true,
              fecha: _fechaInicio,
              label: 'Fecha Inicio',
              icon: Icons.calendar_today_rounded,
            ),
          ),
          const SizedBox(width: 16),
          Expanded(
            child: _buildBotonFecha(
              esInicio: false,
              fecha: _fechaFin,
              label: 'Fecha Fin',
              icon: Icons.event_rounded,
            ),
          ),
        ],
      );
    },
  ),
  if (_fechaInicio != null || _fechaFin != null)
    Padding(
      padding: const EdgeInsets.only(top: 16),
      child: TextButton.icon(
        onPressed: () {
          setState(() {
            _fechaInicio = null;
            _fechaFin = null;
          });
        },
        icon: const Icon(Icons.clear_rounded, size: 18),
```

```
                    label: const Text('Limpiar filtros'),
                    style: TextButton.styleFrom(
                      foregroundColor: const Color(0xFFFF073A),
                      backgroundColor: const
Color(0xFFFF073A).withOpacity(0.1),
                      padding: const EdgeInsets.symmetric(
                          horizontal: 16, vertical: 12),
                      shape: RoundedRectangleBorder(
                        borderRadius: BorderRadius.circular(10),
                      ),
                    ),
                  ),
                ),
            ],
          ),
        );
      },
    );
  }

  Widget _buildBotonFecha({
    required bool esInicio,
    required DateTime? fecha,
    required String label,
    required IconData icon,
  }) {
    return OutlinedButton.icon(
      onPressed: () => _seleccionarFecha(esInicio),
      icon: Icon(icon, size: 18),
      label: Text(
        fecha == null ? label : '${fecha.day}/${fecha.month}/${fecha.year}',
        style: const TextStyle(fontSize: 14),
      ),
      style: OutlinedButton.styleFrom(
        foregroundColor: fecha == null ? Colors.grey[400] : Colors.white,
        backgroundColor: const Color(0xFF2A2A2A),
        side: BorderSide(
          color: fecha == null
              ? Colors.white.withOpacity(0.1)
              : const Color(0xFFFF073A).withOpacity(0.5),
        ),
        padding: const EdgeInsets.symmetric(horizontal: 16, vertical: 14),
        shape: RoundedRectangleBorder(
          borderRadius: BorderRadius.circular(12),
        ),
      ),
    );
  }

  Widget _buildResumenGeneral() {
    return Consumer3<VentaViewModel, ProductoViewModel, UsuarioViewModel>(
      builder: (context, ventaVM, productoVM, usuarioVM, child) {
        final ventas = ventaVM.ventas;
        final ventasFiltradas = _filtrarVentasPorFecha(ventas);
```

```
        final montoTotal = ventasFiltradas.fold<double>(
          0,
          (sum, venta) => sum + venta.total,
        );

        final totalProductos = productoVM.productos.length;

        return FutureBuilder<Map<String, int>>(
          future: usuarioVM.obtenerEstadisticas(),
          builder: (context, snapshot) {
            final totalUsuarios = snapshot.data?['total'] ?? 0;

            return LayoutBuilder(
              builder: (context, constraints) {
                final screenWidth = constraints.maxWidth;
                final isMobile = screenWidth < 600;
                final isTablet = screenWidth >= 600 && screenWidth < 900;

                if (isMobile) {
                  // Diseño 2x2 para móvil
                  return Column(
                    children: [
                      Row(
                        children: [
                          Expanded(
                            child: _buildKPICard(
                              titulo: 'Ingresos',
                              valor:

'\$${NumberFormat('#,##0').format(montoTotal)}',
                              icon: Icons.attach_money_rounded,
                              color: Color(0xFF00C853),
                              subtitulo: '+12.5%',
                            ),
                          ),
                          const SizedBox(width: 12),
                          Expanded(
                            child: _buildKPICard(
                              titulo: 'Ventas',
                              valor: '${ventasFiltradas.length}',
                              icon: Icons.shopping_bag_rounded,
                              color: Color(0xFF2196F3),
                              subtitulo: '+8.2%',
                            ),
                          ),
                        ],
                      ),
                      const SizedBox(height: 12),
                      Row(
                        children: [
                          Expanded(
                            child: _buildKPICard(
                              titulo: 'Usuarios',
```

```
                            valor: '$totalUsuarios',
                            icon: Icons.people_rounded,
                            color: Color(0xFF9C27B0),
                            subtitulo: '+5.3%',
                          ),
                        ),
                        const SizedBox(width: 12),
                        Expanded(
                          child: _buildKPICard(
                            titulo: 'Productos',
                            valor: '$totalProductos',
                            icon: Icons.inventory_2_rounded,
                            color: const Color(0xFFFF073A),
                            subtitulo: '+3.1%',
                          ),
                        ),
                      ],
                    ),
                  ],
                );
              } else if (isTablet) {
                // Diseño 2x2 para tablet
                return Column(
                  children: [
                    Row(
                      children: [
                        Expanded(
                          child: _buildKPICard(
                            titulo: 'Ingresos Totales',
                            valor:

'\$${NumberFormat('#,##0.00').format(montoTotal)}',
                            icon: Icons.attach_money_rounded,
                            color: Color(0xFF00C853),
                            subtitulo: '+12.5%',
                          ),
                        ),
                        const SizedBox(width: 12),
                        Expanded(
                          child: _buildKPICard(
                            titulo: 'Ventas',
                            valor: '${ventasFiltradas.length}',
                            icon: Icons.shopping_bag_rounded,
                            color: Color(0xFF2196F3),
                            subtitulo: '+8.2%',
                          ),
                        ),
                      ],
                    ),
                    const SizedBox(height: 12),
                    Row(
                      children: [
                        Expanded(
                          child: _buildKPICard(
```

```
                titulo: 'Usuarios',
                valor: '$totalUsuarios',
                icon: Icons.people_rounded,
                color: Color(0xFF9C27B0),
                subtitulo: '+5.3%',
              ),
            ),
            const SizedBox(width: 12),
            Expanded(
              child: _buildKPICard(
                titulo: 'Productos',
                valor: '$totalProductos',
                icon: Icons.inventory_2_rounded,
                color: const Color(0xFFFF073A),
                subtitulo: '+3.1%',
              ),
            ),
          ],
        ),
      ],
    );
  }

  // Diseño horizontal para desktop
  return Row(
    children: [
      Expanded(
        child: _buildKPICard(
          titulo: 'Ingresos Totales',
          valor:

'\$${NumberFormat('#,##0.00').format(montoTotal)}',
          icon: Icons.attach_money_rounded,
          color: Color(0xFF00C853),
          subtitulo: '+12.5%',
        ),
      ),
      const SizedBox(width: 12),
      Expanded(
        child: _buildKPICard(
          titulo: 'Ventas',
          valor: '${ventasFiltradas.length}',
          icon: Icons.shopping_bag_rounded,
          color: Color(0xFF2196F3),
          subtitulo: '+8.2%',
        ),
      ),
      const SizedBox(width: 12),
      Expanded(
        child: _buildKPICard(
          titulo: 'Usuarios',
          valor: '$totalUsuarios',
          icon: Icons.people_rounded,
          color: Color(0xFF9C27B0),
```

```
                    subtitulo: '+5.3%',
                  ),
                ),
                const SizedBox(width: 12),
                Expanded(
                  child: _buildKPICard(
                    titulo: 'Productos',
                    valor: '$totalProductos',
                    icon: Icons.inventory_2_rounded,
                    color: const Color(0xFFFF073A),
                    subtitulo: '+3.1%',
                  ),
                ),
              ],
            );
          },
        );
      },
    );
  }

  Widget _buildKPICard({
    required String titulo,
    required String valor,
    required IconData icon,
    required Color color,
    String? subtitulo,
  }) {
    final animation = _animationController != null
        ? CurvedAnimation(
            parent: _animationController!,
            curve: Curves.easeOutCubic,
          )
        : null;

    return AnimatedBuilder(
      animation: animation ?? kAlwaysCompleteAnimation,
      builder: (context, child) {
        final animValue = animation?.value ?? 1.0;
        return Transform.scale(
          scale: 0.8 + (0.2 * animValue),
          child: Opacity(
            opacity: animValue,
            child: child,
          ),
        );
      },
      child: LayoutBuilder(
        builder: (context, constraints) {
          final isMobile = constraints.maxWidth < 150;

          return Container(
```

```
            padding: EdgeInsets.all(isMobile ? 12 : 20),
            decoration: BoxDecoration(
              gradient: LinearGradient(
                colors: [
                  color.withValues(alpha: 0.7),
                  color.withValues(alpha: 0.9)
                ],
                begin: Alignment.topLeft,
                end: Alignment.bottomRight,
              ),
              borderRadius: BorderRadius.circular(16),
              boxShadow: [
                BoxShadow(
                  color: color.withValues(alpha: 0.3),
                  blurRadius: 10,
                  offset: const Offset(0, 4),
                ),
              ],
            ),
            child: Column(
              crossAxisAlignment: CrossAxisAlignment.start,
              children: [
                Row(
                  mainAxisAlignment: MainAxisAlignment.spaceBetween,
                  children: [
                    Icon(icon, color: Colors.white, size: isMobile ? 20 :
28),
                    if (subtitulo != null)
                      Container(
                        padding: EdgeInsets.symmetric(
                          horizontal: isMobile ? 6 : 8,
                          vertical: 4,
                        ),
                        decoration: BoxDecoration(
                          color: Colors.white.withValues(alpha: 0.3),
                          borderRadius: BorderRadius.circular(8),
                        ),
                        child: Text(
                          subtitulo,
                          style: TextStyle(
                            color: Colors.white,
                            fontSize: isMobile ? 10 : 12,
                            fontWeight: FontWeight.bold,
                          ),
                        ),
                      ),
                  ],
                ),
                SizedBox(height: isMobile ? 8 : 16),
                Text(
                  titulo,
                  style: TextStyle(
                    color: Colors.white.withValues(alpha: 0.9),
                    fontSize: isMobile ? 11 : 14,
```

```
                  fontWeight: FontWeight.w500,
                ),
                maxLines: 2,
                overflow: TextOverflow.ellipsis,
              ),
              const SizedBox(height: 4),
              FittedBox(
                fit: BoxFit.scaleDown,
                alignment: Alignment.centerLeft,
                child: Text(
                  valor,
                  style: TextStyle(
                    color: Colors.white,
                    fontSize: isMobile ? 18 : 24,
                    fontWeight: FontWeight.bold,
                  ),
                ),
              ),
            ],
          ),
        );
      },
    ),
  );
}

Widget _buildEstadisticasUsuarios() {
  return Consumer<UsuarioViewModel>(
    builder: (context, vm, child) {
      return FutureBuilder<Map<String, int>>(
        future: vm.obtenerEstadisticas(),
        builder: (context, snapshot) {
          if (!snapshot.hasData) {
            return _buildLoadingCard();
          }

          final stats = snapshot.data!;
          final total = stats['total'] ?? 0;

          return FadeTransition(
            opacity:
                _animationController ?? const AlwaysStoppedAnimation(1.0),
            child: LayoutBuilder(
              builder: (context, constraints) {
                final isMobile = constraints.maxWidth < 600;

                return Container(
                  padding: EdgeInsets.all(isMobile ? 16 : 20),
                  decoration: BoxDecoration(
                    color: const Color(0xFF1A1A1A),
                    borderRadius: BorderRadius.circular(16),
                    border: Border.all(
                        color: Colors.white.withValues(alpha: 0.1)),
                    boxShadow: [
```

```
                    BoxShadow(
                      color: Colors.black.withValues(alpha: 0.2),
                      blurRadius: 10,
                      offset: const Offset(0, 4),
                    ),
                  ],
                ),
                child: Column(
                  crossAxisAlignment: CrossAxisAlignment.start,
                  children: [
                    Row(
                      children: [
                        Container(
                          padding: const EdgeInsets.all(8),
                          decoration: BoxDecoration(
                            color: const Color(0xFFFF073A)
                                .withValues(alpha: 0.15),
                            borderRadius: BorderRadius.circular(8),
                          ),
                          child: Icon(
                            Icons.people_rounded,
                            color: const Color(0xFFFF073A),
                            size: isMobile ? 18 : 20,
                          ),
                        ),
                        SizedBox(width: isMobile ? 8 : 12),
                        Flexible(
                          child: Text(
                            'Estadísticas de Usuarios',
                            style: TextStyle(
                              fontSize: isMobile ? 16 : 18,
                              fontWeight: FontWeight.bold,
                              color: Colors.white,
                            ),
                          ),
                        ),
                      ],
                    ),
                    SizedBox(height: isMobile ? 16 : 24),
                    isMobile
                        ? Column(
                            children: [
                              _buildStatRow(
                                'Total de Usuarios',
                                stats['total']?.toString() ?? '0',
                                Icons.people,
                                Colors.blue,
                              ),
                              Divider(
                                  color:
                                      Colors.white.withValues(alpha:
0.1),
                                  height: 24),
                              _buildStatRow(
```

```
                                  'Administradores',
                                  stats['admins']?.toString() ?? '0',
                                  Icons.admin_panel_settings,
                                  Colors.purple,
                                ),
                                Divider(
                                    color:
                                        Colors.white.withValues(alpha:
0.1),
                                    height: 24),
                                _buildStatRow(
                                  'Vendedores',
                                  stats['vendedores']?.toString() ?? '0',
                                  Icons.store,
                                  Colors.orange,
                                ),
                                Divider(
                                    color:
                                        Colors.white.withValues(alpha:
0.1),
                                    height: 24),
                                _buildStatRow(
                                  'Clientes',
                                  stats['clientes']?.toString() ?? '0',
                                  Icons.person,
                                  Colors.green,
                                ),
                                const SizedBox(height: 24),
                                SizedBox(
                                  height: 220,
                                  child: _buildUsuariosChart(stats, total),
                                ),
                              ],
                            )
                          : Row(
                              children: [
                                Expanded(
                                  flex: 2,
                                  child: Column(
                                    children: [
                                      _buildStatRow(
                                        'Total de Usuarios',
                                        stats['total']?.toString() ?? '0',
                                        Icons.people,
                                        Colors.blue,
                                      ),
                                      Divider(
                                          color: Colors.white
                                              .withValues(alpha: 0.1),
                                          height: 24),
                                      _buildStatRow(
                                        'Administradores',
                                        stats['admins']?.toString() ?? '0',
                                        Icons.admin_panel_settings,
```

```
                                        Colors.purple,
                                      ),
                                      Divider(
                                          color: Colors.white
                                              .withValues(alpha: 0.1),
                                          height: 24),
                                      _buildStatRow(
                                        'Vendedores',
                                        stats['vendedores']?.toString() ??
                                            '0',
                                        Icons.store,
                                        Colors.orange,
                                      ),
                                      Divider(
                                          color: Colors.white
                                              .withValues(alpha: 0.1),
                                          height: 24),
                                      _buildStatRow(
                                        'Clientes',
                                        stats['clientes']?.toString() ??
'0',
                                        Icons.person,
                                        Colors.green,
                                      ),
                                    ],
                                  ),
                                ),
                                const SizedBox(width: 24),
                                Expanded(
                                  child: SizedBox(
                                    height: 220,
                                    child: _buildUsuariosChart(stats,
total),
                                  ),
                                ),
                              ],
                            ),
                          ],
                        );
                      },
                    ),
                  );
                },
              );
            },
          );
        }

  Widget _buildUsuariosChart(Map<String, int> stats, int total) {
    if (total == 0) {
      return const Center(
        child: Text(
          'Sin datos',
```

```
          style: TextStyle(color: Colors.white54),
        ),
      );
    }

    return PieChart(
      PieChartData(
        sectionsSpace: 2,
        centerSpaceRadius: 50,
        sections: [
          PieChartSectionData(
            color: Colors.purple,
            value: (stats['admins'] ?? 0).toDouble(),
            title:
                '${((stats['admins'] ?? 0) / total *
100).toStringAsFixed(1)}%',
            radius: 60,
            titleStyle: const TextStyle(
              fontSize: 12,
              fontWeight: FontWeight.bold,
              color: Colors.white,
            ),
          ),
          PieChartSectionData(
            color: Colors.orange,
            value: (stats['vendedores'] ?? 0).toDouble(),
            title:
                '${((stats['vendedores'] ?? 0) / total *
100).toStringAsFixed(1)}%',
            radius: 60,
            titleStyle: const TextStyle(
              fontSize: 12,
              fontWeight: FontWeight.bold,
              color: Colors.white,
            ),
          ),
          PieChartSectionData(
            color: Colors.green,
            value: (stats['clientes'] ?? 0).toDouble(),
            title:
                '${((stats['clientes'] ?? 0) / total *
100).toStringAsFixed(1)}%',
            radius: 60,
            titleStyle: const TextStyle(
              fontSize: 12,
              fontWeight: FontWeight.bold,
              color: Colors.white,
            ),
          ),
        ],
      ),
    );
  }
```

```
  Widget _buildEstadisticasVentas() {
    return Consumer<VentaViewModel>(
      builder: (context, vm, child) {
        final ventas = vm.ventas;
        final ventasFiltradas = _filtrarVentasPorFecha(ventas);

        final totalVentas = ventasFiltradas.length;
        final montoTotal = ventasFiltradas.fold<double>(
          0,
          (sum, venta) => sum + venta.total,
        );
        final promedioVenta = totalVentas > 0 ? montoTotal / totalVentas : 0;

        final ventaMaxima = ventasFiltradas.isEmpty
            ? 0.0
            : ventasFiltradas
                .map((v) => v.total)
                .reduce((a, b) => a > b ? a : b);

        return FadeTransition(
          opacity: _animationController ?? const AlwaysStoppedAnimation(1.0),
          child: LayoutBuilder(
            builder: (context, constraints) {
              final isMobile = constraints.maxWidth < 600;

              return Container(
                padding: EdgeInsets.all(isMobile ? 16 : 20),
                decoration: BoxDecoration(
                  color: const Color(0xFF0A0A0F),
                  borderRadius: BorderRadius.circular(16),
                  border:
                      Border.all(color: Colors.white.withValues(alpha: 0.1)),
                  boxShadow: [
                    BoxShadow(
                      color: Colors.black.withValues(alpha: 0.2),
                      blurRadius: 10,
                      offset: const Offset(0, 4),
                    ),
                  ],
                ),
                child: Column(
                  crossAxisAlignment: CrossAxisAlignment.start,
                  children: [
                    Row(
                      children: [
                        Container(
                          padding: const EdgeInsets.all(8),
                          decoration: BoxDecoration(
                            color:
                                const Color(0xFFFF073A).withValues(alpha:
0.15),
                            borderRadius: BorderRadius.circular(8),
                          ),
                          child: Icon(
```

```
                        Icons.shopping_cart_rounded,
                        color: const Color(0xFFFF073A),
                        size: isMobile ? 18 : 20,
                      ),
                    ),
                    SizedBox(width: isMobile ? 8 : 12),
                    Flexible(
                      child: Text(
                        'Estadísticas de Ventas',
                        style: TextStyle(
                          fontSize: isMobile ? 16 : 18,
                          fontWeight: FontWeight.bold,
                          color: Colors.white,
                        ),
                      ),
                    ),
                  ],
                ),
                SizedBox(height: isMobile ? 16 : 20),
                LayoutBuilder(
                  builder: (context, constraints) {
                    final fila1 = Row(
                      children: [
                        Expanded(
                          child: _buildStatCard(
                            'Total de Ventas',
                            totalVentas.toString(),
                            Icons.shopping_cart,
                            Colors.blue,
                          ),
                        ),
                        const SizedBox(width: 12),
                        Expanded(
                          child: _buildStatCard(
                            'Monto Total',
'\$${NumberFormat('#,##0.00').format(montoTotal)}',
                            Icons.attach_money,
                            Colors.green,
                          ),
                        ),
                      ],
                    );
                    final fila2 = Row(
                      children: [
                        Expanded(
                          child: _buildStatCard(
                            'Promedio por Venta',
'\$${NumberFormat('#,##0.00').format(promedioVenta)}',
                            Icons.analytics,
                            Colors.orange,
                          ),
                        ),
```

```
                              const SizedBox(width: 12),
                              Expanded(
                                child: _buildStatCard(
                                  'Venta Máxima',

'\$${NumberFormat('#,##0.00').format(ventaMaxima)}',
                                    Icons.trending_up,
                                    Colors.purple,
                                  ),
                                ),
                              ],
                            );
                            return Column(
                              children: [
                                fila1,
                                const SizedBox(height: 12),
                                fila2,
                              ],
                            );
                          },
                        ),
                      ],
                    ),
                  );
                },
              ),
            );
          },
        ),
      );
    },
  );
}

Widget _buildStatCard(
    String label, String value, IconData icon, Color color) {
  return LayoutBuilder(
    builder: (context, constraints) {
      final isSmall = constraints.maxWidth < 150;

      return Container(
        padding: EdgeInsets.all(isSmall ? 12 : 16),
        decoration: BoxDecoration(
          color: color.withValues(alpha: 0.1),
          borderRadius: BorderRadius.circular(12),
          border: Border.all(color: color.withValues(alpha: 0.3)),
        ),
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            Row(
              children: [
                Icon(icon, color: color, size: isSmall ? 16 : 20),
                const Spacer(),
                Container(
                  padding: EdgeInsets.all(isSmall ? 4 : 6),
                  decoration: BoxDecoration(
```

```
                    color: color.withValues(alpha: 0.2),
                    borderRadius: BorderRadius.circular(6),
                  ),
                  child: Icon(Icons.arrow_upward,
                      color: color, size: isSmall ? 10 : 14),
                ),
              ],
            ),
            SizedBox(height: isSmall ? 8 : 12),
            Text(
              label,
              style: TextStyle(
                fontSize: isSmall ? 11 : 13,
                color: Colors.white70,
              ),
              maxLines: 2,
              overflow: TextOverflow.ellipsis,
            ),
            const SizedBox(height: 4),
            FittedBox(
              fit: BoxFit.scaleDown,
              alignment: Alignment.centerLeft,
              child: Text(
                value,
                style: TextStyle(
                  fontSize: isSmall ? 14 : 18,
                  fontWeight: FontWeight.bold,
                  color: color,
                ),
              ),
            ),
          ],
        ),
      );
    },
  );
}

Widget _buildGraficoVentasTendencia() {
  return Consumer<VentaViewModel>(
    builder: (context, vm, child) {
      final ventas = vm.ventas;
      final ventasFiltradas = _filtrarVentasPorFecha(ventas);

      // Agrupar ventas por día
      final Map<String, double> ventasPorDia = {};
      final Map<String, int> cantidadPorDia = {};

      for (var venta in ventasFiltradas) {
        final fecha = DateFormat('dd/MM').format(venta.fecha);
        ventasPorDia[fecha] = (ventasPorDia[fecha] ?? 0) + venta.total;
        cantidadPorDia[fecha] = (cantidadPorDia[fecha] ?? 0) + 1;
      }
```

```
final fechasOrdenadas = ventasPorDia.keys.toList()
  ..sort((a, b) {
    final partsA = a.split('/');
    final partsB = b.split('/');
    final diaA = int.parse(partsA[0]);
    final diaB = int.parse(partsB[0]);
    return diaA.compareTo(diaB);
  });

if (fechasOrdenadas.isEmpty) {
  return const SizedBox.shrink();
}

return FadeTransition(
  opacity: _animationController ?? const AlwaysStoppedAnimation(1.0),
  child: LayoutBuilder(
    builder: (context, constraints) {
      final isMobile = constraints.maxWidth < 600;
      final chartHeight = isMobile ? 200.0 : 250.0;

      return Container(
        padding: EdgeInsets.all(isMobile ? 16 : 20),
        decoration: BoxDecoration(
          color: const Color(0xFF1A1A1A),
          borderRadius: BorderRadius.circular(16),
          border:
              Border.all(color: Colors.white.withValues(alpha: 0.1)),
          boxShadow: [
            BoxShadow(
              color: Colors.black.withOpacity(0.2),
              blurRadius: 10,
              offset: const Offset(0, 4),
            ),
          ],
        ),
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            Row(
              children: [
                Container(
                  padding: const EdgeInsets.all(8),
                  decoration: BoxDecoration(
                    color: const Color(0xFFFF073A).withOpacity(0.15),
                    borderRadius: BorderRadius.circular(8),
                  ),
                  child: Icon(
                    Icons.show_chart_rounded,
                    color: const Color(0xFFFF073A),
                    size: isMobile ? 18 : 20,
                  ),
                ),
                SizedBox(width: isMobile ? 8 : 12),
                Flexible(
```

```
                        child: Text(
                          'Tendencia de Ventas',
                          style: TextStyle(
                            fontSize: isMobile ? 16 : 18,
                            fontWeight: FontWeight.bold,
                            color: Colors.white,
                          ),
                        ),
                      ),
                    ],
                  ),
                  SizedBox(height: isMobile ? 16 : 24),
                  SizedBox(
                    height: chartHeight,
                    child: LineChart(
                      LineChartData(
                        gridData: FlGridData(
                          show: true,
                          drawVerticalLine: false,
                          horizontalInterval: 1,
                          getDrawingHorizontalLine: (value) {
                            return FlLine(
                              color: Colors.white.withValues(alpha: 0.1),
                              strokeWidth: 1,
                            );
                          },
                        ),
                        titlesData: FlTitlesData(
                          show: true,
                          rightTitles: const AxisTitles(
                            sideTitles: SideTitles(showTitles: false),
                          ),
                          topTitles: const AxisTitles(
                            sideTitles: SideTitles(showTitles: false),
                          ),
                          bottomTitles: AxisTitles(
                            sideTitles: SideTitles(
                              showTitles: true,
                              reservedSize: 30,
                              interval: 1,
                              getTitlesWidget:
                                  (double value, TitleMeta meta) {
                                if (value.toInt() >= 0 &&
                                    value.toInt() < fechasOrdenadas.length)
{
                                  return Padding(
                                    padding: const EdgeInsets.only(top:
8.0),
                                    child: Text(
                                      fechasOrdenadas[value.toInt()],
                                      style: TextStyle(
                                        color: Colors.white54,
                                        fontSize: isMobile ? 8 : 10,
                                      ),
```

```
                                    ),
                                  );
                              }
                              return const Text('');
                            },
                          ),
                        ),
                        leftTitles: AxisTitles(
                          sideTitles: SideTitles(
                            showTitles: true,
                            interval: null,
                            reservedSize: isMobile ? 35 : 42,
                            getTitlesWidget:
                                (double value, TitleMeta meta) {
                              return Text(
                                '\$${value.toInt()}',
                                style: TextStyle(
                                  color: Colors.white54,
                                  fontSize: isMobile ? 8 : 10,
                                ),
                              );
                            },
                          ),
                        ),
                      ),
                      borderData: FlBorderData(
                        show: true,
                        border: Border.all(
                            color: Colors.white.withValues(alpha: 0.1)),
                      ),
                      minX: 0,
                      maxX: (fechasOrdenadas.length - 1).toDouble(),
                      minY: 0,
                      maxY: ventasPorDia.values
                              .reduce((a, b) => a > b ? a : b) *
                          1.2,
                      lineBarsData: [
                        LineChartBarData(
                          spots:
                              fechasOrdenadas.asMap().entries.map((entry)
{
                            return FlSpot(
                              entry.key.toDouble(),
                              ventasPorDia[entry.value]!,
                            );
                          }).toList(),
                          isCurved: true,
                          gradient: const LinearGradient(
                            colors: [Color(0xFFFF073A),
Color(0xFFFF8E53)],
                          ),
                          barWidth: isMobile ? 2 : 3,
                          isStrokeCapRound: true,
                          dotData: FlDotData(
```

```
                                show: true,
                                getDotPainter: (spot, percent, barData,
index) {
                                  return FlDotCirclePainter(
                                    radius: isMobile ? 3 : 4,
                                    color: Colors.white,
                                    strokeWidth: 2,
                                    strokeColor: const Color(0xFFFF073A),
                                  );
                                },
                              ),
                              belowBarData: BarAreaData(
                                show: true,
                                gradient: LinearGradient(
                                  colors: [
                                    const Color(0xFFFF6B6B)
                                        .withValues(alpha: 0.3),
                                    const Color(0xFFFF8E53)
                                        .withValues(alpha: 0.1),
                                  ],
                                  begin: Alignment.topCenter,
                                  end: Alignment.bottomCenter,
                                ),
                              ),
                            ),
                          ],
                          lineTouchData: LineTouchData(
                            touchTooltipData: LineTouchTooltipData(
                              tooltipBgColor: const Color(0xFF0A0A0F),
                              tooltipRoundedRadius: 8,
                              getTooltipItems:
                                  (List<LineBarSpot> touchedBarSpots) {
                                return touchedBarSpots.map((barSpot) {
                                  final fecha =
                                      fechasOrdenadas[barSpot.x.toInt()];
                                  final cantidad = cantidadPorDia[fecha] ??
0;
                                  return LineTooltipItem(

'$fecha\n\$${barSpot.y.toStringAsFixed(2)}\n$cantidad  ventas',
                                    TextStyle(
                                      color: Colors.white,
                                      fontWeight: FontWeight.bold,
                                      fontSize: isMobile ? 10 : 12,
                                    ),
                                  );
                                }).toList();
                              },
                            ),
                          ),
                        ),
                      ),
                    ),
                  ),
                ],
```

```
              ),
            );
          },
        ),
      );
    },
  );
}

Widget _buildEstadisticasProductos() {
  return Consumer<ProductoViewModel>(
    builder: (context, vm, child) {
      final productos = vm.productos;
      final totalProductos = productos.length;
      final productosActivos = productos.where((p) => p.activo).length;
      final stockTotal = productos.fold<int>(
        0,
        (sum, producto) => sum + producto.stock,
      );
      final productosLowStock = productos.where((p) => p.stock <
10).length;
      final productosAgotados = productos.where((p) => p.stock ==
0).length;

      final valorInventario = productos.fold<double>(
        0,
        (sum, producto) => sum + (producto.precioCompra * producto.stock),
      );

      return FadeTransition(
        opacity: _animationController ?? const AlwaysStoppedAnimation(1.0),
        child: LayoutBuilder(
          builder: (context, constraints) {
            final isMobile = constraints.maxWidth < 600;

            return Container(
              padding: EdgeInsets.all(isMobile ? 16 : 20),
              decoration: BoxDecoration(
                color: const Color(0xFF1A1A1A),
                borderRadius: BorderRadius.circular(16),
                border:
                    Border.all(color: Colors.white.withValues(alpha: 0.1)),
                boxShadow: [
                  BoxShadow(
                    color: Colors.black.withValues(alpha: 0.2),
                    blurRadius: 10,
                    offset: const Offset(0, 4),
                  ),
                ],
              ),
              child: Column(
                crossAxisAlignment: CrossAxisAlignment.start,
                children: [
                  Row(
```

```
                  children: [
                    Container(
                      padding: const EdgeInsets.all(8),
                      decoration: BoxDecoration(
                        color:
                            const Color(0xFFFF073A).withValues(alpha:
0.15),
                        borderRadius: BorderRadius.circular(8),
                      ),
                      child: Icon(
                        Icons.inventory_2_rounded,
                        color: const Color(0xFFFF073A),
                        size: isMobile ? 18 : 20,
                      ),
                    ),
                    SizedBox(width: isMobile ? 8 : 12),
                    Flexible(
                      child: Text(
                        'Estadísticas de Productos',
                        style: TextStyle(
                          fontSize: isMobile ? 16 : 18,
                          fontWeight: FontWeight.bold,
                          color: Colors.white,
                        ),
                      ),
                    ),
                  ],
                ),
                SizedBox(height: isMobile ? 16 : 20),
                Row(
                  children: [
                    Expanded(
                      child: _buildStatCard(
                        'Total de Productos',
                        totalProductos.toString(),
                        Icons.inventory,
                        Colors.blue,
                      ),
                    ),
                    const SizedBox(width: 12),
                    Expanded(
                      child: _buildStatCard(
                        'Productos Activos',
                        productosActivos.toString(),
                        Icons.check_circle,
                        Colors.green,
                      ),
                    ),
                  ],
                ),
                const SizedBox(height: 12),
                Row(
                  children: [
                    Expanded(
```

```
                          child: _buildStatCard(
                            'Stock Total',
                            stockTotal.toString(),
                            Icons.storage,
                            Colors.orange,
                          ),
                        ),
                        const SizedBox(width: 12),
                        Expanded(
                          child: _buildStatCard(
                            'Stock Bajo',
                            productosLowStock.toString(),
                            Icons.warning,
                            Colors.red,
                          ),
                        ),
                      ],
                    ),
                    const SizedBox(height: 12),
                    Row(
                      children: [
                        Expanded(
                          child: _buildStatCard(
                            'Productos Agotados',
                            productosAgotados.toString(),
                            Icons.error_outline,
                            Colors.deepOrange,
                          ),
                        ),
                        const SizedBox(width: 12),
                        Expanded(
                          child: _buildStatCard(
                            'Valor Inventario',
'\$${NumberFormat('#,##0.00').format(valorInventario)}',
                            Icons.monetization_on,
                            Colors.teal,
                          ),
                        ),
                      ],
                    ),
                  ],
                ),
              );
            },
          ),
        );
      },
    );
  }

  Widget _buildGraficoTopProductos() {
    return Consumer2<VentaViewModel, ProductoViewModel>(
      builder: (context, ventaVM, productoVM, child) {
```

```
        final ventas = ventaVM.ventas;
        final ventasFiltradas = _filtrarVentasPorFecha(ventas);

        // Contar productos vendidos
        final Map<int, int> productosVendidos = {};
        for (var venta in ventasFiltradas) {
          for (var detalle in venta.detalles) {
            productosVendidos[detalle.productoId] =
                (productosVendidos[detalle.productoId] ?? 0) +
detalle.cantidad;
          }
        }

        // Ordenar y tomar top 5
        final topProductos = productosVendidos.entries.toList()
          ..sort((a, b) => b.value.compareTo(a.value));
        final top5 = topProductos.take(5).toList();

        if (top5.isEmpty) {
          return const SizedBox.shrink();
        }

        return FadeTransition(
          opacity: _animationController ?? const AlwaysStoppedAnimation(1.0),
          child: LayoutBuilder(
            builder: (context, constraints) {
              final isMobile = constraints.maxWidth < 600;
              final chartHeight = isMobile ? 250.0 : 300.0;

              return Container(
                padding: EdgeInsets.all(isMobile ? 16 : 20),
                decoration: BoxDecoration(
                  color: const Color(0xFF0A0A0F),
                  borderRadius: BorderRadius.circular(16),
                  border: Border.all(color: Colors.white.withOpacity(0.1)),
                  boxShadow: [
                    BoxShadow(
                      color: Colors.black.withOpacity(0.2),
                      blurRadius: 10,
                      offset: const Offset(0, 4),
                    ),
                  ],
                ),
                child: Column(
                  crossAxisAlignment: CrossAxisAlignment.start,
                  children: [
                    Row(
                      children: [
                        Container(
                          padding: const EdgeInsets.all(8),
                          decoration: BoxDecoration(
                            color: const Color(0xFFFF073A).withOpacity(0.15),
                            borderRadius: BorderRadius.circular(8),
                          ),
```

```
        child: Icon(
          Icons.trending_up_rounded,
          color: const Color(0xFFFF073A),
          size: isMobile ? 18 : 20,
        ),
      ),
      SizedBox(width: isMobile ? 8 : 12),
      Flexible(
        child: Text(
          'Top 5 Productos Más Vendidos',
          style: TextStyle(
            fontSize: isMobile ? 16 : 18,
            fontWeight: FontWeight.bold,
            color: Colors.white,
          ),
        ),
      ),
    ],
  ),
  SizedBox(height: isMobile ? 16 : 24),
  SizedBox(
    height: chartHeight,
    child: BarChart(
      BarChartData(
        alignment: BarChartAlignment.spaceAround,
        maxY: top5.first.value.toDouble() * 1.2,
        barTouchData: BarTouchData(
          touchTooltipData: BarTouchTooltipData(
            tooltipBgColor: const Color(0xFF0A0A0F),
            tooltipRoundedRadius:  8,
            getTooltipItem:
                (group, groupIndex, rod, rodIndex) {
              final id = top5[group.x.toInt()].key;
              String nombre;
              if (productoVM.productos.isEmpty) {
                nombre = 'ID $id';
              } else {
                final idx = productoVM.productos
                    .indexWhere((p) => p.id == id);
                nombre = idx == -1
                    ? 'ID $id'
                    : productoVM.productos[idx].nombre;
              }
              return BarTooltipItem(
                '$nombre\n${rod.toY.toInt()} unidades',
                TextStyle(
                  color: Colors.white,
                  fontWeight: FontWeight.bold,
                  fontSize: isMobile ? 10 : 12,
                ),
              );
            },
          ),
        ),
```

```
                titlesData: FlTitlesData(
                  show: true,
                  rightTitles: const AxisTitles(
                    sideTitles: SideTitles(showTitles: false),
                  ),
                  topTitles: const AxisTitles(
                    sideTitles: SideTitles(showTitles: false),
                  ),
                  bottomTitles: AxisTitles(
                    sideTitles: SideTitles(
                      showTitles: true,
                      reservedSize: isMobile ? 50 : 60,
                      getTitlesWidget:
                          (double value, TitleMeta meta) {
                        if (value.toInt() >= 0 &&
                            value.toInt() < top5.length) {
                          final id = top5[value.toInt()].key;
                          String nombre;
                          if (productoVM.productos.isEmpty) {
                            nombre = 'ID $id';
                          } else {
                            final idx = productoVM.productos
                                .indexWhere((p) => p.id == id);
                            nombre = idx == -1
                                ? 'ID $id'
                                : productoVM.productos[idx].nombre;
                          }
                          return Padding(
                            padding: const EdgeInsets.only(top:
8.0),
                            child: Transform.rotate(
                              angle: -0.5,
                              child: Text(
                                nombre.length > (isMobile ? 10 :
15)
                                    ? '${nombre.substring(0,
isMobile ? 10 : 15)}...'
                                    : nombre,
                                style: TextStyle(
                                  color: Colors.white54,
                                  fontSize: isMobile ? 8 : 10,
                                ),
                              ),
                            ),
                          );
                        }
                        return const Text('');
                      },
                    ),
                  ),
                  leftTitles: AxisTitles(
                    sideTitles: SideTitles(
                      showTitles: true,
                      reservedSize: isMobile ? 35 : 40,
```

```
      getTitlesWidget:
          (double value, TitleMeta meta) {
        return Text(
          value.toInt().toString(),
          style: TextStyle(
            color: Colors.white54,
            fontSize: isMobile ? 8 : 10,
          ),
        );
      },
    ),
  ),
),
borderData: FlBorderData(
  show: true,
  border: Border.all(
      color: Colors.white.withValues(alpha: 0.1)),
),
gridData: FlGridData(
  show: true,
  drawVerticalLine: false,
  getDrawingHorizontalLine: (value) {
    return FlLine(
      color: Colors.white.withValues(alpha: 0.1),
      strokeWidth: 1,
    );
  },
),
barGroups: top5.asMap().entries.map((entry) {
  final colors = [
    const Color(0xFFFF073A),
    const Color(0xFF4ECDC4),
    const Color(0xFFFFE66D),
    const Color(0xFFA8E6CF),
    const Color(0xFFFF8B94),
  ];
  return BarChartGroupData(
    x: entry.key,
    barRods: [
      BarChartRodData(
        toY: entry.value.value.toDouble(),
        gradient: LinearGradient(
          colors: [
            colors[entry.key % colors.length],
            colors[entry.key % colors.length]
                .withValues(alpha: 0.5),
          ],
          begin: Alignment.bottomCenter,
          end: Alignment.topCenter,
        ),
        width: isMobile ? 20 : 30,
        borderRadius: const BorderRadius.only(
          topLeft: Radius.circular(6),
          topRight: Radius.circular(6),
```

```
                                  ),
                                ),
                              ],
                            );
                          }).toList(),
                        ),
                      ),
                    ),
                  ],
                ),
              );
            },
          ),
        );
      },
    );
  }

  Widget _buildLoadingCard() {
    return Container(
      padding: const EdgeInsets.all(40),
      decoration: BoxDecoration(
        color: const Color(0xFF1A1A1A),
        borderRadius: BorderRadius.circular(16),
        border: Border.all(color: Colors.white.withOpacity(0.1)),
      ),
      child: const Center(
        child: CircularProgressIndicator(
          valueColor: AlwaysStoppedAnimation<Color>(Color(0xFFFF6B6B)),
        ),
      ),
    );
  }

  Widget _buildStatRow(String label, String value, IconData icon, Color
color) {
    return LayoutBuilder(
      builder: (context, constraints) {
        final isSmall = constraints.maxWidth < 400;

        return Padding(
          padding: const EdgeInsets.symmetric(vertical: 8),
          child: Row(
            children: [
              Container(
                padding: EdgeInsets.all(isSmall ? 8 : 10),
                decoration: BoxDecoration(
                  color: color.withOpacity(0.15),
                  borderRadius: BorderRadius.circular(10),
                ),
                child: Icon(icon, color: color, size: isSmall ? 18 : 22),
              ),
              SizedBox(width: isSmall ? 12 : 16),
              Expanded(
```

```
                child: Text(
                  label,
                  style: TextStyle(
                    fontSize: isSmall ? 13 : 15,
                    color: Colors.white70,
                  ),
                ),
              ),
              Text(
                value,
                style: TextStyle(
                  fontSize: isSmall ? 16 : 20,
                  fontWeight: FontWeight.bold,
                  color: color,
                ),
              ),
            ],
          ),
        );
      },
    );
  }

  List<Venta> _filtrarVentasPorFecha(List<Venta> ventas) {
    return ventas.where((venta) {
      if (_fechaInicio != null && venta.fecha.isBefore(_fechaInicio!)) {
        return false;
      }
      if (_fechaFin != null &&
          venta.fecha.isAfter(_fechaFin!.add(const Duration(days: 1)))) {
        return false;
      }
      return true;
    }).toList();
  }
}

// ignore: unused_element
class _StatItem {
  final String label;
  final String value;
  final IconData icon;
  final Color color;

  _StatItem({
    required this.label,
    required this.value,
    required this.icon,
    required this.color,
  });
}
```

```
Admin_usuarios_tab.dart
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import '../../core/constants.dart';
import '../../viewmodels/usuario_viewmodel.dart';
import '../../data/models/usuario.dart';
import '../../screens/usuario_form_screen.dart';
import '../../screens/usuario_detalle_screen.dart';

/// 👥 TAB DE USUARIOS - SOLO ADMIN
/// Gestiona todos los usuarios de la plataforma
class AdminUsuariosTab extends StatefulWidget {
  const AdminUsuariosTab({Key? key}) : super(key: key);

  @override
  State<AdminUsuariosTab> createState() => _AdminUsuariosTabState();
}

class _AdminUsuariosTabState extends State<AdminUsuariosTab> {
  final _busquedaController = TextEditingController();
  String? _filtroRol;
  bool _soloActivos = true;

  Map<String, dynamic> _getResponsiveSizes(double width) {
    if (width > 900) {
      return {
        'fontSize': 16.0,
        'titleSize': 22.0,
        'padding': 24.0,
        'spacing': 16.0,
        'iconSize': 24.0,
        'cardPadding': 16.0,
      };
    } else if (width > 600) {
      return {
        'fontSize': 15.0,
        'titleSize': 20.0,
        'padding': 20.0,
        'spacing': 14.0,
        'iconSize': 22.0,
        'cardPadding': 14.0,
      };
    } else {
      return {
        'fontSize': 14.0,
        'titleSize': 18.0,
        'padding': 16.0,
        'spacing': 12.0,
        'iconSize': 20.0,
        'cardPadding': 12.0,
      };
    }
  }
```

```
  @override
  void initState() {
    super.initState();
    WidgetsBinding.instance.addPostFrameCallback((_) {
      context.read<UsuarioViewModel>().cargarUsuarios();
    });
  }

  @override
  void dispose() {
    _busquedaController.dispose();
    super.dispose();
  }

  List<Usuario> _filtrarUsuarios(List<Usuario> usuarios) {
    var filtrados = usuarios;

    if (_filtroRol != null) {
      filtrados = filtrados.where((u) => u.rol == _filtroRol).toList();
    }

    if (_soloActivos) {
      filtrados = filtrados.where((u) => u.activo).toList();
    }

    final termino = _busquedaController.text.toLowerCase();
    if (termino.isNotEmpty) {
      filtrados = filtrados.where((u) {
        return u.nombre.toLowerCase().contains(termino) ||
               u.email.toLowerCase().contains(termino) ||
               (u.telefono?.toLowerCase().contains(termino) ?? false);
      }).toList();
    }

    return filtrados;
  }

  @override
  Widget build(BuildContext context) {
    return Consumer<UsuarioViewModel>(
      builder: (context, viewModel, child) {
        final usuariosFiltrados = _filtrarUsuarios(viewModel.usuarios);

        return Column(
          children: [
            _buildHeader(viewModel),
            _buildFiltros(viewModel, usuariosFiltrados),
            Expanded(child: _buildListaUsuarios(viewModel,
usuariosFiltrados)),
          ],
        );
      },
    );
  }
```

```
// ========================================
// HEADER
// ========================================
Widget _buildHeader(UsuarioViewModel viewModel) {
  final width = MediaQuery.of(context).size.width;
  final  sizes  =  _getResponsiveSizes(width);
  return Container(
    padding: EdgeInsets.all(sizes['padding']),
    decoration: BoxDecoration(
      gradient: const LinearGradient(
        colors: [Color(0xFF1A1A1A), Color(0xFF2A2A2A)],
      ),
      boxShadow: [
        BoxShadow(
          color: Colors.black.withOpacity(0.3),
          blurRadius: 8,
          offset: const Offset(0, 2),
        ),
      ],
    ),
    child: LayoutBuilder(
      builder: (context, constraints) {
        final isMobile = constraints.maxWidth < 600;
        final title = Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            Text(
              'Gestión de Usuarios',
              style: TextStyle(
                color: Colors.white,
                fontSize: sizes['titleSize'],
                fontWeight: FontWeight.bold,
              ),
            ),
            SizedBox(height: sizes['spacing'] / 2),
            Text(
              'Administra todos los usuarios de la plataforma',
              style: TextStyle(
                color: Colors.grey,
                fontSize: sizes['fontSize'] - 1,
              ),
            ),
          ],
        );

        final actions = Row(
          mainAxisSize: MainAxisSize.min,
          children: [
            IconButton(
              onPressed: () => viewModel.cargarUsuarios(),
              icon: const Icon(Icons.refresh_rounded),
              color: const Color(0xFFFF6B6B),
              tooltip: 'Actualizar',
```

```
                iconSize: sizes['iconSize'],
              ),
              SizedBox(width: sizes['spacing'] / 2),
              ElevatedButton.icon(
                onPressed: _crearVendedor,
                icon: const Icon(Icons.point_of_sale_rounded),
                label: const Text('Nuevo Vendedor'),
                style: ElevatedButton.styleFrom(
                  backgroundColor: const Color(0xFF3498DB),
                  foregroundColor: Colors.white,
                  padding: EdgeInsets.symmetric(horizontal: sizes['spacing'],
vertical: sizes['spacing'] * 0.6),
                  shape: RoundedRectangleBorder(borderRadius:
BorderRadius.circular(10)),
                ),
              ),
            ],
          );

          if (isMobile) {
            return Column(
              crossAxisAlignment: CrossAxisAlignment.start,
              children: [
                Row(
                  children: [
                    Container(
                      padding: EdgeInsets.all(sizes['spacing']),
                      decoration: BoxDecoration(
                        gradient: const LinearGradient(
                          colors: [Color(0xFFFF6B6B), Color(0xFFFF8E53)],
                        ),
                        borderRadius: BorderRadius.circular(12),
                        boxShadow: [
                          BoxShadow(
                            color: const Color(0xFFFF6B6B).withOpacity(0.3),
                            blurRadius: 8,
                          ),
                        ],
                      ),
                      child: Icon(
                        Icons.people_rounded,
                        color: Colors.white,
                        size: sizes['iconSize'] + 4,
                      ),
                    ),
                    SizedBox(width: sizes['spacing']),
                    Expanded(child: title),
                  ],
                ),
                SizedBox(height: sizes['spacing']),
                Align(
                  alignment: Alignment.centerLeft,
                  child: actions,
                ),
```

```
            ],
          );
        } else {
          return Row(
            children: [
              Container(
                padding: EdgeInsets.all(sizes['spacing']),
                decoration: BoxDecoration(
                  gradient: const LinearGradient(
                    colors: [Color(0xFFFF6B6B), Color(0xFFFF8E53)],
                  ),
                  borderRadius: BorderRadius.circular(12),
                  boxShadow: [
                    BoxShadow(
                      color: const Color(0xFFFF6B6B).withOpacity(0.3),
                      blurRadius: 8,
                    ),
                  ],
                ),
                child: Icon(
                  Icons.people_rounded,
                  color: Colors.white,
                  size: sizes['iconSize'] + 4,
                ),
              ),
              SizedBox(width: sizes['spacing']),
              Expanded(child: title),
              actions,
            ],
          );
        }
      },
    ),
  );
}

// ========================================
// FILTROS
// ========================================
Widget _buildFiltros(
  UsuarioViewModel viewModel,
  List<Usuario> usuariosFiltrados,
) {
  return Container(
    padding: const EdgeInsets.all(16),
    decoration: BoxDecoration(
      color: const Color(0xFF1A1A1A),
      boxShadow: [
        BoxShadow(
          color: Colors.black.withOpacity(0.2),
          blurRadius: 8,
          offset: const Offset(0, 2),
        ),
      ],
```

```
),
child: Column(
  children: [
    Container(
      decoration: BoxDecoration(
        color: const Color(0xFF2A2A2A),
        borderRadius: BorderRadius.circular(12),
        border: Border.all(color: Colors.white.withOpacity(0.1)),
      ),
      child: TextField(
        controller: _busquedaController,
        style: const TextStyle(
          color: Colors.white,
          fontSize: 15,
        ),
        decoration: InputDecoration(
          hintText: 'Buscar por nombre, email o teléfono...',
          hintStyle: TextStyle(
            color: Colors.grey[600],
            fontSize: 14,
          ),
          prefixIcon: Icon(
            Icons.search_rounded,
            color: Colors.grey[600],
            size: 24,
          ),
          suffixIcon: _busquedaController.text.isNotEmpty
              ? IconButton(
                  icon: const Icon(Icons.clear_rounded),
                  color: Colors.grey[600],
                  onPressed: () {
                    _busquedaController.clear();
                    setState(() {});
                  },
                )
              : null,
          border: InputBorder.none,
          contentPadding: const EdgeInsets.symmetric(
            horizontal: 16,
            vertical: 14,
          ),
        ),
        onChanged: (_) => setState(() {}),
      ),
    ),

    const SizedBox(height: 16),

    SingleChildScrollView(
      scrollDirection: Axis.horizontal,
      child: Row(
        children: [
          _buildFiltroChip('Todos', null, Icons.people_rounded),
          _buildFiltroChip('Admins', AppConstants.rolAdmin,
```

```
Icons.admin_panel_settings_rounded),
                _buildFiltroChip('Vendedores', AppConstants.rolVendedor,
Icons.point_of_sale_rounded),
                _buildFiltroChip('Clientes', AppConstants.rolCliente,
Icons.person_rounded),

                const SizedBox(width: 12),
                Container(
                  height: 30,
                  width: 1,
                  color: Colors.grey[700],
                ),
                const SizedBox(width: 12),

                FilterChip(
                  label: Row(
                    children: [
                      Icon(
                        _soloActivos ? Icons.check_circle_rounded :
Icons.all_inclusive_rounded,
                        size: 18,
                        color: _soloActivos ? Colors.white :
Colors.grey[400],
                      ),
                      const SizedBox(width: 6),
                      Text(
                        _soloActivos ? 'Solo Activos' : 'Todos',
                        style: TextStyle(
                          color: _soloActivos ? Colors.white :
Colors.grey[400],
                          fontWeight: _soloActivos ? FontWeight.bold :
FontWeight.w500,
                        ),
                      ),
                    ],
                  ),
                  selected: _soloActivos,
                  backgroundColor: const Color(0xFF2A2A2A),
                  selectedColor: const Color(0xFF27AE60).withOpacity(0.2),
                  side: BorderSide(
                    color: _soloActivos ? const Color(0xFF27AE60) :
Colors.grey[700]!,
                    width: 1.5,
                  ),
                  checkmarkColor: const Color(0xFF27AE60),
                  onSelected: (selected) {
                    setState(() => _soloActivos = selected);
                  },
                  padding: const EdgeInsets.symmetric(horizontal: 12,
vertical: 8),
                ),
              ],
            ),
          ),
```

```
          if (usuariosFiltrados.length != viewModel.usuarios.length) ...[
            const SizedBox(height: 12),
            Container(
              padding: const EdgeInsets.symmetric(horizontal: 12, vertical:
8),
              decoration: BoxDecoration(
                color: const Color(0xFFFF6B6B).withOpacity(0.15),
                borderRadius: BorderRadius.circular(8),
                border: Border.all(
                  color: const Color(0xFFFF6B6B).withOpacity(0.3),
                ),
              ),
              child: Row(
                mainAxisSize: MainAxisSize.min,
                children: [
                  const Icon(
                    Icons.filter_list_rounded,
                    size: 16,
                    color: Color(0xFFFF6B6B),
                  ),
                  const SizedBox(width: 8),
                  Text(
                    'Mostrando ${usuariosFiltrados.length} de
${viewModel.usuarios.length} usuarios',
                    style: const TextStyle(
                      fontSize: 13,
                      color: Colors.white,
                      fontWeight: FontWeight.w500,
                    ),
                  ),
                ],
              ),
            ),
          ],
        ],
      ),
    );
  }

  Widget _buildFiltroChip(String label, String? rol, IconData icono) {
    final seleccionado = _filtroRol == rol;
    Color chipColor = const Color(0xFFFF6B6B);

    if (rol == AppConstants.rolAdmin) {
      chipColor = const Color(0xFFE74C3C);
    } else if (rol == AppConstants.rolVendedor) {
      chipColor = const Color(0xFF3498DB);
    } else if (rol == AppConstants.rolCliente) {
      chipColor = const Color(0xFF27AE60);
    }

    return Padding(
      padding: const EdgeInsets.only(right: 10),
```

```
    child: FilterChip(
      label: Row(
        children: [
          Icon(
            icono,
            size: 18,
            color: seleccionado ? Colors.white : Colors.grey[400],
          ),
          const SizedBox(width: 6),
          Text(
            label,
            style: TextStyle(
              color: seleccionado ? Colors.white : Colors.grey[400],
              fontWeight: seleccionado ? FontWeight.bold : FontWeight.w500,
            ),
          ),
        ],
      ),
      selected: seleccionado,
      backgroundColor: const Color(0xFF2A2A2A),
      selectedColor: chipColor.withOpacity(0.2),
      side: BorderSide(
        color: seleccionado ? chipColor : Colors.grey[700]!,
        width: 1.5,
      ),
      checkmarkColor: chipColor,
      onSelected: (selected) {
        setState(() => _filtroRol = selected ? rol : null);
      },
      padding: const EdgeInsets.symmetric(horizontal: 12, vertical: 8),
    ),
  );
}

// ========================================
// LISTA DE USUARIOS
// ========================================
Widget _buildListaUsuarios(
  UsuarioViewModel viewModel,
  List<Usuario> usuariosFiltrados,
) {
  if (viewModel.cargando) {
    return const Center(
      child: CircularProgressIndicator(color: Color(0xFFFF6B6B)),
    );
  }

  if (usuariosFiltrados.isEmpty) {
    return Center(
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: [
          Icon(
            _busquedaController.text.isNotEmpty
```

```
                ? Icons.search_off_rounded
                : Icons.people_outline_rounded,
            size: 80,
            color: Colors.grey[700],
          ),
          const SizedBox(height: 20),
          Text(
            _busquedaController.text.isNotEmpty
                ? 'No se encontraron usuarios'
                : 'No hay usuarios registrados',
            style: TextStyle(
              fontSize: 20,
              fontWeight: FontWeight.w600,
              color: Colors.grey[400],
            ),
          ),
          const SizedBox(height: 8),
          TextButton.icon(
            onPressed: _crearUsuario,
            icon: const Icon(Icons.add_rounded, color: Color(0xFFFF6B6B)),
            label: const Text(
              'Agregar Usuario',
              style: TextStyle(color: Color(0xFFFF6B6B)),
            ),
          ),
          const SizedBox(height: 8),
          TextButton.icon(
            onPressed: _crearVendedor,
            icon: const Icon(Icons.point_of_sale_rounded, color:
Color(0xFF3498DB)),
            label: const Text(
              'Agregar Vendedor',
              style: TextStyle(color: Color(0xFF3498DB)),
            ),
          ),
        ],
      ),
    );
  }

  return RefreshIndicator(
    onRefresh: () => viewModel.cargarUsuarios(),
    color: const Color(0xFFFF6B6B),
    child: ListView.builder(
      padding: const EdgeInsets.all(16),
      itemCount: usuariosFiltrados.length,
      itemBuilder: (context, index) {
        return _buildUsuarioCard(usuariosFiltrados[index]);
      },
    ),
  );
}

// ======================================
```

```
// CARD DE USUARIO
// ========================================
Widget _buildUsuarioCard(Usuario usuario) {
  Color colorRol = const Color(0xFFE74C3C);
  IconData iconoRol = Icons.person_rounded;
  String labelRol = 'Usuario';

  switch (usuario.rol) {
    case AppConstants.rolAdmin:
      colorRol = const Color(0xFFE74C3C);
      iconoRol = Icons.admin_panel_settings_rounded;
      labelRol = 'Admin';
      break;
    case AppConstants.rolVendedor:
      colorRol = const Color(0xFF3498DB);
      iconoRol = Icons.point_of_sale_rounded;
      labelRol = 'Vendedor';
      break;
    case AppConstants.rolCliente:
      colorRol = const Color(0xFF27AE60);
      iconoRol = Icons.person_rounded;
      labelRol = 'Cliente';
      break;
  }

  return Container(
    margin: const EdgeInsets.only(bottom: 12),
    decoration: BoxDecoration(
      color: const Color(0xFF1A1A1A),
      borderRadius: BorderRadius.circular(16),
      border: Border.all(
        color: Colors.white.withOpacity(0.1),
      ),
      boxShadow: [
        BoxShadow(
          color: Colors.black.withOpacity(0.2),
          blurRadius: 8,
          offset: const Offset(0, 2),
        ),
      ],
    ),
    child: InkWell(
      borderRadius: BorderRadius.circular(16),
      onTap: () => _verDetalle(usuario),
      child: Padding(
        padding: const EdgeInsets.all(16),
        child: Row(
          children: [
            Container(
              width: 60,
              height: 60,
              decoration: BoxDecoration(
                gradient: LinearGradient(
                  colors: [
```

```
        colorRol.withOpacity(0.3),
        colorRol.withOpacity(0.1),
      ],
    ),
    borderRadius: BorderRadius.circular(12),
    border: Border.all(
      color: colorRol,
      width: 2,
    ),
  ),
  child: Icon(
    iconoRol,
    color: colorRol,
    size: 30,
  ),
),
const SizedBox(width: 16),

// Información del usuario
Expanded(
  child: Column(
    crossAxisAlignment: CrossAxisAlignment.start,
    children: [
      Row(
        children: [
          Expanded(
            child: Text(
              usuario.nombre,
              style: const TextStyle(
                fontWeight: FontWeight.bold,
                color: Colors.white,
                fontSize: 16,
              ),
              overflow: TextOverflow.ellipsis,
            ),
          ),
          Container(
            padding: const EdgeInsets.symmetric(
              horizontal: 10,
              vertical: 4,
            ),
            decoration: BoxDecoration(
              color: colorRol.withOpacity(0.2),
              borderRadius: BorderRadius.circular(8),
              border: Border.all(color: colorRol, width: 1.5),
            ),
            child: Text(
              labelRol,
              style: TextStyle(
                color: colorRol,
                fontSize: 11,
                fontWeight: FontWeight.bold,
              ),
            ),
```

```
                        ),
                      ],
                    ),
                    const SizedBox(height: 6),

                    Row(
                      children: [
                        Icon(Icons.email_outlined, size: 14, color:
Colors.grey[500]),
                        const SizedBox(width: 6),
                        Expanded(
                          child: Text(
                            usuario.email,
                            style: TextStyle(
                              color: Colors.grey[400],
                              fontSize: 13,
                            ),
                            overflow: TextOverflow.ellipsis,
                          ),
                        ),
                      ],
                    ),

                    if (usuario.telefono != null) ...[
                      const SizedBox(height: 4),
                      Row(
                        children: [
                          Icon(Icons.phone_outlined, size: 14, color:
Colors.grey[500]),
                          const SizedBox(width: 6),
                          Text(
                            usuario.telefono!,
                            style: TextStyle(
                              color: Colors.grey[400],
                              fontSize: 13,
                            ),
                          ),
                        ],
                      ),
                    ],
                  ],
                ),
              ),

              Column(
                children: [
                  Container(
                    padding: const EdgeInsets.all(8),
                    decoration: BoxDecoration(
                      color: usuario.activo
                          ? const Color(0xFF27AE60).withOpacity(0.2)
                          : const Color(0xFFE74C3C).withOpacity(0.2),
                      borderRadius: BorderRadius.circular(8),
                    ),
```

```
                  child: Icon(
                    usuario.activo
                        ? Icons.check_circle_rounded
                        : Icons.cancel_rounded,
                    color: usuario.activo
                        ? const Color(0xFF27AE60)
                        : const Color(0xFFE74C3C),
                    size: 20,
                  ),
                ),
                const SizedBox(height: 8),
                _buildMenuAcciones(usuario),
              ],
            ),
          ],
        ),
      ),
    ),
  );
}

Widget _buildMenuAcciones(Usuario usuario) {
  return PopupMenuButton<String>(
    icon: Icon(Icons.more_vert_rounded, color: Colors.grey[600]),
    color: const Color(0xFF2A2A2A),
    shape: RoundedRectangleBorder(
      borderRadius: BorderRadius.circular(12),
      side: BorderSide(color: Colors.white.withOpacity(0.1)),
    ),
    onSelected: (value) {
      switch (value) {
       case 'editar':
          _editarUsuario(usuario);
          break;
        case 'eliminar':
          _confirmarEliminar(usuario);
          break;
        case 'toggle':
          _toggleActivo(usuario);
          break;
      }
    },
    itemBuilder: (context) => [
      const PopupMenuItem(
        value: 'editar',
        child: Row(
          children: [
            Icon(Icons.edit_rounded, color: Color(0xFF3498DB), size: 20),
            SizedBox(width: 12),
            Text('Editar', style: TextStyle(color: Colors.white)),
          ],
        ),
      ),
      PopupMenuItem(
```

```
          value: 'toggle',
          child: Row(
            children: [
              Icon(
                usuario.activo ? Icons.block_rounded :
Icons.check_circle_rounded,
                color: usuario.activo ? const Color(0xFFF39C12) : const
Color(0xFF27AE60),
                size: 20,
              ),
              const SizedBox(width: 12),
              Text(
                usuario.activo ? 'Desactivar' : 'Activar',
                style: const TextStyle(color: Colors.white),
              ),
            ],
          ),
        ),
        const PopupMenuItem(
          value: 'eliminar',
          child: Row(
            children: [
              Icon(Icons.delete_rounded, color: Color(0xFFE74C3C), size: 20),
              SizedBox(width: 12),
              Text('Eliminar', style: TextStyle(color: Color(0xFFE74C3C))),
            ],
          ),
        ),
      ],
    );
  }


  // ======================================
  // ACCIONES
  // ======================================
  void _crearUsuario() {
    Navigator.push(
      context,
      MaterialPageRoute(
        builder: (context) => const UsuarioFormScreen(),
      ),
    );
  }

  void _crearVendedor() {
    Navigator.push(
      context,
      MaterialPageRoute(
        builder: (context) => const UsuarioFormScreen(rolInicial:
AppConstants.rolVendedor),
      ),
    );
  }
```

```
  void _verDetalle(Usuario usuario) {
    Navigator.push(
      context,
      MaterialPageRoute(
        builder: (context) => UsuarioDetalleScreen(usuario: usuario),
      ),
    );
  }

  void _editarUsuario(Usuario usuario) {
    Navigator.push(
      context,
      MaterialPageRoute(
        builder: (context) => UsuarioFormScreen(usuario: usuario),
      ),
    );
  }

  void _toggleActivo(Usuario usuario) async {
    final viewModel = context.read<UsuarioViewModel>();
    final nuevoEstado = !usuario.activo;

    final success = await viewModel.actualizarUsuario(
      usuario.copyWith(activo: nuevoEstado),
    );

    if (success && mounted) {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(
          content: Text(
            'Usuario ${nuevoEstado ? "activado" : "desactivado"}',
          ),
          backgroundColor: nuevoEstado ? const Color(0xFF27AE60) : const
Color(0xFFF39C12),
        ),
      );
    }
  }

  void _confirmarEliminar(Usuario usuario) {
    showDialog(
      context: context,
      builder: (context) => AlertDialog(
        backgroundColor: const Color(0xFF1A1A1A),
        shape: RoundedRectangleBorder(borderRadius:
BorderRadius.circular(16)),
        title: Row(
          children: [
            Container(
              padding: const EdgeInsets.all(10),
              decoration: BoxDecoration(
                color: const Color(0xFFE74C3C).withOpacity(0.2),
                borderRadius: BorderRadius.circular(10),
```

```
            ),
            child: const Icon(
              Icons.delete_rounded,
              color: Color(0xFFE74C3C),
              size: 24,
            ),
          ),
          const SizedBox(width: 12),
          const Text(
            'Eliminar Usuario',
            style: TextStyle(color: Colors.white),
          ),
        ],
      ),
      content: Text(
        '¿Estás seguro de eliminar a ${usuario.nombre}?',
        style: TextStyle(color: Colors.grey[400]),
      ),
      actions: [
        TextButton(
          onPressed: () => Navigator.pop(context),
          child: Text(
            'Cancelar',
            style: TextStyle(color: Colors.grey[400]),
          ),
        ),
        ElevatedButton(
          onPressed: () async {
            final viewModel = context.read<UsuarioViewModel>();
            final success = await viewModel.eliminarUsuario(usuario.id!);
            if (mounted) {
              Navigator.pop(context);
              if (success) {
                ScaffoldMessenger.of(context).showSnackBar(
                  const SnackBar(
                    content: Text('Usuario eliminado'),
                    backgroundColor: Color(0xFF27AE60),
                  ),
                );
              }
            }
          },
          style: ElevatedButton.styleFrom(
            backgroundColor: const Color(0xFFE74C3C),
            foregroundColor: Colors.white,
            shape: RoundedRectangleBorder(
              borderRadius: BorderRadius.circular(10),
            ),
          ),
          child: const Text('Eliminar'),
        ),
      ],
    ),
  );
```

```
  }
}

Admin_ventas_tab.dart
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import '../../core/constants.dart';
import '../../viewmodels/venta_viewmodel.dart';
import '../../utils/formatters.dart';
import '../../data/models/venta.dart';

class AdminVentasTab extends StatefulWidget {
  const AdminVentasTab({Key? key}) : super(key: key);

  @override
  State<AdminVentasTab> createState() => _AdminVentasTabState();
}

class _AdminVentasTabState extends State<AdminVentasTab> {
  String _filtroSeleccionado = 'hoy';

  @override
  void initState() {
    super.initState();
    WidgetsBinding.instance.addPostFrameCallback((_) {
      context.read<VentaViewModel>().cargarVentasHoy();
    });
  }

  Map<String, dynamic> _getResponsiveSizes(double width) {
    if (width > 1200) {
      return {
        'titleSize': 22.0,
        'padding': 24.0,
        'spacing': 18.0,
        'fontSize': 15.0,
        'iconSize': 26.0,
        'cardHeight': 120.0,
      };
    } else if (width > 800) {
      return {
        'titleSize': 20.0,
        'padding': 20.0,
        'spacing': 16.0,
        'fontSize': 14.0,
        'iconSize': 24.0,
        'cardHeight': 110.0,
      };
    } else if (width > 600) {
      return {
        'titleSize': 18.0,
        'padding': 14.0,
        'spacing': 12.0,
        'fontSize': 13.0,
```

```
        'iconSize': 22.0,
        'cardHeight': 100.0,
      };
    } else {
      return {
        'titleSize': 16.0,
        'padding': 12.0,
        'spacing': 10.0,
        'fontSize': 12.0,
        'iconSize': 20.0,
        'cardHeight': 90.0,
      };
    }
  }

  @override
  Widget build(BuildContext context) {
    final width = MediaQuery.of(context).size.width;
    final isMobile = width < 600;
    final sizes = _getResponsiveSizes(width);

    return Scaffold(
      backgroundColor: const Color(0xFF1A1A1A),
      body: Consumer<VentaViewModel>(
        builder: (context, viewModel, child) {
          return Column(
            children: [
              // Filtros y resumen
              Container(
                padding: EdgeInsets.all(sizes['padding']),
                color: const Color(0xFF2A2A2A),
                child: Column(
                  children: [
                    // Chips de filtros
                    SingleChildScrollView(
                      scrollDirection: Axis.horizontal,
                      child: Row(
                        children: [
                          _buildFiltroChip('Hoy', 'hoy', viewModel, sizes),
                          _buildFiltroChip('Esta Semana', 'semana',
viewModel, sizes),
                          _buildFiltroChip('Este Mes', 'mes', viewModel,
sizes),
                          _buildFiltroChip('Todas', 'todas', viewModel,
sizes),
                        ],
                      ),
                    ),
                    SizedBox(height: sizes['spacing']),

                    // Resumen de ventas
                    Container(
                      padding: EdgeInsets.all(sizes['padding']),
                      decoration: BoxDecoration(
```

```
                        gradient: const LinearGradient(
                          colors: [Color(0xFFE74C3C), Color(0xFFFF6B6B)],
                        ),
                        borderRadius: BorderRadius.circular(12),
                        boxShadow: [
                          BoxShadow(
                            color: const Color(0xFFE74C3C).withOpacity(0.3),
                            blurRadius: 8,
                            offset: const Offset(0, 4),
                          ),
                        ],
                      ),
                      child: Column(
                        children: [
                          Row(
                            children: [
                              Icon(
                                Icons.admin_panel_settings,
                                color: Colors.white,
                                size: sizes['iconSize'],
                              ),
                              const SizedBox(width: 8),
                              Expanded(
                                child: Text(
                                  _getTituloResumen(),
                                  style: TextStyle(
                                    color: Colors.white,
                                    fontSize: sizes['titleSize'],
                                    fontWeight: FontWeight.bold,
                                  ),
                                  overflow: TextOverflow.ellipsis,
                                ),
                              ),
                            ],
                          ),
                          SizedBox(height: sizes['spacing']),
                          // Adaptar layout según el tamaño de pantalla
                          if (isMobile)
                            Column(
                              children: [
                                _buildResumenItem(
                                  'Completadas',
                                  '${_getVentasCompletadas(viewModel)}',
                                  Icons.check_circle,
                                  sizes,
                                ),
                                Divider(
                                  color: Colors.white.withOpacity(0.3),
                                  height: sizes['spacing'] * 2,
                                ),
                                _buildResumenItem(
                                  'Total Recaudado',
Formateadores.moneda(_getTotalVentas(viewModel)),
```

```
                                    Icons.attach_money,
                                    sizes,
                                  ),
                                ],
                              )
                            else
                              Row(
                                mainAxisAlignment:
MainAxisAlignment.spaceAround,
                                children: [
                                  _buildResumenItem(
                                    'Completadas',
                                    '${_getVentasCompletadas(viewModel)}',
                                    Icons.check_circle,
                                    sizes,
                                  ),
                                  Container(
                                    width: 1,
                                    height: 50,
                                    color: Colors.white.withOpacity(0.3),
                                  ),
                                  _buildResumenItem(
                                    'Total Recaudado',

Formateadores.moneda(_getTotalVentas(viewModel)),
                                    Icons.attach_money,
                                    sizes,
                                  ),
                                ],
                              ),
                          ],
                        ),
                      ),
                    ),
                  ),
                ],
              ),
            ),

            // Lista de ventas
            Expanded(
              child: _buildListaVentas(viewModel, sizes, isMobile),
            ),
          ],
        );
      },
    ),
  );
}

Widget _buildFiltroChip(
  String label,
  String valor,
  VentaViewModel viewModel,
  Map<String, dynamic> sizes,
) {
```

```
    final isSeleccionado = _filtroSeleccionado == valor;

    return Padding(
      padding: const EdgeInsets.only(right: 10),
      child: FilterChip(
        label: Text(
          label,
          style: TextStyle(fontSize: sizes['fontSize']),
        ),
        selected: isSeleccionado,
        backgroundColor: const Color(0xFF1A1A1A),
        selectedColor: const Color(0xFFE74C3C),
        checkmarkColor: Colors.white,
        labelStyle: TextStyle(
          color: isSeleccionado ? Colors.white : Colors.grey[400],
          fontWeight: isSeleccionado ? FontWeight.w600 : FontWeight.normal,
        ),
        side: BorderSide(
          color: isSeleccionado ? const Color(0xFFE74C3C) :
Colors.grey[700]!,
        ),
        onSelected: (selected) {
          setState(() {
            _filtroSeleccionado = valor;
          });

          switch (valor) {
            case 'hoy':
              viewModel.cargarVentasHoy();
              break;
            case 'semana':
              _cargarVentasSemana(viewModel);
              break;
            case 'mes':
              _cargarVentasMes(viewModel);
              break;
            case 'todas':
              viewModel.cargarVentas();
              break;
          }
        },
      ),
    );
  }

  void _cargarVentasSemana(VentaViewModel viewModel) {
    final ahora = DateTime.now();
    final inicioSemana = ahora.subtract(Duration(days: ahora.weekday - 1));
    final inicio = DateTime(inicioSemana.year, inicioSemana.month,
inicioSemana.day);
    final fin = DateTime(ahora.year, ahora.month, ahora.day, 23, 59, 59);
    viewModel.cargarPorFecha(inicio, fin);
  }
```

```
  void _cargarVentasMes(VentaViewModel viewModel) {
    final ahora = DateTime.now();
    final inicio = DateTime(ahora.year, ahora.month, 1);
    final fin = DateTime(ahora.year, ahora.month, ahora.day, 23, 59, 59);
    viewModel.cargarPorFecha(inicio, fin);
  }

  Widget _buildResumenItem(
    String titulo,
    String valor,
    IconData icono,
    Map<String, dynamic> sizes,
  ) {
    return Column(
      children: [
        Icon(icono, color: Colors.white, size: sizes['iconSize'] + 8),
        const SizedBox(height: 4),
        Text(
          titulo,
          style: TextStyle(
            color: Colors.white70,
            fontSize: sizes['fontSize'] - 2,
          ),
        ),
        Text(
          valor,
          style: TextStyle(
            color: Colors.white,
            fontSize: sizes['fontSize'] + 3,
            fontWeight: FontWeight.bold,
          ),
        ),
      ],
    );
  }

  Widget _buildListaVentas(VentaViewModel viewModel, Map<String, dynamic>
sizes, bool isMobile) {
    if (viewModel.cargando) {
      return const Center(
        child: CircularProgressIndicator(color: Color(0xFFE74C3C)),
      );
    }

    final ventas = _getVentasActuales(viewModel);

    if (ventas.isEmpty) {
      return Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            Icon(
              Icons.receipt_long_outlined,
              size: 64,
```

```
              color: Colors.grey[600],
            ),
            const SizedBox(height: 16),
            Text(
              'No hay ventas registradas',
              style: TextStyle(
                fontSize: 18,
                color: Colors.grey[400],
              ),
            ),
            const SizedBox(height: 8),
            Text(
              'Las ventas completadas aparecerán aquí',
              style: TextStyle(
                fontSize: 14,
                color: Colors.grey[600],
              ),
            ),
          ],
        ),
      );
    }

    return ListView.builder(
      padding: EdgeInsets.all(sizes['padding']),
      itemCount: ventas.length,
      itemBuilder: (context, index) {
        final venta = ventas[index];
        return _buildVentaCard(venta, sizes, isMobile);
      },
    );
  }

  Widget _buildVentaCard(Venta venta, Map<String, dynamic> sizes, bool
isMobile) {
    Color colorEstado = const Color(0xFF4CAF50); // Verde para completada
    IconData iconoEstado = Icons.check_circle;
    String textoEstado = 'Completada';

    if (venta.estado == AppConstants.ventaCancelada) {
      colorEstado = const Color(0xFFE74C3C);
      iconoEstado = Icons.cancel;
      textoEstado = 'Cancelada';
    } else if (venta.estado == AppConstants.ventaPendiente) {
      colorEstado = Colors.orange;
      iconoEstado = Icons.pending;
      textoEstado = 'Pendiente';
    }

    return Card(
      margin: const EdgeInsets.only(bottom: 12),
      elevation: 4,
      color: const Color(0xFF2A2A2A),
      shape: RoundedRectangleBorder(
```

```
  borderRadius: BorderRadius.circular(12),
),
child: InkWell(
  borderRadius: BorderRadius.circular(12),
  onTap: () => _mostrarDetalleVenta(venta),
  child: Padding(
    padding: EdgeInsets.all(sizes['padding']),
    child: isMobile
        ? Column(
            crossAxisAlignment: CrossAxisAlignment.start,
            children: [
              // Header con estado
              Row(
                mainAxisAlignment: MainAxisAlignment.spaceBetween,
                children: [
                  Expanded(
                    child: Column(
                      crossAxisAlignment: CrossAxisAlignment.start,
                      children: [
                        Text(
                          venta.numeroVenta,
                          style: TextStyle(
                            fontWeight: FontWeight.bold,
                            color: Colors.white,
                            fontSize: sizes['fontSize'],
                          ),
                        ),
                        const SizedBox(height: 4),
                        Container(
                          padding: const EdgeInsets.symmetric(
                            horizontal: 6,
                            vertical: 2,
                          ),
                          decoration: BoxDecoration(
                            color: colorEstado,
                            borderRadius: BorderRadius.circular(4),
                          ),
                          child: Text(
                            textoEstado,
                            style: TextStyle(
                              color: Colors.white,
                              fontSize: sizes['fontSize'] - 3,
                              fontWeight: FontWeight.bold,
                            ),
                          ),
                        ),
                      ],
                    ),
                  ),
                  Text(
                    Formateadores.moneda(venta.total),
                    style: TextStyle(
                      fontWeight: FontWeight.bold,
                      fontSize: sizes['fontSize'] + 2,
```

```
                color: const Color(0xFFFFD700),
              ),
            ),
          ],
        ),
        const SizedBox(height: 8),
        // Cliente
        Text(
          venta.nombreCliente ?? 'Cliente General',
          style: TextStyle(
            color: Colors.grey[400],
            fontSize: sizes['fontSize'] - 1,
          ),
        ),
        const SizedBox(height: 6),
        // Fecha y cantidad
        Row(
          children: [
            Icon(
              Icons.access_time,
              size: sizes['fontSize'],
              color: Colors.grey[500],
            ),
            const SizedBox(width: 4),
            Text(
              Formateadores.fechaHora(venta.fecha),
              style: TextStyle(
                color: Colors.grey[500],
                fontSize: sizes['fontSize'] - 2,
              ),
            ),
            const Spacer(),
            Icon(
              Icons.shopping_bag,
              size: sizes['fontSize'],
              color: Colors.grey[500],
            ),
            const SizedBox(width: 4),
            Text(
              '${venta.detalles.length} items',
              style: TextStyle(
                fontSize: sizes['fontSize'] - 2,
                color: Colors.grey[500],
              ),
            ),
          ],
        ),
      ],
    )
  : Row(
      children: [
        // Icono de estado
        Container(
          width: 60,
```

```
  height: 60,
  decoration: BoxDecoration(
    color: colorEstado.withOpacity(0.2),
    borderRadius: BorderRadius.circular(12),
    border: Border.all(
      color: colorEstado.withOpacity(0.5),
      width: 2,
    ),
  ),
  child: Icon(
    iconoEstado,
    color: colorEstado,
    size: sizes['iconSize'] + 8,
  ),
),
const SizedBox(width: 16),

// Información de la venta
Expanded(
  child: Column(
    crossAxisAlignment: CrossAxisAlignment.start,
    children: [
      Row(
        children: [
          Text(
            venta.numeroVenta,
            style: TextStyle(
              fontWeight: FontWeight.bold,
              color: Colors.white,
              fontSize: sizes['fontSize'] + 1,
            ),
          ),
          const SizedBox(width: 8),
          Container(
            padding: const EdgeInsets.symmetric(
              horizontal: 8,
              vertical: 2,
            ),
            decoration: BoxDecoration(
              color: colorEstado,
              borderRadius: BorderRadius.circular(4),
            ),
            child: Text(
              textoEstado,
              style: TextStyle(
                color: Colors.white,
                fontSize: sizes['fontSize'] - 3,
                fontWeight: FontWeight.bold,
              ),
            ),
          ),
        ],
      ),
      const SizedBox(height: 4),
```

```
        Text(
          venta.nombreCliente ?? 'Cliente General',
          style: TextStyle(
            color: Colors.grey[400],
            fontSize: sizes['fontSize'] - 1,
          ),
        ),
        const SizedBox(height: 4),
        Row(
          children: [
            Icon(
              Icons.access_time,
              size: sizes['fontSize'],
              color: Colors.grey[500],
            ),
            const SizedBox(width: 4),
            Text(
              Formateadores.fechaHora(venta.fecha),
              style: TextStyle(
                color: Colors.grey[500],
                fontSize: sizes['fontSize'] - 2,
              ),
            ),
          ],
        ),
      ],
    ),
  ),

  // Total y cantidad
  Column(
    crossAxisAlignment: CrossAxisAlignment.end,
    children: [
      Text(
        Formateadores.moneda(venta.total),
        style: TextStyle(
          fontWeight: FontWeight.bold,
          fontSize: sizes['fontSize'] + 4,
          color: const Color(0xFFFFD700),
        ),
      ),
      const SizedBox(height: 4),
      Row(
        mainAxisSize: MainAxisSize.min,
        children: [
          Icon(
            Icons.shopping_bag,
            size: sizes['fontSize'],
            color: Colors.grey[500],
          ),
          const SizedBox(width: 4),
          Text(
            '${venta.detalles.length} items',
            style: TextStyle(
```

```
                              fontSize: sizes['fontSize'] - 2,
                              color: Colors.grey[500],
                            ),
                          ),
                        ],
                      ),
                    ],
                  ),
                ],
              ),
            ),
          ),
        );
      }

      void _mostrarDetalleVenta(Venta venta) {
        if (venta.id == null) return;

        final sizes = _getResponsiveSizes(MediaQuery.of(context).size.width);
        final viewModel = context.read<VentaViewModel>();

        showModalBottomSheet(
          context: context,
          backgroundColor: Colors.transparent,
          isScrollControlled: true,
          builder: (context) => DraggableScrollableSheet(
            initialChildSize: 0.7,
            maxChildSize: 0.9,
            minChildSize: 0.5,
            builder: (context, scrollController) => Container(
              decoration: const BoxDecoration(
                color: Color(0xFF2A2A2A),
                borderRadius: BorderRadius.vertical(top: Radius.circular(20)),
              ),
              child: Column(
                children: [
                  // Handle
                  Container(
                    margin: const EdgeInsets.only(top: 12),
                    width: 40,
                    height: 4,
                    decoration: BoxDecoration(
                      color: Colors.grey[600],
                      borderRadius: BorderRadius.circular(2),
                    ),
                  ),

                  // Header
                  Padding(
                    padding: EdgeInsets.all(sizes['padding']),
                    child: Row(
                      mainAxisAlignment: MainAxisAlignment.spaceBetween,
                      children: [
                        Text(
```

```
                      'Detalle de Venta',
                      style: TextStyle(
                        color: Colors.white,
                        fontSize: sizes['titleSize'],
                        fontWeight: FontWeight.bold,
                      ),
                    ),
                    IconButton(
                      icon: const Icon(Icons.close, color: Colors.white),
                      onPressed: () => Navigator.pop(context),
                    ),
                  ],
                ),
              ),

              Expanded(
                child: FutureBuilder<Venta?>(
                  future: viewModel.obtenerVentaDetalle(venta.id!),
                  builder: (context, snapshot) {
                    if (snapshot.connectionState == ConnectionState.waiting)
{
                      return const Center(
                        child: CircularProgressIndicator(color:
Color(0xFFE74C3C)),
                      );
                    }

                    if (snapshot.hasError) {
                      return Center(
                        child: Text(
                          'Error al cargar detalles',
                          style: TextStyle(color: Colors.red[300]),
                        ),
                      );
                    }

                    final ventaDetalle = snapshot.data ?? venta;

                    return ListView(
                      controller: scrollController,
                      padding: EdgeInsets.all(sizes['padding']),
                      children: [
                        // Info general
                        _buildInfoCard(
                          'Información General',
                          [
                            _buildInfoRow('Número', ventaDetalle.numeroVenta,
sizes),
                            _buildInfoRow('Cliente',
ventaDetalle.nombreCliente ?? 'Cliente General', sizes),
                            _buildInfoRow('Vendedor',
ventaDetalle.nombreVendedor ?? 'N/A', sizes),
                            _buildInfoRow('Fecha',
Formateadores.fechaHora(ventaDetalle.fecha), sizes),
```

```
                            _buildInfoRow('Estado', ventaDetalle.estado,
sizes),
                          ],
                          sizes,
                        ),

                        SizedBox(height: sizes['spacing']),

                        // Productos
                        _buildInfoCard(
                          'Productos (${ventaDetalle.detalles.length})',
                          ventaDetalle.detalles.isEmpty
                              ? [
                                  Padding(
                                    padding: EdgeInsets.symmetric(vertical:
sizes['spacing']),
                                    child: Text(
                                      'No hay detalles disponibles',
                                      style: TextStyle(color:
Colors.grey[400]),
                                    ),
                                  )
                                ]
                              : ventaDetalle.detalles.map((detalle) =>
Padding(
                                  padding: EdgeInsets.only(bottom:
sizes['spacing']),
                                  child: Row(
                                    children: [
                                      Container(
                                        width: 40,
                                        height: 40,
                                        decoration: BoxDecoration(
                                          color: const Color(0xFF1A1A1A),
                                          borderRadius: BorderRadius.circular(8),
                                        ),
                                        child: Center(
                                          child: Text(
                                            '${detalle.cantidad}x',
                                            style: TextStyle(
                                              color: const Color(0xFFE74C3C),
                                              fontWeight: FontWeight.bold,
                                              fontSize: sizes['fontSize'] - 1,
                                            ),
                                          ),
                                        ),
                                      ),
                                      SizedBox(width: sizes['spacing']),
                                      Expanded(
                                        child: Column(
                                          crossAxisAlignment:
CrossAxisAlignment.start,
                                          children: [
                                            Text(
```

```
                                        detalle.nombreProducto ?? 'Producto',
                                        style: TextStyle(
                                          color: Colors.white,
                                          fontSize: sizes['fontSize'],
                                        ),
                                      ),
                                      Text(
Formateadores.moneda(detalle.precioUnitario),
                                        style: TextStyle(
                                          color: Colors.grey[500],
                                          fontSize: sizes['fontSize'] - 2,
                                        ),
                                      ),
                                    ],
                                  ),
                                ),
                                Text(
                                  Formateadores.moneda(detalle.total),
                                  style: TextStyle(
                                    color: const Color(0xFFFFD700),
                                    fontWeight: FontWeight.bold,
                                    fontSize: sizes['fontSize'],
                                  ),
                                ),
                              ],
                            ),
                          )).toList(),
                          sizes,
                        ),

                        SizedBox(height: sizes['spacing']),

                        // Totales
                        _buildInfoCard(
                          'Resumen',
                          [
                            _buildInfoRow('Subtotal',
Formateadores.moneda(ventaDetalle.subtotal), sizes),
                            if (ventaDetalle.descuento > 0)
                              _buildInfoRow('Descuento', '-
${Formateadores.moneda(ventaDetalle.descuento)}', sizes),
                            Divider(color: Colors.grey[700]),
                            Padding(
                              padding: EdgeInsets.symmetric(vertical:
sizes['spacing'] / 2),
                              child: Row(
                                mainAxisAlignment:
MainAxisAlignment.spaceBetween,
                                children: [
                                  Text(
                                    'TOTAL',
                                    style: TextStyle(
                                      color: Colors.white,
```

```
                                    fontWeight: FontWeight.bold,
                                    fontSize: sizes['fontSize'] + 2,
                                  ),
                                ),
                                Text(
                                  Formateadores.moneda(ventaDetalle.total),
                                  style: TextStyle(
                                    color: const Color(0xFFFFD700),
                                    fontWeight: FontWeight.bold,
                                    fontSize: sizes['fontSize'] + 4,
                                  ),
                                ),
                              ],
                            ),
                          ),
                          ],
                          sizes,
                        ),
                      ],
                    );
                  },
                ),
              ),
            ],
          ),
        ),
      );
  }

  Widget _buildInfoCard(String titulo, List<Widget> children, Map<String,
dynamic> sizes) {
    return Container(
      padding: EdgeInsets.all(sizes['padding']),
      decoration: BoxDecoration(
        color: const Color(0xFF1A1A1A),
        borderRadius: BorderRadius.circular(12),
      ),
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          Text(
            titulo,
            style: TextStyle(
              color: const Color(0xFFE74C3C),
              fontWeight: FontWeight.bold,
              fontSize: sizes['fontSize'] + 1,
            ),
          ),
          SizedBox(height: sizes['spacing']),
          ...children,
        ],
      ),
    );
```

```
  }

  Widget _buildInfoRow(String label, String value, Map<String, dynamic>
sizes) {
    return Padding(
      padding: EdgeInsets.only(bottom: sizes['spacing'] / 2),
      child: Row(
        children: [
          SizedBox(
            width: 80,
            child: Text(
              label,
              style: TextStyle(
                color: Colors.grey[500],
                fontWeight: FontWeight.w500,
              ),
            ),
          ),
          Expanded(
            child: Text(
              value,
              style: const TextStyle(
                color: Colors.white,
                fontWeight: FontWeight.w500,
              ),
            ),
          ),
        ],
      ),
    );
  }

  String _getTituloResumen() {
    switch (_filtroSeleccionado) {
      case 'hoy':
        return 'Ventas de Hoy';
      case 'semana':
        return 'Ventas de la Semana';
      case 'mes':
        return 'Ventas del Mes';
      case 'todas':
        return 'Todas las Ventas';
      default:
        return 'Ventas';
    }
  }

  int _getVentasCompletadas(VentaViewModel viewModel) {
    // Para filtros distintos de 'hoy', calcular desde ventas
    if (_filtroSeleccionado != 'hoy') {
      return viewModel.ventas.where((v) =>
        v.estado == AppConstants.ventaCompletada
      ).length;
    }
```

```
    return viewModel.ventasCompletadasHoy;
  }

  double _getTotalVentas(VentaViewModel viewModel) {
    // Para filtros distintos de 'hoy', calcular desde ventas
    if (_filtroSeleccionado != 'hoy') {
      return viewModel.ventas.fold(0.0, (sum, v) => sum + v.total);
    }
    return viewModel.totalVentasHoy;
  }

  List<Venta> _getVentasActuales(VentaViewModel viewModel) {
    // Para filtros distintos de 'hoy', usar ventas generales
    if (_filtroSeleccionado != 'hoy') {
      return viewModel.ventas;
    }
    return viewModel.ventasHoy;
  }
}
```

Vendedor:
Vendedor_dashboard.dart

```
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import '../../data/models/usuario.dart';
import '../../viewmodels/venta_viewmodel.dart';
import 'vendedor_clientes_tab.dart';
import 'vendedor_productos_tab.dart';
import 'vendedor_ventas_tab.dart';
import 'vendedor_pedidos_tab.dart';
// import 'vendedor_crear_venta.dart'; // ELIMINADO: Vendedor ya no puede
crear ventas directas
import '../../core/app_routes.dart';
import '../../services/auth_service.dart';
import '../../services/session_timeout_service.dart';

class VendedorDashboard extends StatefulWidget {
  final Usuario vendedor;

  const VendedorDashboard({
    Key? key,
    required this.vendedor,
  }) : super(key: key);

  @override
  State<VendedorDashboard> createState() => _VendedorDashboardState();
}

class _VendedorDashboardState extends State<VendedorDashboard>
    with SingleTickerProviderStateMixin, WidgetsBindingObserver {
  late TabController _tabController;
  final SessionTimeoutService _sessionTimeoutService =
SessionTimeoutService();
```

```
  @override
  void initState() {
    super.initState();
    WidgetsBinding.instance.addObserver(this);
    _sessionTimeoutService.initialize(context);
    _tabController = TabController(length: 4, vsync: this, initialIndex: 0);
// Cambiado de 5 a 4 tabs

    WidgetsBinding.instance.addPostFrameCallback((_) {
      context.read<VentaViewModel>().cargarPorVendedor(widget.vendedor.id!);
    });
  }

  @override
  void dispose() {
    WidgetsBinding.instance.removeObserver(this);
    _sessionTimeoutService.cancel();
    _tabController.dispose();
    super.dispose();
  }

  @override
  Widget build(BuildContext context) {
    // Obtener información del dispositivo
    final size = MediaQuery.of(context).size;
    final isTablet = size.width >= 600;
    final isDesktop = size.width >= 1024;
    final isMobile = size.width < 600;
    final orientation = MediaQuery.of(context).orientation;

    return Scaffold(
      backgroundColor: const Color(0xFF1A1A1A),
      appBar: _buildAppBar(context, isMobile, isTablet),
      drawer: isMobile ? _buildDrawer(context) : null,
      body: _buildBody(context, isMobile, isTablet, isDesktop, orientation),
      floatingActionButton: _buildFAB(context, isMobile, isTablet),
      floatingActionButtonLocation: _getFABLocation(isMobile),
    );
  }

  PreferredSizeWidget _buildAppBar(BuildContext context, bool isMobile, bool
isTablet) {
    return AppBar(
      backgroundColor: const Color(0xFF1A1A1A),
      automaticallyImplyLeading: false,
      title: LayoutBuilder(
        builder: (context, constraints) {
          return Text(
            isMobile
              ? widget.vendedor.nombre.split(' ').first
              : 'Vendedor: ${widget.vendedor.nombre}',
            style: TextStyle(
              color: Colors.white,
              fontSize: isMobile ? 18 : 20,
```

```
            ),
            overflow: TextOverflow.ellipsis,
          );
        },
      ),
      centerTitle: !isMobile,
      elevation: 0,
      actions: isMobile ? null : _buildAppBarActions(context, isTablet),
      bottom: isMobile
        ? null
        : TabBar(
            controller: _tabController,
            isScrollable: true,
            indicatorColor: Theme.of(context).colorScheme.primary,
            labelColor: Colors.white,
            unselectedLabelColor: Colors.white70,
            tabAlignment: isTablet ? TabAlignment.start :
TabAlignment.center,
            labelPadding: EdgeInsets.symmetric(horizontal: isTablet ? 24 :
16),
            tabs: _buildTabs(isTablet),
          ),
    );
  }

  List<Widget> _buildAppBarActions(BuildContext context, bool isTablet) {
    final iconSize = isTablet ? 24.0 : 20.0;

    return [
      IconButton(
        icon: Icon(Icons.person_outline, color: Colors.white, size:
iconSize),
        tooltip: 'Mi Perfil',
        onPressed: () {
          Navigator.pushNamed(
            context,
            AppRoutes.perfilUsuario,
            arguments: widget.vendedor,
          );
        },
      ),
      IconButton(
        icon: Icon(Icons.settings_outlined, color: Colors.white, size:
iconSize),
        tooltip: 'Configuración',
        onPressed: () {
          Navigator.pushNamed(
            context,
            AppRoutes.configuracionUsuarios,
            arguments: widget.vendedor,
          );
        },
      ),
      IconButton(
```

```
      icon: Icon(Icons.logout, color: Colors.white, size: iconSize),
      tooltip: 'Cerrar sesión',
      onPressed: () {
        _handleLogout(context);
      },
    ),
    SizedBox(width: isTablet ? 8 : 4),
  ];
}

List<Tab> _buildTabs(bool isTablet) {
  return [
    // ELIMINADO: Tab de Nueva Venta - Vendedor ya no puede crear ventas
    Tab(
      icon: const Icon(Icons.receipt_long),
      text: 'Ventas',
      height: isTablet ? 72 : 64,
    ),
    Tab(
      icon: const Icon(Icons.assignment),
      text: 'Pedidos',
      height: isTablet ? 72 : 64,
    ),
    Tab(
      icon: const Icon(Icons.people),
      text: 'Clientes',
      height: isTablet ? 72 : 64,
    ),
    Tab(
      icon: const Icon(Icons.inventory),
      text: 'Productos',
      height: isTablet ? 72 : 64,
    ),
  ];
}

Widget _buildDrawer(BuildContext context) {
  return Drawer(
    backgroundColor: const Color(0xFF2A2A2A),
    child: ListView(
      padding: EdgeInsets.zero,
      children: [
        DrawerHeader(
          decoration: const BoxDecoration(
            color: Color(0xFF1A1A1A),
          ),
          child: Column(
            crossAxisAlignment: CrossAxisAlignment.start,
            mainAxisAlignment: MainAxisAlignment.end,
            children: [
              CircleAvatar(
                radius: 30,
                backgroundColor: Theme.of(context).colorScheme.primary,
                child: Text(
```

```
                    widget.vendedor.nombre.isNotEmpty ?
widget.vendedor.nombre[0].toUpperCase() : 'V',
                    style: const TextStyle(
                      fontSize: 24,
                      fontWeight: FontWeight.bold,
                      color: Colors.white,
                    ),
                  ),
                ),
                const SizedBox(height: 12),
                Text(
                  widget.vendedor.nombre,
                  style: const TextStyle(
                   color: Colors.white,
                    fontSize: 18,
                    fontWeight: FontWeight.bold,
                  ),
                  overflow: TextOverflow.ellipsis,
                ),
                Text(
                  'Vendedor',
                  style: TextStyle(
                    color: Colors.grey[400],
                    fontSize: 14,
                  ),
                ),
              ],
            ),
          ),
          // ELIMINADO: Opción de Nueva Venta del drawer
          _buildDrawerItem(
            icon: Icons.receipt_long,
            title: 'Ventas',
            onTap: () {
              Navigator.pop(context);
              _tabController.index = 0; // Ahora Ventas es el primer tab
            },
          ),
          _buildDrawerItem(
            icon: Icons.assignment,
            title: 'Pedidos',
            onTap: () {
              Navigator.pop(context);
              _tabController.index = 1; // Pedidos es el segundo tab
            },
          ),
          _buildDrawerItem(
            icon: Icons.people,
            title: 'Clientes',
            onTap: () {
              Navigator.pop(context);
              _tabController.index = 2; // Clientes es el tercer tab
            },
          ),
```

```
        _buildDrawerItem(
          icon: Icons.inventory,
          title: 'Productos',
          onTap: () {
            Navigator.pop(context);
            _tabController.index = 3; // Productos es el cuarto tab
          },
        ),
        const Divider(color: Colors.white24),
        _buildDrawerItem(
          icon: Icons.person_outline,
          title: 'Mi Perfil',
          onTap: () {
            Navigator.pop(context);
            Navigator.pushNamed(
              context,
              AppRoutes.perfilUsuario,
              arguments: widget.vendedor,
            );
          },
        ),
        _buildDrawerItem(
          icon: Icons.settings_outlined,
          title: 'Configuración',
          onTap: () {
            Navigator.pop(context);
            Navigator.pushNamed(
              context,
              AppRoutes.configuracionUsuarios,
              arguments: widget.vendedor,
            );
          },
        ),
        _buildDrawerItem(
          icon: Icons.logout,
          title: 'Cerrar sesión',
          onTap: () {
            Navigator.pop(context);
            _handleLogout(context);
          },
        ),
      ],
    ),
  );
}

Widget _buildDrawerItem({
  required IconData icon,
  required String title,
  required VoidCallback onTap,
}) {
  return ListTile(
    leading: Icon(icon, color: Colors.white70),
    title: Text(
```

```
        title,
        style: const TextStyle(color: Colors.white),
      ),
      onTap: onTap,
      hoverColor: Colors.white10,
    );
  }

  Widget _buildBody(
    BuildContext context,
    bool isMobile,
    bool isTablet,
    bool isDesktop,
    Orientation orientation,
  ) {
    if (isMobile) {
      // En móviles, usamos un BottomNavigationBar
      return Column(
        children: [
          Expanded(
            child: TabBarView(
              controller: _tabController,
              children: _buildTabViews(),
            ),
          ),
          _buildBottomNavigationBar(context),
        ],
      );
    } else {
      // En tablets y desktop, usamos TabBarView normal
      return Container(
        padding: EdgeInsets.all(isDesktop ? 24 : 16),
        child: TabBarView(
          controller: _tabController,
          children: _buildTabViews(),
        ),
      );
    }
  }

  List<Widget> _buildTabViews() {
    return [
      // ELIMINADO: VendedorCrearVenta - Vendedor ya no puede crear ventas
directas
      VendedorVentasTab(
        vendedorId: widget.vendedor.id!,
        onNuevaVentaTap: () {
          // Función deshabilitada - ya no hay tab de Nueva Venta
        },
      ),
      VendedorPedidosTab(vendedorId: widget.vendedor.id!),
      VendedorClientesTab(vendedorId: widget.vendedor.id!),
      const VendedorProductosTab(),
    ];
```

```
  }

  Widget _buildBottomNavigationBar(BuildContext context) {
    return Container(
      decoration: BoxDecoration(
        color: const Color(0xFF2A2A2A),
        boxShadow: [
          BoxShadow(
            color: Colors.black.withOpacity(0.3),
            blurRadius: 8,
            offset: const Offset(0, -2),
          ),
        ],
      ),
      child: SafeArea(
        child: TabBar(
          controller: _tabController,
          indicatorColor: Theme.of(context).colorScheme.primary,
          labelColor: Theme.of(context).colorScheme.primary,
          unselectedLabelColor: Colors.white60,
          labelStyle: const TextStyle(fontSize: 11, fontWeight:
FontWeight.w600),
          unselectedLabelStyle: const TextStyle(fontSize: 11),
          indicatorSize: TabBarIndicatorSize.tab,
          tabs: const [
            // ELIMINADO: Tab de Nueva Venta
            Tab(icon: Icon(Icons.receipt_long, size: 22), text: 'Ventas'),
            Tab(icon: Icon(Icons.assignment, size: 22), text: 'Pedidos'),
            Tab(icon: Icon(Icons.people, size: 22), text: 'Clientes'),
            Tab(icon: Icon(Icons.inventory, size: 22), text: 'Productos'),
          ],
        ),
      ),
    );
  }

  Widget? _buildFAB(BuildContext context, bool isMobile, bool isTablet) {
    // FAB eliminado - Vendedor ya no puede crear ventas directas
    return null;
  }

  FloatingActionButtonLocation _getFABLocation(bool isMobile) {
    return isMobile
        ? FloatingActionButtonLocation.endFloat
        : FloatingActionButtonLocation.endFloat;
  }

  void _handleLogout(BuildContext context) {
    showDialog(
      context: context,
      builder: (context) => AlertDialog(
        backgroundColor: const Color(0xFF2A2A2A),
        title: const Text(
          '¿Cerrar sesión?',
```

```
          style: TextStyle(color: Colors.white),
        ),
        content: const Text(
          '¿Estás seguro de que deseas cerrar sesión?',
          style: TextStyle(color: Colors.white70),
        ),
        actions: [
          TextButton(
            onPressed: () => Navigator.pop(context),
            child: const Text('Cancelar'),
          ),
          TextButton(
            onPressed: () {
              Navigator.pop(context);
              context.read<AuthService>().cerrarSesion();
              Navigator.pushNamedAndRemoveUntil(
                context,
                AppRoutes.home,
                (route) => false,
              );
            },
            child: const Text(
              'Cerrar sesión',
              style: TextStyle(color: Colors.red),
            ),
          ),
        ],
      ),
    );
  }
}
```

Vendedor_clientes_tab.dart

```
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import '../../data/models/usuario.dart';
import '../../viewmodels/cliente_viewmodel.dart';

/// TAB DE CLIENTES - VENDEDOR (CORREGIDO)
/// Gestiona los clientes del vendedor (solo lectura)
class VendedorClientesTab extends StatefulWidget {
  final int vendedorId;

  const VendedorClientesTab({
    Key? key,
    required this.vendedorId,
  }) : super(key: key);

  @override
  State<VendedorClientesTab> createState() => _VendedorClientesTabState();
}

class _VendedorClientesTabState extends State<VendedorClientesTab> {
  final _busquedaController = TextEditingController();
```

```
bool _soloActivos = true;
bool _isInitialized = false;

@override
void initState() {
  super.initState();
  _busquedaController.addListener(() {
    if (mounted) setState(() {});
  });
}

@override
void didChangeDependencies() {
  super.didChangeDependencies();
  if (!_isInitialized) {
    _isInitialized = true;
    // Cargar datos de forma segura
    Future.microtask(() {
      if (mounted) {
        context.read<ClienteViewModel>().cargarClientes();
      }
    });
  }
}

@override
void dispose() {
  _busquedaController.dispose();
  super.dispose();
}

Map<String, dynamic> _getResponsiveSizes(double width) {
  if (width > 1024) {
    return {
      'fontSize': 16.0,
      'titleSize': 22.0,
      'padding': 24.0,
      'spacing': 16.0,
      'iconSize': 24.0,
      'cardPadding': 16.0,
      'avatarSize': 60.0,
    };
  } else if (width > 600) {
    return {
      'fontSize': 15.0,
      'titleSize': 20.0,
      'padding': 20.0,
      'spacing': 14.0,
      'iconSize': 22.0,
      'cardPadding': 14.0,
      'avatarSize': 56.0,
    };
  } else if (width > 360) {
    return {
```

```
      'fontSize': 14.0,
      'titleSize': 18.0,
      'padding': 16.0,
      'spacing': 12.0,
      'iconSize': 20.0,
      'cardPadding': 12.0,
      'avatarSize': 52.0,
    };
  } else {
    return {
      'fontSize': 13.0,
      'titleSize': 16.0,
      'padding': 12.0,
      'spacing': 10.0,
      'iconSize': 18.0,
      'cardPadding': 10.0,
      'avatarSize': 48.0,
    };
  }
}

List<Usuario> _filtrarClientes(List<Usuario> clientes) {
  try {
    var filtrados = List<Usuario>.from(clientes);

    if (_soloActivos) {
      filtrados = filtrados.where((c) => c.activo).toList();
    }

    final termino = _busquedaController.text.trim().toLowerCase();
    if (termino.isNotEmpty) {
      filtrados = filtrados.where((c) {
        final nombre = c.nombre.toLowerCase();
        final email = c.email.toLowerCase();
        final telefono = (c.telefono ?? '').toLowerCase();
        return nombre.contains(termino) ||
            email.contains(termino) ||
            telefono.contains(termino);
      }).toList();
    }

    return filtrados;
  } catch (e) {
    debugPrint('Error al filtrar clientes: $e');
    return [];
  }
}

@override
Widget build(BuildContext context) {
  return Consumer<ClienteViewModel>(
    builder: (context, viewModel, child) {
      final clientesFiltrados = _filtrarClientes(viewModel.clientes);
```

```
      return LayoutBuilder(
        builder: (context, constraints) {
          final width = constraints.maxWidth;
          final sizes = _getResponsiveSizes(width);
          final isMobile = width < 600;
          final isDesktop = width >= 1024;

          return Column(
            children: [
              _buildHeader(viewModel, sizes, isMobile),
              _buildFiltros(viewModel, clientesFiltrados, sizes, isMobile),
              Expanded(
                child: _buildListaClientes(
                  viewModel,
                  clientesFiltrados,
                  sizes,
                  isMobile,
                  isDesktop,
                ),
              ),
            ],
          );
        },
      );
    },
  );
}

// ========================================
// HEADER
// ========================================
Widget _buildHeader(
  ClienteViewModel viewModel,
  Map<String, dynamic> sizes,
  bool isMobile,
) {
  return Container(
    padding: EdgeInsets.all(sizes['padding']),
    decoration: BoxDecoration(
      gradient: const LinearGradient(
        colors: [Color(0xFF1A1A1A), Color(0xFF2A2A2A)],
      ),
      boxShadow: [
        BoxShadow(
          color: Colors.black.withOpacity(0.3),
          blurRadius: 8,
          offset: const Offset(0, 2),
        ),
      ],
    ),
    child: Row(
      children: [
        Container(
          padding: EdgeInsets.all(sizes['spacing']),
```

```
  decoration: BoxDecoration(
    gradient: const LinearGradient(
      colors: [Color(0xFF27AE60), Color(0xFF2ECC71)],
    ),
    borderRadius: BorderRadius.circular(12),
    boxShadow: [
      BoxShadow(
        color: const Color(0xFF27AE60).withOpacity(0.3),
        blurRadius: 8,
      ),
    ],
  ),
  child: Icon(
    Icons.people_rounded,
    color: Colors.white,
    size: sizes['iconSize'] + 4,
  ),
),
SizedBox(width: sizes['spacing']),
Expanded(
  child: Column(
    crossAxisAlignment: CrossAxisAlignment.start,
    children: [
      Text(
        'Mis Clientes',
        style: TextStyle(
          color: Colors.white,
          fontSize: sizes['titleSize'],
          fontWeight: FontWeight.bold,
        ),
      ),
      if (!isMobile) ...[
        SizedBox(height: sizes['spacing'] / 3),
        Text(
          'Consulta información de tus clientes',
          style: TextStyle(
            color: Colors.grey[400],
            fontSize: sizes['fontSize'] - 1,
          ),
        ),
      ],
    ],
  ),
),
IconButton(
  onPressed: () {
    if (mounted) {
      viewModel.cargarClientes();
    }
  },
  icon: const Icon(Icons.refresh_rounded),
  color: const Color(0xFF27AE60),
  tooltip: 'Actualizar',
  iconSize: sizes['iconSize'],
```

```
        ),
      ],
    ),
  );
}

// ========================================
// FILTROS
// ========================================
Widget _buildFiltros(
  ClienteViewModel viewModel,
  List<Usuario> clientesFiltrados,
  Map<String, dynamic> sizes,
  bool isMobile,
) {
  return Container(
    padding: EdgeInsets.all(sizes['padding']),
    decoration: BoxDecoration(
      color: const Color(0xFF1A1A1A),
      boxShadow: [
        BoxShadow(
          color: Colors.black.withOpacity(0.2),
          blurRadius: 8,
          offset: const Offset(0, 2),
        ),
      ],
    ),
    child: Column(
      children: [
        // Barra de búsqueda
        Container(
          decoration: BoxDecoration(
            color: const Color(0xFF2A2A2A),
            borderRadius: BorderRadius.circular(12),
            border: Border.all(color: Colors.white.withOpacity(0.1)),
          ),
          child: TextField(
            controller: _busquedaController,
            style: TextStyle(
              color: Colors.white,
              fontSize: sizes['fontSize'],
            ),
            decoration: InputDecoration(
              hintText: 'Buscar por nombre, email o teléfono...',
              hintStyle: TextStyle(
                color: Colors.grey[600],
                fontSize: sizes['fontSize'] - 1,
              ),
              prefixIcon: Icon(
                Icons.search_rounded,
                color: Colors.grey[600],
                size: sizes['iconSize'],
              ),
              suffixIcon: _busquedaController.text.isNotEmpty
```

```
          ? IconButton(
              icon: Icon(
                Icons.clear_rounded,
                size: sizes['iconSize'] - 2,
              ),
              color: Colors.grey[600],
              onPressed: () {
                _busquedaController.clear();
              },
            )
          : null,
      border: InputBorder.none,
      contentPadding: EdgeInsets.symmetric(
        horizontal: sizes['spacing'],
        vertical: sizes['spacing'] * 0.8,
      ),
    ),
  ),
),

SizedBox(height: sizes['spacing']),

// Filtro de activos/inactivos y contador
Row(
  children: [
    Expanded(
      child: InkWell(
        onTap: () {
          if (mounted) {
            setState(() => _soloActivos = !_soloActivos);
          }
        },
        borderRadius: BorderRadius.circular(8),
        child: Container(
          padding: EdgeInsets.symmetric(
            horizontal: sizes['spacing'],
            vertical: sizes['spacing'] * 0.7,
          ),
          decoration: BoxDecoration(
            color: _soloActivos
                ? const Color(0xFF27AE60).withOpacity(0.2)
                : const Color(0xFF2A2A2A),
            borderRadius: BorderRadius.circular(8),
            border: Border.all(
              color: _soloActivos
                  ? const Color(0xFF27AE60)
                  : Colors.grey[700]!,
              width: 1.5,
            ),
          ),
          child: Row(
            mainAxisSize: MainAxisSize.min,
            children: [
              Icon(
```

```
                        _soloActivos
                            ? Icons.check_circle_rounded
                            : Icons.all_inclusive_rounded,
                        size: sizes['iconSize'] - 2,
                        color: _soloActivos ? Colors.white :
Colors.grey[400],
                      ),
                      SizedBox(width: sizes['spacing'] / 2),
                      Text(
                        _soloActivos ? 'Solo Activos' : 'Todos',
                        style: TextStyle(
                          color:
                              _soloActivos ? Colors.white :
Colors.grey[400],
                          fontWeight: _soloActivos
                              ? FontWeight.bold
                              : FontWeight.w500,
                          fontSize: sizes['fontSize'] - 1,
                        ),
                      ),
                    ],
                  ),
                ),
              ),
            ),
            if (clientesFiltrados.length != viewModel.clientes.length) ...[
              SizedBox(width: sizes['spacing']),
              Container(
                padding: EdgeInsets.symmetric(
                  horizontal: sizes['spacing'],
                  vertical: sizes['spacing'] * 0.5,
                ),
                decoration: BoxDecoration(
                  color: const Color(0xFF27AE60).withOpacity(0.15),
                  borderRadius: BorderRadius.circular(8),
                  border: Border.all(
                    color: const Color(0xFF27AE60).withOpacity(0.3),
                  ),
                ),
                child: Row(
                  mainAxisSize: MainAxisSize.min,
                  children: [
                    Icon(
                      Icons.filter_list_rounded,
                      size: sizes['iconSize'] - 4,
                      color: const Color(0xFF27AE60),
                    ),
                    SizedBox(width: sizes['spacing'] / 2),
                    Text(

'${clientesFiltrados.length}/${viewModel.clientes.length}',
                      style: TextStyle(
                        fontSize: sizes['fontSize'] - 1,
                        color: Colors.white,
```

```
                        fontWeight: FontWeight.w600,
                      ),
                    ),
                  ],
                ),
              ),
            ],
          ),
        ),
      ],
    ),
  );
}

// ======================================
// LISTA DE CLIENTES
// ======================================
Widget _buildListaClientes(
  ClienteViewModel viewModel,
  List<Usuario> clientesFiltrados,
  Map<String, dynamic> sizes,
  bool isMobile,
  bool isDesktop,
) {
  if (viewModel.cargando) {
    return const Center(
      child: CircularProgressIndicator(color: Color(0xFF27AE60)),
    );
  }

  if (viewModel.error != null) {
    return Center(
      child: Padding(
        padding: EdgeInsets.all(sizes['padding']),
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            Icon(
              Icons.error_outline_rounded,
              size: sizes['iconSize'] * 3,
              color: Colors.red[400],
            ),
            SizedBox(height: sizes['spacing']),
            Text(
              'Error al cargar clientes',
              style: TextStyle(
                fontSize: sizes['titleSize'] - 2,
                fontWeight: FontWeight.w600,
                color: Colors.grey[400],
              ),
            ),
            SizedBox(height: sizes['spacing'] / 2),
            Text(
              viewModel.error!,
```

```
            style: TextStyle(
              fontSize: sizes['fontSize'] - 1,
              color: Colors.grey[500],
            ),
            textAlign: TextAlign.center,
          ),
          SizedBox(height: sizes['spacing'] * 1.5),
          ElevatedButton.icon(
            onPressed: () {
              if (mounted) {
                viewModel.cargarClientes();
              }
            },
            icon: const Icon(Icons.refresh_rounded),
            label: const Text('Reintentar'),
            style: ElevatedButton.styleFrom(
              backgroundColor: const Color(0xFF27AE60),
              foregroundColor: Colors.white,
              padding: EdgeInsets.symmetric(
                horizontal: sizes['spacing'] * 1.5,
                vertical: sizes['spacing'] * 0.8,
              ),
            ),
          ),
        ],
      ),
    ),
  );
}

if (clientesFiltrados.isEmpty) {
  return Center(
    child: Column(
      mainAxisAlignment: MainAxisAlignment.center,
      children: [
        Icon(
          _busquedaController.text.isNotEmpty
              ? Icons.search_off_rounded
              : Icons.people_outline_rounded,
          size: sizes['iconSize'] * 3.5,
          color: Colors.grey[700],
        ),
        SizedBox(height: sizes['spacing'] * 1.5),
        Text(
          _busquedaController.text.isNotEmpty
              ? 'No se encontraron clientes'
              : 'No hay clientes registrados',
          style: TextStyle(
            fontSize: sizes['titleSize'] - 2,
            fontWeight: FontWeight.w600,
            color: Colors.grey[400],
          ),
        ),
        SizedBox(height: sizes['spacing'] / 2),
```

```
          Text(
            _busquedaController.text.isNotEmpty
                ? 'Intenta con otros términos'
                : 'Los clientes aparecerán aquí',
            style: TextStyle(
              fontSize: sizes['fontSize'],
              color: Colors.grey[500],
            ),
          ),
        ],
      ),
    );
  }

  return RefreshIndicator(
    onRefresh: () async {
      if (mounted) {
        await viewModel.cargarClientes();
      }
    },
    color: const Color(0xFF27AE60),
    child: isDesktop
        ? _buildGridView(clientesFiltrados, sizes)
        : _buildListView(clientesFiltrados, sizes),
  );
}

Widget _buildListView(List<Usuario> clientes, Map<String, dynamic> sizes) {
  return ListView.builder(
    padding: EdgeInsets.all(sizes['padding']),
    itemCount: clientes.length,
    itemBuilder: (context, index) {
      return _buildClienteCard(clientes[index], sizes);
    },
  );
}

Widget _buildGridView(List<Usuario> clientes, Map<String, dynamic> sizes) {
  return GridView.builder(
    padding: EdgeInsets.all(sizes['padding']),
    gridDelegate: const SliverGridDelegateWithMaxCrossAxisExtent(
      maxCrossAxisExtent: 400,
      childAspectRatio: 2.5,
      crossAxisSpacing: 16,
      mainAxisSpacing: 16,
    ),
    itemCount: clientes.length,
    itemBuilder: (context, index) {
      return _buildClienteCard(clientes[index], sizes);
    },
  );
}

// ======================================
```

```
// CARD DE CLIENTE
// ========================================
Widget _buildClienteCard(Usuario cliente, Map<String, dynamic> sizes) {
  return Container(
    margin: EdgeInsets.only(bottom: sizes['spacing']),
    decoration: BoxDecoration(
      color: const Color(0xFF1A1A1A),
      borderRadius: BorderRadius.circular(16),
      border: Border.all(
        color: Colors.white.withOpacity(0.1),
      ),
      boxShadow: [
        BoxShadow(
          color: Colors.black.withOpacity(0.2),
          blurRadius: 8,
          offset: const Offset(0, 2),
        ),
      ],
    ),
    child: Material(
      color: Colors.transparent,
      child: InkWell(
        borderRadius: BorderRadius.circular(16),
        onTap: () => _verDetalle(cliente),
        child: Padding(
          padding: EdgeInsets.all(sizes['cardPadding']),
          child: Row(
            children: [
              // Avatar
              Container(
                width: sizes['avatarSize'],
                height: sizes['avatarSize'],
                decoration: BoxDecoration(
                  gradient: LinearGradient(
                    colors: [
                      const Color(0xFF27AE60).withOpacity(0.3),
                      const Color(0xFF27AE60).withOpacity(0.1),
                    ],
                  ),
                  borderRadius: BorderRadius.circular(12),
                  border: Border.all(
                    color: const Color(0xFF27AE60),
                    width: 2,
                  ),
                ),
                child: Center(
                  child: Text(
                    cliente.nombre.isNotEmpty
                        ? cliente.nombre[0].toUpperCase()
                        : '?',
                    style: TextStyle(
                      color: const Color(0xFF27AE60),
                      fontSize: sizes['titleSize'],
                      fontWeight: FontWeight.bold,
```

```
      ),
    ),
  ),
),
SizedBox(width: sizes['spacing']),

// Información del cliente
Expanded(
  child: Column(
    crossAxisAlignment: CrossAxisAlignment.start,
    mainAxisSize: MainAxisSize.min,
    children: [
      Row(
        children: [
          Expanded(
            child: Text(
              cliente.nombre,
              style: TextStyle(
                color: Colors.white,
                fontSize: sizes['fontSize'],
                fontWeight: FontWeight.w600,
              ),
              maxLines: 1,
              overflow: TextOverflow.ellipsis,
            ),
          ),
          if (!cliente.activo)
            Container(
              margin: const EdgeInsets.only(left: 8),
              padding: const EdgeInsets.symmetric(
                horizontal: 8,
                vertical: 2,
              ),
              decoration: BoxDecoration(
                color: Colors.red.withOpacity(0.2),
                borderRadius: BorderRadius.circular(4),
              ),
              child: Text(
                'Inactivo',
                style: TextStyle(
                  color: Colors.red[300],
                  fontSize: sizes['fontSize'] - 4,
                  fontWeight: FontWeight.w600,
                ),
              ),
            ),
        ],
      ),
      SizedBox(height: sizes['spacing'] / 3),
      // Email
      Row(
        children: [
          Icon(
            Icons.email_outlined,
```

```
                      size: sizes['iconSize'] - 6,
                      color: Colors.grey[500],
                    ),
                    SizedBox(width: sizes['spacing'] / 2),
                    Expanded(
                      child: Text(
                        cliente.email,
                        style: TextStyle(
                          color: Colors.grey[400],
                          fontSize: sizes['fontSize'] - 2,
                        ),
                        maxLines: 1,
                        overflow: TextOverflow.ellipsis,
                      ),
                    ),
                  ],
                ),
                // Teléfono (si existe)
                if (cliente.telefono != null &&
                    cliente.telefono!.isNotEmpty) ...[
                  SizedBox(height: sizes['spacing'] / 4),
                  Row(
                    children: [
                      Icon(
                        Icons.phone_outlined,
                        size: sizes['iconSize'] - 6,
                        color: Colors.grey[500],
                      ),
                      SizedBox(width: sizes['spacing'] / 2),
                      Text(
                        cliente.telefono!,
                        style: TextStyle(
                          color: Colors.grey[400],
                          fontSize: sizes['fontSize'] - 2,
                        ),
                      ),
                    ],
                  ),
                ],
              ],
            ),
          ),

          // Flecha de navegación
          Icon(
            Icons.chevron_right_rounded,
            color: Colors.grey[600],
            size: sizes['iconSize'] + 4,
          ),
        ],
      ),
    ),
  ),
),
```

```
  );
}

void _verDetalle(Usuario cliente) {
  showModalBottomSheet(
    context: context,
    backgroundColor: Colors.transparent,
    isScrollControlled: true,
    builder: (context) => _buildDetalleCliente(cliente),
  );
}

Widget _buildDetalleCliente(Usuario cliente) {
  final sizes = _getResponsiveSizes(MediaQuery.of(context).size.width);

  return Container(
    margin: const EdgeInsets.all(16),
    decoration: BoxDecoration(
      color: const Color(0xFF1A1A1A),
      borderRadius: BorderRadius.circular(20),
      border: Border.all(
        color: Colors.white.withOpacity(0.1),
      ),
    ),
    child: Column(
      mainAxisSize: MainAxisSize.min,
      children: [
        // Handle
        Container(
          margin: const EdgeInsets.only(top: 12),
          width: 40,
          height: 4,
          decoration: BoxDecoration(
            color: Colors.grey[600],
            borderRadius: BorderRadius.circular(2),
          ),
        ),

        Padding(
          padding: EdgeInsets.all(sizes['padding']),
          child: Column(
            children: [
              // Avatar grande
              Container(
                width: sizes['avatarSize'] * 1.5,
                height: sizes['avatarSize'] * 1.5,
                decoration: BoxDecoration(
                  gradient: const LinearGradient(
                    colors: [Color(0xFF27AE60), Color(0xFF2ECC71)],
                  ),
                  borderRadius: BorderRadius.circular(20),
                ),
                child: Center(
                  child: Text(
```

```
                      cliente.nombre.isNotEmpty
                          ? cliente.nombre[0].toUpperCase()
                          : '?',
                      style: TextStyle(
                        color: Colors.white,
                        fontSize: sizes['titleSize'] * 1.5,
                        fontWeight: FontWeight.bold,
                      ),
                    ),
                  ),
                ),

                SizedBox(height: sizes['spacing']),

                // Nombre
                Text(
                  cliente.nombre,
                  style: TextStyle(
                    color: Colors.white,
                    fontSize: sizes['titleSize'],
                    fontWeight: FontWeight.bold,
                  ),
                  textAlign: TextAlign.center,
                ),

                SizedBox(height: sizes['spacing'] * 2),

                // Información de contacto
                _buildInfoRow(
                  Icons.email_outlined,
                  'Email',
                  cliente.email,
                  sizes,
                ),
                if (cliente.telefono != null && cliente.telefono!.isNotEmpty)
                  _buildInfoRow(
                    Icons.phone_outlined,
                    'Teléfono',
                    cliente.telefono!,
                    sizes,
                  ),
                if (cliente.direccion != null &&
cliente.direccion!.isNotEmpty)
                  _buildInfoRow(
                    Icons.location_on_outlined,
                    'Dirección',
                    cliente.direccion!,
                    sizes,
                  ),

                SizedBox(height: sizes['spacing'] * 2),

                // Botón cerrar
                SizedBox(
```

```
                width: double.infinity,
                child: ElevatedButton(
                  onPressed: () => Navigator.pop(context),
                  style: ElevatedButton.styleFrom(
                    backgroundColor: const Color(0xFF27AE60),
                    foregroundColor: Colors.white,
                    padding: EdgeInsets.symmetric(
                      vertical: sizes['spacing'],
                    ),
                    shape: RoundedRectangleBorder(
                      borderRadius: BorderRadius.circular(12),
                    ),
                  ),
                  child: Text(
                    'Cerrar',
                    style: TextStyle(
                      fontSize: sizes['fontSize'],
                      fontWeight: FontWeight.w600,
                    ),
                  ),
                ),
              ),
            ],
          ),
        ),
      ],
    ),
  );
}

Widget _buildInfoRow(
  IconData icon,
  String label,
  String value,
  Map<String, dynamic> sizes,
) {
  return Padding(
    padding: EdgeInsets.only(bottom: sizes['spacing']),
    child: Row(
      children: [
        Container(
          padding: EdgeInsets.all(sizes['spacing'] / 2),
          decoration: BoxDecoration(
            color: const Color(0xFF27AE60).withOpacity(0.15),
            borderRadius: BorderRadius.circular(8),
          ),
          child: Icon(
            icon,
            color: const Color(0xFF27AE60),
            size: sizes['iconSize'],
          ),
        ),
        SizedBox(width: sizes['spacing']),
        Expanded(
```

```
            child: Column(
              crossAxisAlignment: CrossAxisAlignment.start,
              children: [
                Text(
                  label,
                  style: TextStyle(
                    color: Colors.grey[500],
                    fontSize: sizes['fontSize'] - 2,
                  ),
                ),
                Text(
                  value,
                  style: TextStyle(
                    color: Colors.white,
                    fontSize: sizes['fontSize'],
                  ),
                ),
              ],
            ),
          ),
        ],
      ),
    );
  }
}
```

vendedor_crear_venta.dart

```
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import '../../data/models/producto.dart';
import '../../data/models/usuario.dart';
import '../../data/models/venta.dart';
import '../../data/models/detalle_venta.dart';
import '../../viewmodels/venta_viewmodel.dart';
import '../../viewmodels/usuario_viewmodel.dart';
import '../../viewmodels/producto_viewmodel.dart';
import '../../core/constants.dart';

class VendedorCrearVenta extends StatefulWidget {
  final int vendedorId;
  final bool showAppBar;

  const VendedorCrearVenta({
    Key? key,
    required this.vendedorId,
    this.showAppBar = true,
  }) : super(key: key);

  @override
  State<VendedorCrearVenta> createState() => _VendedorCrearVentaState();
}

class _VendedorCrearVentaState extends State<VendedorCrearVenta> {
  List<ItemCarrito> carrito = [];
```

```
Usuario? clienteSeleccionado;
String metodoPago = 'efectivo';
final observacionesController = TextEditingController();

List<Producto> productos = [];
List<Producto> productosFiltrados = [];

final busquedaController = TextEditingController();
int cantidadSeleccionada = 1;

bool cargando = true;
String? error;

@override
void initState() {
  super.initState();
  _cargarProductos();
  WidgetsBinding.instance.addPostFrameCallback((_) {
    if (mounted) {
      context.read<UsuarioViewModel>().cargarClientes();
    }
  });
}

Future<void> _cargarProductos() async {
  try {
    setState(() => cargando = true);

    // Cargar productos reales desde el ViewModel
    final productoVM = context.read<ProductoViewModel>();
    await productoVM.cargarProductos();
    productos = productoVM.productos;

    productosFiltrados = productos;
    setState(() => cargando = false);
  } catch (e) {
    setState(() {
      error = 'Error al cargar productos: $e';
      cargando = false;
    });
  }
}

void _filtrarProductos(String texto) {
  if (texto.isEmpty) {
    setState(() => productosFiltrados = productos);
    return;
  }

  setState(() {
    productosFiltrados = productos
        .where((p) =>
            p.nombre.toLowerCase().contains(texto.toLowerCase()) ||
            p.codigo.toLowerCase().contains(texto.toLowerCase()))
```

```
          .toList();
    });
  }

  void _agregarAlCarrito(Producto producto) {
    if (cantidadSeleccionada <= 0) {
      ScaffoldMessenger.of(context).showSnackBar(
        const SnackBar(content: Text('Ingresa una cantidad válida')),
      );
      return;
    }

    if (cantidadSeleccionada > producto.stock) {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(
            content: Text('Stock insuficiente. Disponible:
${producto.stock}')),
      );
      return;
    }

    final indice =
        carrito.indexWhere((item) => item.producto.id == producto.id);

    if (indice != -1) {
      final nuevoCarrito = carrito;
      nuevoCarrito[indice] = ItemCarrito(
        producto: producto,
        cantidad: nuevoCarrito[indice].cantidad + cantidadSeleccionada,
      );
      setState(() => carrito = nuevoCarrito);
    } else {
      setState(() {
        carrito.add(ItemCarrito(
          producto: producto,
          cantidad: cantidadSeleccionada,
        ));
      });
    }

    setState(() => cantidadSeleccionada = 1);
    busquedaController.clear();

    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(
        content: Text('${producto.nombre} agregado al carrito'),
        duration: const Duration(seconds: 1),
      ),
    );
  }

  void _eliminarDelCarrito(int index) {
    setState(() => carrito.removeAt(index));
  }
```

```
  void _modificarCantidad(int index, int nuevaCantidad) {
    if (nuevaCantidad <= 0) {
      _eliminarDelCarrito(index);
      return;
    }

    if (nuevaCantidad > carrito[index].producto.stock) {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(
          content: Text('Stock insuficiente:
${carrito[index].producto.stock}'),
        ),
      );
      return;
    }

    setState(() {
      carrito[index] = ItemCarrito(
        producto: carrito[index].producto,
        cantidad: nuevaCantidad,
      );
    });
  }

  double get subtotal => carrito.fold(
      0.0, (sum, item) => sum + (item.producto.precioVenta * item.cantidad));

  double get descuentoTotal => 0.0;

  double get impuesto => subtotal * AppConstants.ivaEcuador;

  double get total => subtotal + impuesto - descuentoTotal;

  Future<void> _guardarVenta() async {
    if (carrito.isEmpty) {
      ScaffoldMessenger.of(context).showSnackBar(
        const SnackBar(content: Text('El carrito está vacío')),
      );
      return;
    }

    // Mostrar diálogo de confirmación con los datos
    showDialog(
      context: context,
      barrierDismissible: false,
      builder: (context) => AlertDialog(
        backgroundColor: const Color(0xFF2A2A2A),
        title: const Text(
          'Confirmar Venta',
          style: TextStyle(color: Colors.white, fontWeight: FontWeight.bold),
        ),
        content: SingleChildScrollView(
          child: Column(
```

```
              mainAxisSize: MainAxisSize.min,
              crossAxisAlignment: CrossAxisAlignment.start,
              children: [
                Text(
                  'Resumen de la Venta:',
                  style: TextStyle(
                    color: Colors.grey[400],
                    fontWeight: FontWeight.bold,
                    fontSize: 16,
                  ),
                ),
                const SizedBox(height: 12),
                ...carrito.map((item) => Padding(
                      padding: const EdgeInsets.symmetric(vertical: 4),
                      child: Row(
                        mainAxisAlignment: MainAxisAlignment.spaceBetween,
                        children: [
                          Expanded(
                            child: Text(
                              '${item.producto.nombre} x ${item.cantidad}',
                              style: const TextStyle(color: Colors.white),
                              overflow: TextOverflow.ellipsis,
                            ),
                          ),
                          Text(
                            '\$${(item.producto.precioVenta *
item.cantidad).toStringAsFixed(2)}',
                            style: const TextStyle(color: Color(0xFFE74C3C)),
                          ),
                        ],
                      ),
                    )),
                const Divider(color: Color(0xFFE74C3C), height: 16),
                Row(
                  mainAxisAlignment: MainAxisAlignment.spaceBetween,
                  children: [
                    const Text(
                      'Subtotal:',
                      style: TextStyle(color: Colors.grey),
                    ),
                    Text(
                      '\$${subtotal.toStringAsFixed(2)}',
                      style: const TextStyle(color: Colors.white),
                    ),
                  ],
                ),
                const SizedBox(height: 4),
                Row(
                  mainAxisAlignment: MainAxisAlignment.spaceBetween,
                  children: [
                    const Text(
                      'Impuesto (12%):',
                      style: TextStyle(color: Colors.grey),
                    ),
```

```
            Text(
              '\$${impuesto.toStringAsFixed(2)}',
              style: const TextStyle(color: Colors.white),
            ),
          ],
        ),
        const SizedBox(height: 8),
        Row(
          mainAxisAlignment: MainAxisAlignment.spaceBetween,
          children: [
            const Text(
              'Total:',
              style: TextStyle(
                color: Color(0xFFE74C3C),
                fontWeight: FontWeight.bold,
                fontSize: 16,
              ),
            ),
            Text(
              '\$${total.toStringAsFixed(2)}',
              style: const TextStyle(
                color: Color(0xFFE74C3C),
                fontWeight: FontWeight.bold,
                fontSize: 16,
              ),
            ),
          ],
        ),
        const SizedBox(height: 12),
        Text(
          'Método de Pago: ${metodoPago.toUpperCase()}',
          style: const TextStyle(color: Colors.grey),
        ),
      ],
    ),
  ),
  actions: [
    TextButton(
      onPressed: () => Navigator.pop(context),
      child: const Text(
        'Cancelar',
        style: TextStyle(color: Colors.grey),
      ),
    ),
    ElevatedButton(
      onPressed: () {
        Navigator.pop(context);
        _procesarVenta();
      },
      style: ElevatedButton.styleFrom(
        backgroundColor: const Color(0xFFE74C3C),
        foregroundColor: Colors.white,
      ),
      child: const Text('Confirmar y Guardar'),
```

```
          ),
        ],
      ),
    );
  }

  Future<void> _procesarVenta() async {
    try {
      print(

'\n╔═══════════════════════════════════════════════════════════════════════════╗');
      print('║INICIANDO PROCESO DE VENTA
║');
      print(

'╚═══════════════════════════════════════════════════════════════════════════╝\n');

      // Mostrar indicador de carga
      showDialog(
        context: context,
        barrierDismissible: false,
        builder: (context) => const Center(
          child: CircularProgressIndicator(color: Color(0xFFE74C3C)),
        ),
      );

      // Importar Provider para acceder a VentaViewModel
      final ventaViewModel = context.read<VentaViewModel>();

      print('🛍️ Información de la venta:');
      print('   - Vendedor ID: ${widget.vendedorId}');
      print(
          '   - Cliente: ${clienteSeleccionado?.nombre ?? "Venta
Mostrador"}');
      print('   - Subtotal: $subtotal');
      print('   - Impuesto: $impuesto');
      print('   - Descuento: $descuentoTotal');
      print('   - Total: $total');
      print('   - Método de pago: $metodoPago');
      print('   - Items en carrito: ${carrito.length}\n');

      // Crear objeto Venta con datos del carrito
      final detalles = carrito.map((item) {
        final subtotalItem = item.producto.precioVenta * item.cantidad;
        print('   🎁 Producto: ${item.producto.nombre}');
        print('      - Cantidad: ${item.cantidad}');
        print('      - Precio: ${item.producto.precioVenta}');
        print('      - Subtotal: $subtotalItem\n');

        return DetalleVenta(
          id: null,
          ventaId: 0, // Se asignará en BD
          productoId: item.producto.id ?? 0,
          cantidad: item.cantidad,
```

```
          precioUnitario: item.producto.precioVenta,
          subtotal: subtotalItem,
          descuento: 0,
          total: subtotalItem,
          nombreProducto: item.producto.nombre,
          codigoProducto: item.producto.codigo,
        );
      }).toList();

      // Variables para el cliente final
      int? clienteIdVenta = clienteSeleccionado?.id;
      String nombreClienteVenta = clienteSeleccionado?.nombre ?? 'Venta
Mostrador';

      // Si es Venta Mostrador (clienteSeleccionado es null), buscar o crear
Consumidor Final
      if (clienteSeleccionado == null) {
        print('Q Venta Mostrador detectada. Buscando usuario "Consumidor
Final"...');

        final usuarioVM = context.read<UsuarioViewModel>();

        // Buscar si ya existe "Consumidor Final" en la lista cargada
        // Validamos también por rol para asegurarnos que sea un cliente
        final consumidorFinalExistente = usuarioVM.usuarios.where((u) =>
            u.nombre.toLowerCase().contains('consumidor final') &&
            (u.rol == 'cliente' || u.rol == 'usuario')
        ).firstOrNull;

        if (consumidorFinalExistente != null) {
          print('✅ Usuario "Consumidor Final" encontrado (ID:
${consumidorFinalExistente.id})');
          clienteIdVenta = consumidorFinalExistente.id;
        } else {
          print('⚠ "Consumidor Final" no encontrado. Intentando crear nuevo
usuario...');

          try {
            // Crear usuario Consumidor Final
            final nuevoConsumidor = Usuario(
              nombre: 'Consumidor Final',
              email:
'consumidor_final_${DateTime.now().millisecondsSinceEpoch}@aztecafest.com',
// Email único
              password: 'generic_password_123', // Contraseña genérica (no se
usará)
              rol: 'cliente',
              telefono: '0000000000',
              direccion: 'Mostrador',
              fechaCreacion: DateTime.now(),
              activo: true,
            );

            // Usamos el ViewModel para crear (esto también recarga la lista)
```

```
            final creado = await usuarioVM.crearUsuario(nuevoConsumidor);

            if (creado) {
              // Buscar el usuario recién creado
              // Recargamos la lista localmente desde el VM actualizado
              final usuarioRecienCreado = usuarioVM.usuarios.where((u) =>
                  u.email == nuevoConsumidor.email
              ).firstOrNull;

              if (usuarioRecienCreado != null) {
                 print('✅ Nuevo "Consumidor Final" creado y recuperado (ID:
${usuarioRecienCreado.id})');
                 clienteIdVenta = usuarioRecienCreado.id;
              } else {
                 print('❌ Error: Usuario creado pero no encontrado en la
lista actualizada.');
              }
            } else {
               print('❌ Error al crear usuario "Consumidor Final":
${usuarioVM.error}');
            }
          } catch (e) {
            print('❌ Excepción al crear Consumidor Final: $e');
          }
        }

        // Si aún no tenemos ID (falló todo), usamos 0 o null y cruzamos los
dedos
        // o mostramos advertencia
        if (clienteIdVenta == null) {
           print('⚠️ ADVERTENCIA: No se pudo obtener ID válido para
Consumidor Final.');
        }
      }

      final venta = Venta(
        id: null,
        numeroVenta: 'V-${DateTime.now().millisecondsSinceEpoch}',
        vendedorId: widget.vendedorId,
        clienteId: clienteIdVenta, // ID resuelto (o null si falló todo)
        nombreCliente: nombreClienteVenta,
        fecha: DateTime.now(),
        subtotal: subtotal,
        impuesto: impuesto,
        descuento: descuentoTotal,
        total: total,
        metodoPago: metodoPago,
        estado: AppConstants.ventaCompletada,
        observaciones: observacionesController.text,
        detalles: detalles,
      );

      print('🔄 Guardando venta en BD...\n');
```

```
      // Guardar la venta en BD
      final resultado = await ventaViewModel.crearVenta(venta, detalles);

      if (!mounted) return;

      // Cerrar el diálogo de carga
      Navigator.pop(context);

      if (resultado) {
        print('\n✅ VENTA REGISTRADA EXITOSAMENTE\n');

        // Mostrar mensaje de éxito
        ScaffoldMessenger.of(context).showSnackBar(
          SnackBar(
            content: Row(
              children: const [
                Icon(Icons.check_circle, color: Colors.white),
                SizedBox(width: 12),
                Text('Venta registrada exitosamente'),
              ],
            ),
            backgroundColor: Colors.green[600],
            duration: const Duration(seconds: 2),
          ),
        );

        // Limpiar el carrito
        setState(() {
          carrito.clear();
          clienteSeleccionado = null;
          observacionesController.clear();
        });

        // Retroceder después de 2.5 segundos (tiempo suficiente para ver el
mensaje)
        Future.delayed(const Duration(milliseconds: 2500), () {
          if (mounted && Navigator.canPop(context)) {
            Navigator.pop(context);
          }
        });
      } else {
        print('\n❌ ERROR AL REGISTRAR LA VENTA\n');

        ScaffoldMessenger.of(context).showSnackBar(
          SnackBar(
            content: const Text('Error al registrar la venta'),
            backgroundColor: Colors.red[600],
            duration: const Duration(seconds: 2),
          ),
        );
      }
    } catch (e) {
      print('\n❌ EXCEPCIÓN: $e\n');
```

```
      if (!mounted) return;

      // Cerrar el diálogo de carga de forma segura
      if (Navigator.canPop(context)) {
        Navigator.pop(context);
      }

      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(
          content: Text('Error al guardar venta: $e'),
          backgroundColor: Colors.red[600],
          duration: const Duration(seconds: 2),
        ),
      );
    }
  }

  void _mostrarCarritoModal(BuildContext context) {
    showModalBottomSheet(
      context: context,
      backgroundColor: const Color(0xFF1A1A1A),
      isScrollControlled: true,
      builder: (context) => DraggableScrollableSheet(
        initialChildSize: 0.9,
        minChildSize: 0.5,
        maxChildSize: 0.95,
        expand: false,
        builder: (context, scrollController) => Column(
          children: [
            Container(
              padding: const EdgeInsets.all(16),
              decoration: const BoxDecoration(
                color: Color(0xFF2A2A2A),
                borderRadius: BorderRadius.vertical(top:
Radius.circular(16)),
              ),
              child: Row(
                mainAxisAlignment: MainAxisAlignment.spaceBetween,
                children: [
                  Text(
                    'Carrito (${carrito.length})',
                    style: const TextStyle(
                      color: Colors.white,
                      fontWeight: FontWeight.bold,
                      fontSize: 18,
                    ),
                  ),
                  IconButton(
                    icon: const Icon(Icons.close, color: Colors.white),
                    onPressed: () => Navigator.pop(context),
                  ),
                ],
              ),
```

```
),
Expanded(
  child: carrito.isEmpty
      ? Center(
          child: Text(
            'Carrito vacío',
            style: TextStyle(color: Colors.grey[600]),
          ),
        )
      : ListView.builder(
          controller: scrollController,
          itemCount: carrito.length,
          itemBuilder: (context, index) {
            final item = carrito[index];
            return _CarritoItem(
              item: item,
              onModificar: (cantidad) {
                _modificarCantidad(index, cantidad);
                setState(() {});
              },
              onEliminar: () {
                _eliminarDelCarrito(index);
                setState(() {});
              },
            );
          },
        ),
),
Container(
  padding: const EdgeInsets.all(16),
  decoration: const BoxDecoration(
    color: Color(0xFF2A2A2A),
    border: Border(
      top: BorderSide(color: Color(0xFFE74C3C), width: 1.5),
    ),
  ),
  child: Column(
    mainAxisSize: MainAxisSize.min,
    children: [
      _ResumenTotal(
        subtotal: subtotal,
        impuesto: impuesto,
        descuento: descuentoTotal,
        total: total,
      ),
      const SizedBox(height: 16),
      Text(
        'Método de Pago:',
        style: TextStyle(
          color: Colors.grey[400],
          fontWeight: FontWeight.bold,
        ),
      ),
      const SizedBox(height: 8),
```

```
                  DropdownButtonFormField<String>(
                    value: metodoPago,
                    items: const [
                      DropdownMenuItem(
                          value: 'efectivo', child: Text('Efectivo')),
                      DropdownMenuItem(
                          value: 'tarjeta', child: Text('Tarjeta')),
                      DropdownMenuItem(
                          value: 'transferencia', child:
Text('Transferencia')),
                    ],
                    onChanged: (valor) {
                      setState(() => metodoPago = valor!);
                    },
                    decoration: InputDecoration(
                      filled: true,
                      fillColor: const Color(0xFF1A1A1A),
                      border: OutlineInputBorder(
                        borderRadius: BorderRadius.circular(6),
                      ),
                    ),
                    dropdownColor: const Color(0xFF2A2A2A),
                  ),
                  const SizedBox(height: 12),
                  SizedBox(
                    width: double.infinity,
                    child: ElevatedButton(
                      onPressed: carrito.isNotEmpty ? _guardarVenta : null,
                      style: ElevatedButton.styleFrom(
                        backgroundColor: const Color(0xFFE74C3C),
                        disabledBackgroundColor: Colors.grey[700],
                        padding: const EdgeInsets.symmetric(vertical: 16),
                      ),
                      child:  const  Text(
                        'Guardar  Venta',
                        style: TextStyle(
                          color: Colors.white,
                          fontWeight: FontWeight.bold,
                          fontSize: 16,
                        ),
                      ),
                    ),
                  ),
                ],
              ),
            ),
          ],
        ),
      ),
    );
  }

  @override
  void dispose() {
```

```
  busquedaController.dispose();
  observacionesController.dispose();
  super.dispose();
}

@override
Widget build(BuildContext context) {
  final screenWidth = MediaQuery.of(context).size.width;
  final isTablet = screenWidth >= 600 && screenWidth < 1024;
  final isMobile = screenWidth < 600;

  return Scaffold(
    backgroundColor: const Color(0xFF1A1A1A),
    appBar: widget.showAppBar
        ? AppBar(
            backgroundColor: const Color(0xFF2A2A2A),
            elevation: 0,
            leading: IconButton(
              icon: const Icon(Icons.arrow_back, color: Colors.white),
              onPressed: () => Navigator.pop(context),
            ),
            title: const Text(
              'Nueva Venta',
              style: TextStyle(color: Colors.white),
            ),
            centerTitle: false,
          )
        : null,
    floatingActionButton: isMobile
        ? FloatingActionButton.extended(
            onPressed: () => _mostrarCarritoModal(context),
            backgroundColor: const Color(0xFFE74C3C),
            icon: const Icon(Icons.shopping_cart),
            label: Text('Carrito (${carrito.length})'),
          )
        : null,
    body: cargando
        ? const Center(
            child: CircularProgressIndicator(color: Color(0xFFE74C3C)),
          )
        : error != null
            ? Center(
                child: Text(
                  error!,
                  style: const TextStyle(color: Colors.white),
                ),
              )
            : Column(
                children: [
                  Container(
                    padding: EdgeInsets.all(isMobile ? 12 : 16),
                    decoration: const BoxDecoration(
                      color: Color(0xFF2A2A2A),
                      border: Border(
```

```
                    bottom: BorderSide(
                      color: Color(0xFFE74C3C),
                      width: 2,
                    ),
                  ),
                ),
                child: Column(
                  crossAxisAlignment: CrossAxisAlignment.start,
                  children: [
                    Text(
                      'Nueva Venta - Azteca Fest',
                      style: TextStyle(
                        color: Colors.white,
                        fontSize: isMobile ? 16 : 18,
                        fontWeight: FontWeight.bold,
                      ),
                    ),
                    SizedBox(height: isMobile ? 8 : 12),
                    Container(
                      padding: const EdgeInsets.symmetric(
                          horizontal: 12, vertical: 4),
                      decoration:  BoxDecoration(
                        color: const Color(0xFF1A1A1A),
                        borderRadius: BorderRadius.circular(8),
                        border: Border.all(color: Colors.grey[600]!),
                      ),
                      child: Consumer<UsuarioViewModel>(
                        builder: (context, viewModel, _) {
                          final clientes = viewModel.usuarios;
                          return DropdownButtonHideUnderline(
                            child: DropdownButton<Usuario?>(
                              isExpanded: true,
                              hint: Text(
                                'Seleccionar cliente',
                                style: TextStyle(
                                  color: Colors.grey,
                                  fontSize: isMobile ? 14 : 16,
                                ),
                              ),
                              value: clienteSeleccionado,
                              items: [
                                const DropdownMenuItem<Usuario?>(
                                  value: null,
                                  child: Text(
                                    'Venta al Mostrador',
                                    style: TextStyle(color:
Colors.white),
                                    ),
                                  ),
                                  ...clientes.map(
                                    (c) => DropdownMenuItem<Usuario?>(
                                      value: c,
                                      child: Text(
                                        c.nombre,
```

```
                                        style: const TextStyle(
                                            color: Colors.white),
                                      ),
                                    ),
                                  ),
                                ],
                                onChanged: (valor) {
                                  setState(
                                      () => clienteSeleccionado = valor);
                                },
                                dropdownColor: const Color(0xFF2A2A2A),
                              ),
                            );
                          },
                        ),
                      ),
                    ],
                  ),
                ),
                Expanded(
                  child: isMobile
                      ? _buildMobileLayout()
                      : _buildDesktopLayout(isTablet),
                ),
              ],
            ),
      );
    }

    Widget _buildMobileLayout() {
      return Column(
        children: [
          Padding(
            padding: const EdgeInsets.all(12),
            child: TextField(
              controller: busquedaController,
              onChanged: _filtrarProductos,
              style: const TextStyle(color: Colors.white),
              decoration: InputDecoration(
                hintText: 'Buscar producto...',
                hintStyle: TextStyle(color: Colors.grey[600]),
                prefixIcon: const Icon(Icons.search, color: Color(0xFFE74C3C)),
                filled: true,
                fillColor: const Color(0xFF2A2A2A),
                border: OutlineInputBorder(
                  borderRadius: BorderRadius.circular(8),
                  borderSide: BorderSide(color: Colors.grey[600]!),
                ),
              ),
            ),
          ),
          Expanded(
            child: ListView.builder(
              padding: const EdgeInsets.only(bottom: 80),
```

```
            itemCount: productosFiltrados.length,
            itemBuilder: (context, index) {
              final producto = productosFiltrados[index];
              return _ProductoItem(
                producto: producto,
                onAgregar: () => _agregarAlCarrito(producto),
                cantidadController: cantidadSeleccionada,
                onCantidadChange: (valor) {
                  setState(() => cantidadSeleccionada = valor);
                },
                isMobile: true,
              );
            },
          ),
        ),
      ],
    );
  }

  Widget _buildDesktopLayout(bool isTablet) {
    return Row(
      children: [
        Expanded(
          flex: isTablet ? 1 : 1,
          child: Column(
            children: [
              Padding(
                padding: const EdgeInsets.all(12),
                child: TextField(
                  controller: busquedaController,
                  onChanged: _filtrarProductos,
                  style: const TextStyle(color: Colors.white),
                  decoration: InputDecoration(
                    hintText: 'Buscar producto...',
                    hintStyle: TextStyle(color: Colors.grey[600]),
                    prefixIcon:
                        const Icon(Icons.search, color: Color(0xFFE74C3C)),
                    filled: true,
                    fillColor: const Color(0xFF2A2A2A),
                    border: OutlineInputBorder(
                      borderRadius: BorderRadius.circular(8),
                      borderSide: BorderSide(color: Colors.grey[600]!),
                    ),
                  ),
                ),
              ),
              Expanded(
                child: ListView.builder(
                  itemCount: productosFiltrados.length,
                  itemBuilder: (context, index) {
                    final producto = productosFiltrados[index];
                    return _ProductoItem(
                      producto: producto,
                      onAgregar: () => _agregarAlCarrito(producto),
```

```
                cantidadController: cantidadSeleccionada,
                onCantidadChange: (valor) {
                  setState(() => cantidadSeleccionada = valor);
                },
                isMobile: false,
              );
            },
          ),
        ),
      ],
    ),
  ),
  Expanded(
    flex: isTablet ? 1 : 1,
    child: Column(
      children: [
        Expanded(
          child: Container(
            margin: const EdgeInsets.all(12),
            decoration: BoxDecoration(
              color: const Color(0xFF2A2A2A),
              borderRadius: BorderRadius.circular(8),
              border: Border.all(color: Color(0xFFE74C3C), width: 1.5),
            ),
            child: Column(
              children: [
                Padding(
                  padding: const EdgeInsets.all(12),
                  child: Text(
                    'Carrito (${carrito.length})',
                    style: const TextStyle(
                      color: Colors.white,
                      fontWeight: FontWeight.bold,
                      fontSize: 16,
                    ),
                  ),
                ),
                Expanded(
                  child: carrito.isEmpty
                      ? Center(
                          child: Text(
                            'Carrito vacío',
                            style: TextStyle(color: Colors.grey[600]),
                          ),
                        )
                      : ListView.builder(
                          itemCount: carrito.length,
                          itemBuilder: (context, index) {
                            final item = carrito[index];
                            return _CarritoItem(
                              item: item,
                              onModificar: (cantidad) =>
                                  _modificarCantidad(index, cantidad),
                              onEliminar: () =>
```

```
                          _eliminarDelCarrito(index),
                    );
                  },
                ),
          ),
        ],
      ),
    ),
  ),
  Container(
    padding: const EdgeInsets.all(12),
    decoration: const BoxDecoration(
      color: Color(0xFF2A2A2A),
      border: Border(
        top: BorderSide(color: Color(0xFFE74C3C), width: 1.5),
      ),
    ),
    child: Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
        _ResumenTotal(
          subtotal: subtotal,
          impuesto: impuesto,
          descuento: descuentoTotal,
          total: total,
        ),
        const SizedBox(height: 12),
        Text(
          'Método de Pago:',
          style: TextStyle(
            color: Colors.grey[400],
            fontWeight: FontWeight.bold,
          ),
        ),
        const SizedBox(height: 8),
        DropdownButtonFormField<String>(
          value: metodoPago,
          items: const [
            DropdownMenuItem(
                value: 'efectivo', child: Text('Efectivo')),
            DropdownMenuItem(
                value: 'tarjeta', child: Text('Tarjeta')),
            DropdownMenuItem(
                value: 'transferencia',
                child: Text('Transferencia')),
          ],
          onChanged: (valor) {
            setState(() => metodoPago = valor!);
          },
          decoration: InputDecoration(
            filled: true,
            fillColor: const Color(0xFF1A1A1A),
            border: OutlineInputBorder(
              borderRadius: BorderRadius.circular(6),
```

```
                        ),
                      ),
                      dropdownColor: const Color(0xFF2A2A2A),
                    ),
                    const SizedBox(height: 12),
                    SizedBox(
                      width: double.infinity,
                      child: ElevatedButton(
                        onPressed: carrito.isNotEmpty ? _guardarVenta : null,
                        style: ElevatedButton.styleFrom(
                          backgroundColor: const Color(0xFFE74C3C),
                          disabledBackgroundColor: Colors.grey[700],
                          padding: const EdgeInsets.symmetric(vertical: 12),
                        ),
                        child:  const  Text(
                          'Guardar  Venta',
                          style: TextStyle(
                            color: Colors.white,
                            fontWeight: FontWeight.bold,
                            fontSize: 16,
                          ),
                        ),
                      ),
                    ),
                  ],
                ),
              ),
            ],
          ),
        ),
      ],
    );
  }
}

class _ProductoItem extends StatefulWidget {
  final Producto producto;
  final VoidCallback onAgregar;
  final int cantidadController;
  final Function(int) onCantidadChange;
  final bool isMobile;

  const _ProductoItem({
    required this.producto,
    required this.onAgregar,
    required this.cantidadController,
    required this.onCantidadChange,
    required this.isMobile,
  });

  @override
  State<_ProductoItem> createState() => _ProductoItemState();
}
```

```
class _ProductoItemState extends State<_ProductoItem> {
  @override
  Widget build(BuildContext context) {
    return Container(
      margin: EdgeInsets.symmetric(
        horizontal: widget.isMobile ? 12 : 12,
        vertical: widget.isMobile ? 8 : 6,
      ),
      padding: EdgeInsets.all(widget.isMobile ? 16 : 12),
      decoration: BoxDecoration(
        color: const Color(0xFF2A2A2A),
        borderRadius: BorderRadius.circular(8),
        border: Border.all(color: Colors.grey[700]!),
      ),
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          Row(
            mainAxisAlignment: MainAxisAlignment.spaceBetween,
            children: [
              Expanded(
                child: Column(
                  crossAxisAlignment: CrossAxisAlignment.start,
                  children: [
                    Text(
                      widget.producto.nombre,
                      style: TextStyle(
                        color: Colors.white,
                        fontWeight: FontWeight.bold,
                        fontSize: widget.isMobile ? 15 : 14,
                      ),
                    ),
                    const SizedBox(height: 4),
                    Text(
                      '\$${widget.producto.precioVenta.toStringAsFixed(2)}',
                      style: TextStyle(
                        color: const Color(0xFFE74C3C),
                        fontWeight: FontWeight.bold,
                        fontSize: widget.isMobile ? 16 : 14,
                      ),
                    ),
                  ],
                ),
              ),
              Text(
                'Stock: ${widget.producto.stock}',
                style: TextStyle(
                  color: Colors.grey[500],
                  fontSize: widget.isMobile ? 13 : 12,
                ),
              ),
            ],
          ),
          SizedBox(height: widget.isMobile ? 12 : 8),
```

```
widget.isMobile
    ? Column(
        children: [
          TextField(
            decoration: InputDecoration(
              hintText: 'Cantidad',
              hintStyle: TextStyle(color: Colors.grey[600]),
              filled: true,
              fillColor: const Color(0xFF1A1A1A),
              border: OutlineInputBorder(
                borderRadius: BorderRadius.circular(6),
              ),
              contentPadding: const EdgeInsets.symmetric(
                horizontal: 12,
                vertical: 16,
              ),
            ),
            keyboardType: TextInputType.number,
            style: const TextStyle(color: Colors.white),
            onChanged: (valor) {
              widget.onCantidadChange(int.tryParse(valor) ?? 1);
            },
          ),
          const SizedBox(height: 8),
          SizedBox(
            width: double.infinity,
            child: ElevatedButton.icon(
              onPressed: widget.onAgregar,
              icon: const Icon(Icons.add_shopping_cart, size: 20),
              label: const Text(
                'Agregar al Carrito',
                style: TextStyle(fontSize: 15),
              ),
              style: ElevatedButton.styleFrom(
                backgroundColor: const Color(0xFFE74C3C),
                padding: const EdgeInsets.symmetric(vertical: 14),
              ),
            ),
          ),
        ],
      )
    : Row(
        children: [
          Expanded(
            flex: 2,
            child: TextField(
              decoration: InputDecoration(
                hintText: 'Cantidad',
                hintStyle: TextStyle(color: Colors.grey[600]),
                filled: true,
                fillColor: const Color(0xFF1A1A1A),
                border: OutlineInputBorder(
                  borderRadius: BorderRadius.circular(6),
                ),
```

```
                        ),
                        keyboardType: TextInputType.number,
                        style: const TextStyle(color: Colors.white),
                        onChanged: (valor) {
                          widget.onCantidadChange(int.tryParse(valor) ?? 1);
                        },
                      ),
                    ),
                    const SizedBox(width: 8),
                    Expanded(
                      flex: 3,
                      child: ElevatedButton.icon(
                        onPressed: widget.onAgregar,
                        icon: const Icon(Icons.add_shopping_cart, size: 16),
                        label: const Text('Agregar'),
                        style: ElevatedButton.styleFrom(
                          backgroundColor: const Color(0xFFE74C3C),
                          padding: const EdgeInsets.symmetric(vertical: 12),
                        ),
                      ),
                    ),
                  ],
                ),
              ],
            ),
          );
        }
      }

      class _CarritoItem extends StatelessWidget {
        final ItemCarrito item;
        final Function(int) onModificar;
        final VoidCallback onEliminar;

        const _CarritoItem({
          required this.item,
          required this.onModificar,
          required this.onEliminar,
        });

        @override
        Widget build(BuildContext context) {
          final isMobile = MediaQuery.of(context).size.width < 600;

          return Container(
            padding: EdgeInsets.all(isMobile ? 12 : 8),
            margin: EdgeInsets.symmetric(horizontal: isMobile ? 8 : 0),
            decoration: BoxDecoration(
              border: Border(
                bottom: BorderSide(color: Colors.grey[700]!),
              ),
            ),
            child: Column(
              crossAxisAlignment: CrossAxisAlignment.start,
```

```
        children: [
          Row(
            mainAxisAlignment: MainAxisAlignment.spaceBetween,
            children: [
              Expanded(
                child: Column(
                  crossAxisAlignment: CrossAxisAlignment.start,
                  children: [
                    Text(
                      item.producto.nombre,
                      style: TextStyle(
                        color: Colors.white,
                        fontWeight: FontWeight.bold,
                        fontSize: isMobile ? 14 : 12,
                      ),
                    ),
                    const SizedBox(height: 4),
                    Text(
                      '\$${(item.producto.precioVenta *
item.cantidad).toStringAsFixed(2)}',
                      style: TextStyle(
                        color: const Color(0xFFE74C3C),
                        fontWeight: FontWeight.bold,
                        fontSize: isMobile ? 15 : 13,
                      ),
                    ),
                  ],
                ),
              ),
              IconButton(
                icon: Icon(
                  Icons.delete,
                  color: Colors.red,
                  size: isMobile ? 22 : 18,
                ),
                onPressed: onEliminar,
                padding: EdgeInsets.zero,
                constraints: const BoxConstraints(),
              ),
            ],
          ),
          const SizedBox(height: 8),
          Row(
            children: [
              IconButton(
                icon: Icon(
                  Icons.remove,
                  size: isMobile ? 20 : 16,
                  color: Colors.grey,
                ),
                onPressed: () => onModificar(item.cantidad - 1),
                padding: EdgeInsets.zero,
                constraints: const BoxConstraints(),
              ),
```

```
              const SizedBox(width: 8),
              Text(
                '${item.cantidad}',
                style: TextStyle(
                  color: Colors.white,
                  fontSize: isMobile ? 16 : 12,
                  fontWeight: FontWeight.bold,
                ),
              ),
              const SizedBox(width: 8),
              IconButton(
                icon: Icon(
                  Icons.add,
                  size: isMobile ? 20 : 16,
                  color: Colors.grey,
                ),
                onPressed: () => onModificar(item.cantidad + 1),
                padding: EdgeInsets.zero,
                constraints: const BoxConstraints(),
              ),
            ],
          ),
        ],
      ),
    );
  }
}

class _ResumenTotal extends StatelessWidget {
  final double subtotal;
  final double impuesto;
  final double descuento;
  final double total;

  const _ResumenTotal({
    required this.subtotal,
    required this.impuesto,
    required this.descuento,
    required this.total,
  });

  @override
  Widget build(BuildContext context) {
    return Column(
      crossAxisAlignment: CrossAxisAlignment.start,
      children: [
        _FilaResumen('Subtotal:', subtotal),
        _FilaResumen('Impuesto (16%):', impuesto),
        if (descuento > 0) _FilaResumen('Descuento:', -descuento),
        const Divider(color: Color(0xFFE74C3C), height: 12),
        _FilaResumen(
          'Total:',
          total,
          esBold: true,
```

```
          color: const Color(0xFFE74C3C),
        ),
      ],
    );
  }
}

class _FilaResumen extends StatelessWidget {
  final String label;
  final double valor;
  final bool esBold;
  final Color? color;

  const _FilaResumen(
    this.label,
    this.valor, {
    this.esBold = false,
    this.color,
  });

  @override
  Widget build(BuildContext context) {
    final isMobile = MediaQuery.of(context).size.width < 600;

    return Padding(
      padding: const EdgeInsets.symmetric(vertical: 4),
      child: Row(
        mainAxisAlignment: MainAxisAlignment.spaceBetween,
        children: [
          Text(
            label,
            style: TextStyle(
              color: Colors.grey[400],
              fontWeight: esBold ? FontWeight.bold : FontWeight.normal,
              fontSize: esBold ? (isMobile ? 16 : 14) : (isMobile ? 14 : 12),
            ),
          ),
          Text(
            '\$${valor.toStringAsFixed(2)}',
            style: TextStyle(
              color: color ?? Colors.white,
              fontWeight: esBold ? FontWeight.bold : FontWeight.normal,
              fontSize: esBold ? (isMobile ? 16 : 14) : (isMobile ? 14 : 12),
            ),
          ),
        ],
      ),
    );
  }
}

class ItemCarrito {
  final Producto producto;
  final int cantidad;
```

```
  ItemCarrito({
    required this.producto,
    required this.cantidad,
  });
}

Cliente:
Cliente_dashboard.dart
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import '../../data/models/usuario.dart';
import '../../services/auth_service.dart';
import '../../services/session_timeout_service.dart';
import '../../viewmodels/producto_viewmodel.dart';
import '../../viewmodels/venta_viewmodel.dart';
import '../cliente/catalogo_screen.dart';
import '../cliente/carrito_screen.dart';
import '../cliente/mis_pedidos_tab.dart';
import '../widgets/cart_badge.dart';
import '../widgets/shared/perfil_cliente_screen.dart';
import '../widgets/shared/configuracion_clientescreen.dart';

class ClienteDashboard extends StatefulWidget {
  final Usuario cliente;

  const ClienteDashboard({
    Key? key,
    required this.cliente,
  }) : super(key: key);

  @override
  State<ClienteDashboard> createState() => _ClienteDashboardState();
}

class _ClienteDashboardState extends State<ClienteDashboard>
    with WidgetsBindingObserver {
  int _currentIndex = 0;

  late final List<Widget> _pages;
  final SessionTimeoutService _sessionTimeoutService =
SessionTimeoutService();

  void _navigateToCatalogo() {
    setState(() {
      _currentIndex = 0;
    });
  }

  void _navigateToPedidos() {
    setState(() {
      _currentIndex = 2; // Índice del tab "Mis Pedidos"
    });
  }
```

```
  @override
  void initState() {
    super.initState();

    _pages = [
      const CatalogoScreen(isFromClienteDashboard: true),
      CarritoScreen(
        isFromClienteDashboard: true,
        onNavigateToCatalogo: _navigateToCatalogo,
      ),
      const MisPedidosTab(),
      PerfilClienteScreen(
        cliente: widget.cliente,
        onNavigateToPedidos: _navigateToPedidos,
      ),
    ];

    WidgetsBinding.instance.addObserver(this);
    _sessionTimeoutService.initialize(context);

    WidgetsBinding.instance.addPostFrameCallback((_) {
      if (mounted) {
        context.read<ProductoViewModel>().cargarProductos();
        if (widget.cliente.id != null) {

context.read<VentaViewModel>().cargarPorCliente(widget.cliente.id!);
        }
      }
    });
  }

  @override
  void dispose() {
    WidgetsBinding.instance.removeObserver(this);
    _sessionTimeoutService.cancel();
    super.dispose();
  }

  @override
  Widget build(BuildContext context) {
    return WillPopScope(
      onWillPop: () async => false,
      child: Scaffold(
        backgroundColor: const Color(0xFF1A1A1A),
        appBar: AppBar(
          title: Text('Hola, ${widget.cliente.nombre}',
              style: const TextStyle(color: Colors.white)),
          backgroundColor: const Color(0xFF1A1A1A),
          elevation: 0,
          automaticallyImplyLeading: false,
          actions: [
            CartBadge(
              onTap: () {
```

```
                setState(() {
                  _currentIndex = 1; // Ir al tab del carrito
                });
              },
            ),
            IconButton(
              icon: const Icon(Icons.settings_outlined, color: Colors.white),
              onPressed: () {
                Navigator.push(
                  context,
                  MaterialPageRoute(
                    builder: (context) => ConfiguracionClienteScreen(
                      cliente:
                          widget.cliente, // ✅ Cambié 'usuario' por
'cliente'
                    ),
                  ),
                );
              },
              tooltip: 'Configuración',
            ),

            IconButton(
              icon: const Icon(Icons.logout, color: Colors.white),
              onPressed: _confirmarCerrarSesion,
              tooltip: 'Cerrar sesión',
            ),
          ],
        ),
        body: IndexedStack(
          index: _currentIndex,
          children: _pages,
        ),
        bottomNavigationBar: BottomNavigationBar(
          currentIndex: _currentIndex,
          onTap: (index) {
            setState(() {
              _currentIndex = index;
            });
          },
          type: BottomNavigationBarType.fixed,
          selectedItemColor: Theme.of(context).colorScheme.primary,
          unselectedItemColor: Colors.grey[500],
          selectedFontSize: 12,
          unselectedFontSize: 11,
          elevation: 8,
          backgroundColor: const Color(0xFF2A2A2A),
          items: const [
            BottomNavigationBarItem(
              icon: Icon(Icons.store),
              activeIcon: Icon(Icons.store, size: 28),
              label: 'Catálogo',
            ),
            BottomNavigationBarItem(
```

```
            icon: Icon(Icons.shopping_cart_outlined),
            activeIcon: Icon(Icons.shopping_cart, size: 28),
            label: 'Carrito',
          ),
          BottomNavigationBarItem(
            icon: Icon(Icons.receipt_long_outlined),
            activeIcon: Icon(Icons.receipt_long, size: 28),
            label: 'Mis Pedidos',
          ),
          BottomNavigationBarItem(
            icon: Icon(Icons.person_outline),
            activeIcon: Icon(Icons.person, size: 28),
            label: 'Perfil',
          ),
        ],
      ),
    ),
  );
}

void _confirmarCerrarSesion() {
  showDialog(
    context: context,
    builder: (context) => AlertDialog(
      shape: RoundedRectangleBorder(
        borderRadius: BorderRadius.circular(16),
      ),
      title: Row(
        children: [
          const Icon(Icons.logout, color: Color(0xFFFF073A)),
          const SizedBox(width: 12),
          const Text('Cerrar Sesión'),
        ],
      ),
      content: const Text(
        '¿Estás seguro que deseas cerrar sesión?',
        style: TextStyle(fontSize: 16, color: Colors.white),
      ),
      actions: [
        TextButton(
          onPressed: () => Navigator.pop(context),
          child: Text(
            'Cancelar',
            style: TextStyle(color: Colors.white70),
          ),
        ),
        ElevatedButton(
          onPressed: () async {
            Navigator.pop(context);

            if (mounted) {
              context.read<AuthService>().cerrarSesion();
            }
```

```
              if (mounted) {
                Navigator.pushNamedAndRemoveUntil(
                  context,
                  '/login',
                  (route) => false,
                );
              }
            },
            style: ElevatedButton.styleFrom(
              backgroundColor: const Color(0xFFFF073A),
              foregroundColor: Colors.white,
              shape: RoundedRectangleBorder(
                borderRadius: BorderRadius.circular(8),
              ),
            ),
            child: const Text('Cerrar Sesión'),
          ),
        ],
      ),
    );
  }
}
```

Catalogo_screen.dart

```
import 'dart:io';
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import '../../data/models/producto.dart';
import '../../services/carrito_service.dart';
import '../../viewmodels/producto_viewmodel.dart';
import '../../utils/responsive_helper.dart';
import '../widgets/product_feedback_overlay.dart';
import 'package:font_awesome_flutter/font_awesome_flutter.dart';

class CatalogoScreen extends StatefulWidget {
  final bool isFromClienteDashboard;

  const CatalogoScreen({
    super.key,
    this.isFromClienteDashboard = false,
  });

  @override
  State<CatalogoScreen> createState() => _CatalogoScreenState();
}

class _CatalogoScreenState extends State<CatalogoScreen> {
  String _busqueda = '';
  String _categoriaSeleccionada = 'Todos';
  List<String> _categorias = ['Todos'];

  // Map para almacenar las keys del overlay
  final Map<int, GlobalKey<ProductFeedbackOverlayState>> _overlayKeys = {};
```

```
  GlobalKey<ProductFeedbackOverlayState> _getOverlayKey(int productoId) {
    if (!_overlayKeys.containsKey(productoId)) {
      _overlayKeys[productoId] = GlobalKey<ProductFeedbackOverlayState>();
    }
    return _overlayKeys[productoId]!;
  }

  @override
  void initState() {
    super.initState();
    WidgetsBinding.instance.addPostFrameCallback((_) {
      context.read<ProductoViewModel>().cargarProductos();
      _cargarCategorias();
    });
  }

  Future<void> _cargarCategorias() async {
    try {
      final cats = await
context.read<ProductoViewModel>().obtenerCategorias();
      if (mounted) {
        setState(() {
          _categorias = ['Todos', ...cats];
        });
      }
    } catch (e) {
      print('[v0] Error al cargar categorías: $e');
    }
  }

  IconData _getCategoriaIcon(String categoria) {
    switch (categoria) {
      case 'Bebidas':
        return Icons.local_drink_rounded;
      case 'Comida':
        return Icons.restaurant_rounded;
      case 'Snacks':
        return Icons.fastfood_rounded;
      case 'Postres':
        return Icons.cake_rounded;
      case 'Salchipapas':
        return FontAwesomeIcons.hotdog;
      case 'Sándwiches':
      case 'Sandwiches':
        return Icons.lunch_dining_rounded;
      default:
        return Icons.category_rounded;
    }
  }

  @override
  Widget build(BuildContext context) {
    final resp = ResponsiveHelper(context);
    final productoVM = context.watch<ProductoViewModel>();
```

```
    final carrito = context.watch<CarritoService>();

    final productosFiltrados = _filtrarProductos(productoVM.productos);

    return Scaffold(
      backgroundColor: const Color(0xFF121212),
      appBar: widget.isFromClienteDashboard
          ? null
          : AppBar(
              backgroundColor: const Color(0xFF1A1A1A),
              title: const Text('Catálogo'),
            ),
      body: Column(
        children: [
          // Barra de búsqueda
          Padding(
            padding: EdgeInsets.all(resp.paddingMedium),
            child: TextField(
              onChanged: (value) => setState(() => _busqueda = value),
              style: const TextStyle(color: Colors.white),
              decoration: InputDecoration(
                hintText: 'Buscar productos...',
                hintStyle: TextStyle(color: Colors.grey[600]),
                prefixIcon: const Icon(Icons.search, color: Colors.grey),
                filled: true,
                fillColor: const Color(0xFF2A2A2A),
                border: OutlineInputBorder(
                  borderRadius: BorderRadius.circular(12),
                  borderSide: BorderSide.none,
                ),
                contentPadding: const EdgeInsets.symmetric(vertical: 0,
horizontal: 16),
              ),
            ),
          ),

          // Filtros de categoría
          SizedBox(
            height: 48,
            child: ListView.builder(
              scrollDirection: Axis.horizontal,
              padding: EdgeInsets.symmetric(horizontal: resp.paddingMedium),
              itemCount: _categorias.length,
              itemBuilder: (context, index) {
                final cat = _categorias[index];
                final isSelected = cat == _categoriaSeleccionada;
                final icon = cat == 'Todos' ? Icons.all_inclusive_rounded :
_getCategoriaIcon(cat);

                return Padding(
                  padding: const EdgeInsets.only(right: 8),
                  child: FilterChip(
                    avatar: Icon(
                      icon,
```

```
                      size: 16,
                      color: isSelected ? Colors.white : Colors.grey[400],
                    ),
                    label: Text(cat),
                    selected: isSelected,
                    onSelected: (_) => setState(() => _categoriaSeleccionada
= cat),
                    showCheckmark: false,
                    backgroundColor: const Color(0xFF1A1A1A),
                    selectedColor: const Color(0xFFFF073A),
                    labelStyle: TextStyle(
                      color: isSelected ? Colors.white : Colors.grey[400],
                      fontWeight: isSelected ? FontWeight.bold :
FontWeight.normal,
                      fontSize: 13,
                    ),
                    padding: const EdgeInsets.symmetric(horizontal: 8,
vertical: 8),
                    shape: RoundedRectangleBorder(
                      borderRadius: BorderRadius.circular(12),
                      side: BorderSide(
                        color: isSelected ? const Color(0xFFFF073A) :
Colors.white.withOpacity(0.1),
                        width: 1,
                      ),
                    ),
                  ),
                );
              },
            ),
          ),

          const SizedBox(height: 8),

          // Grid de productos
          Expanded(
            child: productoVM.cargando
                ? const Center(
                    child: CircularProgressIndicator(color:
Color(0xFFFF073A)),
                  )
                : productosFiltrados.isEmpty
                    ? Center(
                        child: Column(
                          mainAxisAlignment: MainAxisAlignment.center,
                          children: [
                            Icon(Icons.inventory_2_outlined,
                                size: 64, color: Colors.grey[700]),
                            const SizedBox(height: 16),
                            Text(
                              'No hay productos disponibles',
                              style: TextStyle(color: Colors.grey[500],
fontSize: 16),
                            ),
```

```
                        ],
                      ),
                    )
                  : RefreshIndicator(
                      onRefresh: () async {
                        await productoVM.cargarProductos();
                        await _cargarCategorias();
                      },
                      color: const Color(0xFFFF073A),
                      child: GridView.builder(
                        padding: EdgeInsets.all(resp.paddingMedium),
                        gridDelegate:
SliverGridDelegateWithFixedCrossAxisCount(
                          crossAxisCount: resp.isMobile ? 2 :
(resp.isTablet ? 3 : 4),
                          childAspectRatio: 0.65,
                          crossAxisSpacing: resp.paddingSmall,
                          mainAxisSpacing: resp.paddingSmall,
                        ),
                        itemCount: productosFiltrados.length,
                        itemBuilder: (context, index) {
                          final p = productosFiltrados[index];
                          return _productoCard(p, carrito, resp);
                        },
                      ),
                    ),
          ),
        ],
      ),
    );
  }

  List<Producto> _filtrarProductos(List<Producto> productos) {
    return productos.where((p) {
      final matchBusqueda = _busqueda.isEmpty ||
          p.nombre.toLowerCase().contains(_busqueda.toLowerCase()) ||
          p.codigo.toLowerCase().contains(_busqueda.toLowerCase());

      final matchCategoria = _categoriaSeleccionada == 'Todos' ||
          p.categoria == _categoriaSeleccionada;

      return matchBusqueda && matchCategoria && p.activo;
    }).toList();
  }

  Widget _productoCard(Producto p, CarritoService carrito, ResponsiveHelper
resp) {
    final overlayKey = _getOverlayKey(p.id ?? 0);
    final cantEnCarrito = carrito.getCantidadProducto(p.id ?? 0);

    return GestureDetector(
      onTap: () => _mostrarDetallesProducto(context, p, resp),
      child: ProductFeedbackOverlay(
        key: overlayKey,
```

```
        child: Container(
          decoration: BoxDecoration(
            color: const Color(0xFF1E1E1E),
            borderRadius: BorderRadius.circular(12),
            border: Border.all(color: Colors.grey[850]!, width: 0.5),
          ),
          child: Column(
            crossAxisAlignment: CrossAxisAlignment.start,
            children: [
              // Imagen del producto - altura fija
              SizedBox(
                height: 100,
                width: double.infinity,
                child: ClipRRect(
                  borderRadius: const BorderRadius.vertical(top:
Radius.circular(12)),
                  child: _buildProductImage(p),
                ),
              ),

              Flexible(
                child: Padding(
                  padding: const EdgeInsets.all(8),
                  child: Column(
                    crossAxisAlignment: CrossAxisAlignment.start,
                    mainAxisSize: MainAxisSize.min,
                    children: [
                      // Nombre del producto
                      Text(
                        p.nombre,
                        style: TextStyle(
                          color: Colors.white,
                          fontWeight: FontWeight.w600,
                          fontSize: resp.fontSmall,
                        ),
                        maxLines: 1,
                        overflow: TextOverflow.ellipsis,
                      ),
                      const SizedBox(height: 2),
                      // Categoría
                      Text(
                        p.categoria,
                        style: TextStyle(
                          color: Colors.grey[500],
                          fontSize: resp.fontSmall - 2,
                        ),
                        maxLines: 1,
                        overflow: TextOverflow.ellipsis,
                      ),
                      const SizedBox(height: 4),
                      // Precio
                      Text(
                        '\$${p.precioVenta.toStringAsFixed(2)}',
                        style: TextStyle(
```

```
                          color: const Color(0xFF4CAF50),
                          fontSize: resp.fontMedium,
                          fontWeight: FontWeight.bold,
                        ),
                      ),

                      // Spacer para empujar el botón hacia abajo
                      const Spacer(),

                      // Botón agregar o selector de cantidad
                      SizedBox(
                        height: 32,
                        child: cantEnCarrito > 0
                            ? _buildQuantitySelector(p, carrito,
cantEnCarrito, overlayKey)
                            : _buildAddButton(p, carrito, overlayKey),
                      ),
                    ],
                  ),
                ),
              ),
            ],
          ),
        ),
      ),
    );
  }

  Widget _buildQuantitySelector(
    Producto p,
    CarritoService carrito,
    int cant,
    GlobalKey<ProductFeedbackOverlayState> overlayKey,
  ) {
    return Container(
      decoration: BoxDecoration(
        color: const Color(0xFF2A2A2A),
        borderRadius: BorderRadius.circular(8),
      ),
      child: Row(
        mainAxisAlignment: MainAxisAlignment.spaceEvenly,
        children: [
          IconButton(
            onPressed: () => carrito.decrementarCantidad(p.id ?? 0),
            icon: const Icon(Icons.remove, color: Color(0xFFFF073A), size:
18),
            constraints: const BoxConstraints(),
            padding: EdgeInsets.zero,
          ),
          Text(
            '$cant',
            style: const TextStyle(
              color: Colors.white,
              fontWeight: FontWeight.bold,
```

```
            fontSize: 14,
          ),
        ),
        IconButton(
          onPressed: () {
            carrito.incrementarCantidad(p.id ?? 0);
            overlayKey.currentState?.showFeedback();
          },
          icon: const Icon(Icons.add, color: Color(0xFF4CAF50), size: 18),
          constraints: const BoxConstraints(),
          padding: EdgeInsets.zero,
        ),
      ],
    ),
  );
}

Widget _buildAddButton(
  Producto p,
  CarritoService carrito,
  GlobalKey<ProductFeedbackOverlayState> overlayKey,
) {
  return SizedBox(
    width: double.infinity,
    height: 32,
    child: ElevatedButton(
      onPressed: () {
        carrito.agregarProducto(p);
        overlayKey.currentState?.showFeedback();
      },
      style: ElevatedButton.styleFrom(
        backgroundColor: const Color(0xFFFF073A),
        shape: RoundedRectangleBorder(
          borderRadius: BorderRadius.circular(8),
        ),
        padding: EdgeInsets.zero,
      ),
      child: const Text(
        'Agregar',
        style: TextStyle(
          color: Colors.white,
          fontSize: 12,
        ),
      ),
    ),
  );
}

Widget _buildProductImage(Producto p) {
  if (p.imagenUrl != null && p.imagenUrl!.isNotEmpty) {
    final url = p.imagenUrl!.trim();

    // Si es una URL de red (http o https)
    if (url.startsWith('http://') || url.startsWith('https://')) {
```

```
      return Image.network(
        url,
        fit: BoxFit.cover,
        width: double.infinity,
        height: double.infinity,
        loadingBuilder: (context, child, loadingProgress) {
          if (loadingProgress == null) return child;
          return Container(
            color: const Color(0xFF1A1A1A),
            child: Center(
              child: CircularProgressIndicator(
                value: loadingProgress.expectedTotalBytes != null
                    ? loadingProgress.cumulativeBytesLoaded /
                        loadingProgress.expectedTotalBytes!
                    : null,
                strokeWidth: 2,
                color: const Color(0xFFFF073A),
              ),
            ),
          );
        },
        errorBuilder: (context, error, stackTrace) {
          print('[v0] Error cargando imagen de red: $error - URL: $url');
          return _buildPlaceholderImage();
        },
      );
    } else {
      // Es un archivo local - verificar que existe
      try {
        final file = File(url);
        if (file.existsSync()) {
          return Image.file(
            file,
            fit: BoxFit.cover,
            width: double.infinity,
            height: double.infinity,
            errorBuilder: (context, error, stackTrace) {
              print('[v0] Error cargando imagen local: $error');
              return _buildPlaceholderImage();
            },
          );
        }
      } catch (e) {
        print('[v0] Error accediendo archivo local: $e');
      }
    }
  }
  return _buildPlaceholderImage();
}

Widget _buildPlaceholderImage() {
  return Container(
    color: const Color(0xFF1A1A1A),
    child: const Center(
```

```
      child: Icon(
        Icons.restaurant_menu,
        size: 35,
        color: Colors.white38,
      ),
    ),
  );
}

// ---------------------------------------------------------------------------------------------
//            MOSTRAR DETALLES DEL PRODUCTO
// ---------------------------------------------------------------------------------------------
void _mostrarDetallesProducto(
    BuildContext context, Producto producto, ResponsiveHelper resp) {
  final carrito = context.read<CarritoService>();

  // Calcular tamaño de imagen responsivo
  final screenWidth = MediaQuery.of(context).size.width;
  final imageSize = resp.isMobile ? screenWidth * 0.5 : screenWidth * 0.25;

  showModalBottomSheet(
    context: context,
    backgroundColor: Colors.transparent,
    isScrollControlled: true,
    builder: (context) => StatefulBuilder(
      builder: (context, setModalState) {
        final cantActual = carrito.getCantidadProducto(producto.id ?? 0);

        return Container(
          constraints: BoxConstraints(
            maxHeight: MediaQuery.of(context).size.height * 0.85,
          ),
          decoration: const BoxDecoration(
            color: Color(0xFF1E1E1E),
            borderRadius: BorderRadius.vertical(top: Radius.circular(24)),
          ),
          child: SingleChildScrollView(
            child: Padding(
              padding: EdgeInsets.all(resp.paddingLarge),
              child: Column(
                mainAxisSize: MainAxisSize.min,
                crossAxisAlignment: CrossAxisAlignment.start,
                children: [
                  // Handle bar
                  Center(
                    child: Container(
                      width: 40,
                      height: 4,
                      margin: const EdgeInsets.only(bottom: 20),
                      decoration: BoxDecoration(
                        color: Colors.grey[700],
                        borderRadius: BorderRadius.circular(2),
                      ),
                    ),
```

```
                    ),

                    // Imagen centrada
                    Center(
                      child: ClipRRect(
                        borderRadius: BorderRadius.circular(16),
                        child: SizedBox(
                          width: imageSize,
                          height: imageSize,
                          child: _buildProductImage(producto),
                        ),
                      ),
                    ),

                    SizedBox(height: resp.paddingLarge),

                    // Nombre
                    Text(
                      producto.nombre,
                      style: TextStyle(
                        color: Colors.white,
                        fontSize: resp.fontLarge + 4,
                        fontWeight: FontWeight.bold,
                      ),
                    ),

                    SizedBox(height: resp.paddingSmall),

                    // Categoría
                    Container(
                      padding: const EdgeInsets.symmetric(horizontal: 12,
vertical: 6),
                      decoration: BoxDecoration(
                        color: const Color(0xFFFF073A).withOpacity(0.2),
                        borderRadius: BorderRadius.circular(20),
                      ),
                      child: Text(
                        producto.categoria,
                        style: TextStyle(
                          color: const Color(0xFFFF073A),
                          fontSize: resp.fontSmall,
                        ),
                      ),
                    ),

                    SizedBox(height: resp.paddingMedium),

                    // Descripción
                    if (producto.descripcion != null &&
producto.descripcion!.isNotEmpty)
                      Text(
                        producto.descripcion!,
                        style: TextStyle(
                          color: Colors.grey[400],
```

```
                        fontSize: resp.fontMedium,
                      ),
                    ),

                SizedBox(height: resp.paddingLarge),

                // Precio y Stock
                Row(
                  mainAxisAlignment: MainAxisAlignment.spaceBetween,
                  children: [
                    Column(
                      crossAxisAlignment: CrossAxisAlignment.start,
                      children: [
                        Text(
                          'Precio',
                          style: TextStyle(color: Colors.grey[500],
fontSize: resp.fontSmall),
                        ),
                        Text(
                          '\$${producto.precioVenta.toStringAsFixed(2)}',
                          style: TextStyle(
                            color: const Color(0xFF4CAF50),
                            fontSize: resp.fontLarge + 8,
                            fontWeight: FontWeight.bold,
                          ),
                        ),
                      ],
                    ),
                    Column(
                      crossAxisAlignment: CrossAxisAlignment.end,
                      children: [
                        Text(
                          'Disponible',
                          style: TextStyle(color: Colors.grey[500],
fontSize: resp.fontSmall),
                        ),
                        Text(
                          '${producto.stock} unidades',
                          style: TextStyle(
                            color: producto.stock > 5 ? Colors.white :
const Color(0xFFFF9800),
                            fontSize: resp.fontMedium,
                            fontWeight: FontWeight.bold,
                          ),
                        ),
                      ],
                    ),
                  ],
                ),

                SizedBox(height: resp.paddingLarge),

                // Botones de acción
                if (cantActual > 0)
```

```
                      Row(
                        children: [
                          // Selector de cantidad
                          Container(
                            decoration:  BoxDecoration(
                              color: const Color(0xFF2A2A2A),
                              borderRadius: BorderRadius.circular(12),
                            ),
                            child: Row(
                              children: [
                                IconButton(
                                  onPressed: () {
                                    carrito.decrementarCantidad(producto.id
?? 0);

                                    setModalState(() {});
                                  },
                                  icon: const Icon(Icons.remove, color:
Color(0xFFFF073A)),
                                ),
                                Padding(
                                  padding: const
EdgeInsets.symmetric(horizontal:  16),
                                  child: Text(
                                    '$cantActual',
                                    style: const TextStyle(
                                      color: Colors.white,
                                      fontSize: 18,
                                      fontWeight: FontWeight.bold,
                                    ),
                                  ),
                                ),
                                IconButton(
                                  onPressed: () {
                                    carrito.incrementarCantidad(producto.id
?? 0);

                                    setModalState(() {});
                                  },
                                  icon: const Icon(Icons.add, color:
Color(0xFF4CAF50)),

                                ),
                              ],
                            ),
                          ),
                          const SizedBox(width: 16),
                          // Subtotal
                          Expanded(
                            child: Container(
                              padding: const EdgeInsets.symmetric(vertical:
16),

                              decoration:  BoxDecoration(
                                color: const Color(0xFF4CAF50),
                                borderRadius: BorderRadius.circular(12),
                              ),
                              child: Center(
```

```
                                child: Text(
                                  'Subtotal: \$${(producto.precioVenta *
cantActual).toStringAsFixed(2)}',
                                  style: const TextStyle(
                                    color: Colors.white,
                                    fontWeight: FontWeight.bold,
                                    fontSize: 16,
                                  ),
                                ),
                              ),
                            ),
                          ),
                        ],
                      )
                    else
                      SizedBox(
                        width: double.infinity,
                        child: ElevatedButton.icon(
                          onPressed: () {
                            carrito.agregarProducto(producto);
                            setModalState(() {});
                          },
                          icon: const Icon(Icons.add_shopping_cart),
                          label: const Text('Agregar al carrito'),
                          style: ElevatedButton.styleFrom(
                            backgroundColor: const Color(0xFFFF073A),
                            foregroundColor: Colors.white,
                            padding: const EdgeInsets.symmetric(vertical:
16),
                            shape: RoundedRectangleBorder(
                              borderRadius: BorderRadius.circular(12),
                            ),
                          ),
                        ),
                      ),

                    SizedBox(height: resp.paddingMedium),
                  ],
                ),
              ),
            ),
          );
        },
      ),
    );
  }
}
```

Carrito_screen.dart

```dart
import 'dart:io';
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import '../../services/carrito_service.dart';
import '../../services/auth_service.dart';
import '../../viewmodels/venta_viewmodel.dart';
import '../../core/app_routes.dart';
import '../../core/constants.dart';
import '../../data/models/venta.dart';
import '../../data/models/detalle_venta.dart';
import '../cliente/crear_pedido_cliente_screen.dart';

class CarritoScreen extends StatelessWidget {
  final bool isFromClienteDashboard;
  final VoidCallback? onNavigateToCatalogo;

  const CarritoScreen({
    Key? key,
    this.isFromClienteDashboard = false,
    this.onNavigateToCatalogo,
  }) : super(key: key);

  @override
  Widget build(BuildContext context) {
    final carrito = context.watch<CarritoService>();
    final authService = context.watch<AuthService>();
    final width = MediaQuery.of(context).size.width;
    final sizes = AppConstants.getResponsiveSizes(width);
    final isResponsive = !AppConstants.isMobile(width);

    return Scaffold(
      backgroundColor: AppConstants.fondoOscuro,
      appBar: AppBar(
        backgroundColor: AppConstants.fondoMedio,
        elevation: 0,
        automaticallyImplyLeading: false,
        title: Text(
          'Mi Carrito',
          style: TextStyle(
            fontSize: sizes['titleSize'],
            fontWeight: FontWeight.bold,
            color: AppConstants.textoBlanco,
          ),
        ),
        actions: [
          if (!carrito.estaVacio)
            TextButton.icon(
              icon: Icon(
                Icons.delete_outline,
                color: const Color(0xFFFF073A),
                size: sizes['iconSize'],
              ),
              label: Text(
```

```
              'Vaciar',
              style: TextStyle(
                fontSize: sizes['fontSize'],
                fontWeight: FontWeight.bold,
                color: const Color(0xFFFF073A),
              ),
            ),
            onPressed: () => _mostrarDialogoVaciar(context, carrito),
          ),
      ],
    ),
    body: carrito.estaVacio
        ? _buildCarritoVacio(context)
        : Column(
            children: [
              Expanded(
                child: ListView.builder(
                  padding: EdgeInsets.all(sizes['padding']),
                  itemCount: carrito.items.length,
                  itemBuilder: (context, index) {
                    final item = carrito.items[index];
                    return _buildItemCarrito(
                      context,
                      item,
                      carrito,
                      isResponsive,
                    );
                  },
                ),
              ),
              _buildResumen(context, carrito, authService),
            ],
          ),
  );
}

Widget _buildCarritoVacio(BuildContext context) {
  return Center(
    child: Column(
      mainAxisAlignment: MainAxisAlignment.center,
      children: [
        Icon(
          Icons.shopping_cart_outlined,
          size: 100,
          color: Colors.grey[600],
        ),
        const SizedBox(height: 24),
        Text(
          'Tu carrito está vacío',
          style: TextStyle(
            fontSize: 20,
            color: Colors.grey[400],
            fontWeight: FontWeight.w500,
          ),
```

```
          ),
          const SizedBox(height: 16),
          ElevatedButton(
            onPressed: () {
              if (isFromClienteDashboard && onNavigateToCatalogo != null) {
                onNavigateToCatalogo!();
              } else if (isFromClienteDashboard) {
                ScaffoldMessenger.of(context).showSnackBar(
                  const SnackBar(
                    content: Text('Usa la barra inferior para ir al
Catálogo'),
                    backgroundColor: Color(0xFF2A2A2A),
                    duration: Duration(seconds: 2),
                  ),
                );
              } else {
                Navigator.pushNamed(context, AppRoutes.home);
              }
            },
            style: ElevatedButton.styleFrom(
              backgroundColor: const Color(0xFFFF073A),
            ),
            child: Text(isFromClienteDashboard
                ? 'Ir al Catálogo ↓'
                : 'Explorar productos'),
          ),
        ],
      ),
    );
  }

  Widget _buildItemCarrito(
    BuildContext context,
    ItemCarrito item,
    CarritoService carrito,
    bool isResponsive,
  ) {
    return Card(
      margin: const EdgeInsets.only(bottom: 12),
      color: const Color(0xFF2A2A2A),
      elevation: 4,
      child: Padding(
        padding: const EdgeInsets.all(12),
        child: isResponsive
            ? Row(
                children: [
                  _buildImagen(item),
                  const SizedBox(width: 16),
                  Expanded(child: _buildInfo(item)),
                  _buildControles(context, item, carrito),
                ],
              )
            : Column(
                children: [
```

```
                Row(
                  children: [
                    _buildImagen(item),
                    const SizedBox(width: 12),
                    Expanded(child: _buildInfo(item)),
                    IconButton(
                      icon: const Icon(
                        Icons.delete_outline,
                        color: Color(0xFFFF073A),
                      ),
                      onPressed: () {
                        if (item.producto.id != null) {
                          carrito.eliminarProducto(item.producto.id!);
                        }
                      },
                    ),
                  ],
                ),
                const SizedBox(height: 12),
                _buildControles(context, item, carrito),
              ],
            ),
    ),
  );
}

Widget _buildImagen(ItemCarrito item) {
  return ClipRRect(
    borderRadius: BorderRadius.circular(8),
    child: _buildItemImage(item.producto),
  );
}

Widget _buildItemImage(producto) {
  if (producto.imagenUrl != null && producto.imagenUrl!.isNotEmpty) {
    final url = producto.imagenUrl!.trim();

    if (url.startsWith('http://') || url.startsWith('https://')) {
      return Image.network(
        url,
        width: 80,
        height: 80,
        fit: BoxFit.cover,
        loadingBuilder: (context, child, loadingProgress) {
          if (loadingProgress == null) return child;
          return Container(
            width: 80,
            height: 80,
            color: const Color(0xFF1A1A1A),
            child: const Center(
              child: CircularProgressIndicator(
                strokeWidth: 2,
                color: Color(0xFFFF073A),
              ),
```

```
            ),
          );
        },
        errorBuilder: (context, error, stackTrace) {
          return _buildImagenPlaceholder();
        },
      );
    } else {
      try {
        final file = File(url);
        if (file.existsSync()) {
          return Image.file(
            file,
            width: 80,
            height: 80,
            fit: BoxFit.cover,
            errorBuilder: (context, error, stackTrace) {
              return _buildImagenPlaceholder();
            },
          );
        }
      } catch (e) {
        // Ignorar error
      }
    }
  }
  return _buildImagenPlaceholder();
}

Widget _buildImagenPlaceholder() {
  return Container(
    width: 80,
    height: 80,
    color: const Color(0xFF1A1A1A),
    child: Icon(
      Icons.image_not_supported,
      color: Colors.grey[700],
      size: 40,
    ),
  );
}

Widget _buildInfo(ItemCarrito item) {
  final stockBajo = item.cantidad > item.producto.stock;

  return Column(
    crossAxisAlignment: CrossAxisAlignment.start,
    children: [
      Text(
        item.producto.nombre,
        style: const TextStyle(
          fontSize: 16,
          fontWeight: FontWeight.bold,
          color: Colors.white,
```

```
        ),
        maxLines: 2,
        overflow: TextOverflow.ellipsis,
      ),
      const SizedBox(height: 4),
      Text(
        'Código: ${item.producto.codigo}',
        style: TextStyle(
          fontSize: 12,
          color: Colors.grey[500],
        ),
      ),
      const SizedBox(height: 4),
      Text(
        'Stock disponible: ${item.producto.stock}',
        style: TextStyle(
          fontSize: 12,
          color: stockBajo ? const Color(0xFFFF073A) : Colors.grey[500],
          fontWeight: stockBajo ? FontWeight.bold : FontWeight.normal,
        ),
      ),
      const SizedBox(height: 8),
      Text(
        '\$${item.producto.precioVenta.toStringAsFixed(2)} c/u',
        style: const TextStyle(
          fontSize: 14,
          color: Color(0xFFFFD700),
          fontWeight: FontWeight.w600,
        ),
      ),
      if (stockBajo)
        const Padding(
          padding: EdgeInsets.only(top: 4),
          child: Text(
            '¡Stock insuficiente!',
            style: TextStyle(
              fontSize: 12,
              color: Color(0xFFFF073A),
              fontWeight: FontWeight.bold,
            ),
          ),
        ),
    ],
  );
}

Widget _buildControles(
  BuildContext context,
  ItemCarrito item,
  CarritoService carrito,
) {
  return Container(
    decoration: BoxDecoration(
      border: Border.all(color: const Color(0xFFFF073A)),
```

```
  borderRadius: BorderRadius.circular(8),
  color: const Color(0xFF1A1A1A),
),
child: Row(
  mainAxisSize: MainAxisSize.min,
  children: [
    IconButton(
      icon: const Icon(Icons.remove_circle, color: Color(0xFFFF073A)),
      onPressed: item.producto.id != null
          ? () {
              carrito.actualizarCantidad(
                item.producto.id!,
                item.cantidad - 1,
              );
            }
          : null,
    ),
    Container(
      padding: const EdgeInsets.symmetric(horizontal: 16),
      child: Text(
        '${item.cantidad}',
        style: const TextStyle(
          fontSize: 18,
          fontWeight: FontWeight.bold,
          color: Colors.white,
        ),
      ),
    ),
    IconButton(
      icon: const Icon(Icons.add_circle, color: Color(0xFFFF073A)),
      onPressed: item.producto.id != null
          ? () {
              if (item.cantidad < item.producto.stock) {
                carrito.actualizarCantidad(
                  item.producto.id!,
                  item.cantidad + 1,
                );
              } else {
                ScaffoldMessenger.of(context).showSnackBar(
                  const SnackBar(
                    content: Text('Stock insuficiente'),
                    duration: Duration(seconds: 2),
                    backgroundColor: Color(0xFF2A2A2A),
                  ),
                );
              }
            }
          : null,
    ),
    const SizedBox(width: 16),
    Text(
      '\$${item.subtotal.toStringAsFixed(2)}',
      style: const TextStyle(
        fontSize: 18,
```

```
            fontWeight: FontWeight.bold,
            color: Color(0xFFFFD700),
          ),
        ),
        const SizedBox(width: 8),
      ],
    ),
  );
}

Widget _buildResumen(
  BuildContext context,
  CarritoService carrito,
  AuthService authService,
) {
  return Container(
    padding: const EdgeInsets.all(24),
    decoration: BoxDecoration(
      color: const Color(0xFF2A2A2A),
      boxShadow: [
        BoxShadow(
          color: Colors.black.withOpacity(0.3),
          blurRadius: 10,
          offset: const Offset(0, -5),
        ),
      ],
    ),
    child: Column(
      children: [
        // Subtotal y Total
        Row(
          mainAxisAlignment: MainAxisAlignment.spaceBetween,
          children: [
            const Text(
              'Subtotal:',
              style: TextStyle(fontSize: 16, color: Colors.white),
            ),
            Text(
              '\$${carrito.total.toStringAsFixed(2)}',
              style: const TextStyle(fontSize: 16, color: Colors.white),
            ),
          ],
        ),
        const SizedBox(height: 8),
        Row(
          mainAxisAlignment: MainAxisAlignment.spaceBetween,
          children: [
            const Text(
              'Total:',
              style: TextStyle(
                fontSize: 20,
                fontWeight: FontWeight.bold,
                color: Colors.white,
              ),
```

```
            ),
            Text(
              '\$${carrito.total.toStringAsFixed(2)}',
              style: const TextStyle(
                fontSize: 24,
                fontWeight: FontWeight.bold,
                color: Color(0xFFFFD700),
              ),
            ),
          ],
        ),
        const SizedBox(height: 16),

        // ✅ UN SOLO BOTÓN PARA CLIENTES: "Hacer Pedido"
        if (authService.estaAutenticado &&
            authService.usuarioActual!.rol == 'cliente')
          SizedBox(
            width: double.infinity,
            child: ElevatedButton.icon(
              onPressed: () => _crearPedido(context, carrito),
              icon: const Icon(Icons.assignment_outlined),
              label: const Text('Hacer Pedido'),
              style: ElevatedButton.styleFrom(
                backgroundColor: const Color(0xFF4A9EFF),
                foregroundColor: Colors.white,
                padding: const EdgeInsets.symmetric(vertical: 16),
                shape: RoundedRectangleBorder(
                  borderRadius: BorderRadius.circular(12),
                ),
              ),
            ),
          )
        // ✅ BOTÓN PARA VENDEDOR/ADMIN (Registrar Venta)
        else if (authService.estaAutenticado)
          SizedBox(
            width: double.infinity,
            child: ElevatedButton(
              onPressed: () => _procesarVenta(context, carrito,
authService),
              style: ElevatedButton.styleFrom(
                backgroundColor: const Color(0xFFFF073A),
                padding: const EdgeInsets.symmetric(vertical: 16),
                shape: RoundedRectangleBorder(
                  borderRadius: BorderRadius.circular(12),
                ),
              ),
              child: const Text(
                'Registrar Venta',
                style: TextStyle(
                  fontSize: 18,
                  fontWeight: FontWeight.bold,
                  color: Colors.white,
                ),
              ),
```

```
            ),
          )
        // ✅ BOTÓN PARA NO AUTENTICADOS
        else
          SizedBox(
            width: double.infinity,
            child: ElevatedButton(
              onPressed: () => _mostrarDialogoLogin(context),
              style: ElevatedButton.styleFrom(
                backgroundColor: const Color(0xFFFF073A),
                padding: const EdgeInsets.symmetric(vertical: 16),
                shape: RoundedRectangleBorder(
                  borderRadius: BorderRadius.circular(12),
                ),
              ),
              child: const Text(
                'Iniciar sesión para comprar',
                style: TextStyle(
                  fontSize: 18,
                  fontWeight: FontWeight.bold,
                  color: Colors.white,
                ),
              ),
            ),
          ),
      ],
    ),
  );
}

void _mostrarDialogoVaciar(BuildContext context, CarritoService carrito) {
  showDialog(
    context: context,
    builder: (context) => AlertDialog(
      backgroundColor: const Color(0xFF2A2A2A),
      title: const Text(
        'Vaciar carrito',
        style: TextStyle(color: Colors.white),
      ),
      content: Text(
        '¿Estás seguro de que deseas vaciar el carrito?',
        style: TextStyle(color: Colors.grey[400]),
      ),
      actions: [
        TextButton(
          onPressed: () => Navigator.pop(context),
          child: Text(
            'Cancelar',
            style: TextStyle(color: Colors.grey[400]),
          ),
        ),
        ElevatedButton(
          onPressed: () {
            carrito.vaciar();
```

```
            Navigator.pop(context);
          },
          style: ElevatedButton.styleFrom(
            backgroundColor: const Color(0xFFFF073A),
          ),
          child: const Text('Vaciar'),
        ),
      ],
    ),
  );
}

void _mostrarDialogoLogin(BuildContext context) {
  showDialog(
    context: context,
    builder: (context) => AlertDialog(
      backgroundColor: const Color(0xFF2A2A2A),
      title: const Text(
        'Iniciar sesión',
        style: TextStyle(color: Colors.white),
      ),
      content: Text(
        'Necesitas iniciar sesión para realizar una compra.',
        style: TextStyle(color: Colors.grey[400]),
      ),
      actions: [
        TextButton(
          onPressed: () => Navigator.pop(context),
          child: Text(
            'Cancelar',
            style: TextStyle(color: Colors.grey[400]),
          ),
        ),
        ElevatedButton(
          onPressed: () {
            Navigator.pop(context);
            Navigator.pushNamed(context, AppRoutes.login);
          },
          style: ElevatedButton.styleFrom(
            backgroundColor: const Color(0xFFFF073A),
          ),
          child: const Text('Iniciar sesión'),
        ),
      ],
    ),
  );
}

Future<void> _procesarVenta(
  BuildContext context,
  CarritoService carrito,
  AuthService authService,
) async {
  if (!carrito.validarStock()) {
```

```
      final productosInsuficientes = carrito.getProductosStockInsuficiente();
      showDialog(
        context: context,
        builder: (context) => AlertDialog(
          backgroundColor: const Color(0xFF2A2A2A),
          title: const Text(
            'Stock Insuficiente',
            style: TextStyle(color: Colors.white),
          ),
          content: Text(
            'Los siguientes productos no tienen stock
suficiente:\n\n${productosInsuficientes.map((item) => '•
${item.producto.nombre}').join('\n')}',
            style: TextStyle(color: Colors.grey[400]),
          ),
          actions: [
            ElevatedButton(
              onPressed: () => Navigator.pop(context),
              style: ElevatedButton.styleFrom(
                backgroundColor: const Color(0xFFFF073A),
              ),
              child: const Text('Entendido'),
            ),
          ],
        ),
      );
      return;
    }

    showDialog(
      context: context,
      barrierDismissible: false,
      builder: (context) => const Center(
        child: CircularProgressIndicator(color: Color(0xFFFF073A)),
      ),
    );

    try {
      final ventaViewModel = context.read<VentaViewModel>();

      final venta = Venta(
        numeroVenta: 'V-${DateTime.now().millisecondsSinceEpoch}',
        fecha: DateTime.now(),
        clienteId: authService.usuarioActual!.id!,
        vendedorId: authService.usuarioActual!.id!,
        subtotal: carrito.total,
        descuento: 0,
        impuesto: 0,
        total: carrito.total,
        metodoPago: 'efectivo',
        estado: 'completada',
        nombreCliente: authService.usuarioActual!.nombre,
      );
```

```
      final detalles = carrito.items.map((item) {
        return DetalleVenta(
          ventaId: 0,
          productoId: item.producto.id!,
          cantidad: item.cantidad,
          precioUnitario: item.producto.precioVenta,
          subtotal: item.subtotal,
          descuento: 0,
          total: item.subtotal,
        );
      }).toList();

      final exito = await ventaViewModel.crearVenta(venta, detalles);

      if (context.mounted) Navigator.pop(context);

      if (exito) {
        carrito.vaciar();

        if (context.mounted) {
          showDialog(
            context: context,
            barrierDismissible: false,
            builder: (context) => AlertDialog(
              backgroundColor: const Color(0xFF2A2A2A),
              title: Row(
                children: const [
                  Icon(Icons.check_circle, color: Color(0xFF4CAF50), size:
32),
                  SizedBox(width: 12),
                  Text('Compra Exitosa', style: TextStyle(color:
Colors.white)),
                ],
              ),
              content: Text(
                'Tu compra por \$${venta.total.toStringAsFixed(2)} ha sido
registrada exitosamente.\n\n'
                'Número de venta: ${venta.numeroVenta}\n'
                'Productos: ${detalles.length}',
                style: TextStyle(color: Colors.grey[400]),
              ),
              actions: [
                ElevatedButton(
                  onPressed: () {
                    Navigator.pop(context);
                    Navigator.pop(context);

                    final usuario = authService.usuarioActual;
                    if (usuario != null) {
                      Navigator.pushNamedAndRemoveUntil(
                        context,
                        AppRoutes.clienteDashboard,
                        (route) => false,
                        arguments: usuario,
```

```
                    );
                  } else {
                    Navigator.pushNamedAndRemoveUntil(
                      context,
                      AppRoutes.home,
                      (route) => false,
                    );
                  }
                },
                style: ElevatedButton.styleFrom(
                  backgroundColor: const Color(0xFFFF073A),
                ),
                child: const Text('Aceptar'),
              ),
            ],
          ),
        );
      }
    } else {
      if (context.mounted) {
        ScaffoldMessenger.of(context).showSnackBar(
          SnackBar(
            content:
                Text(ventaViewModel.error ?? 'Error al procesar la venta'),
            backgroundColor: const Color(0xFFFF073A),
          ),
        );
      }
    }
  } catch (e) {
    if (context.mounted) {
      Navigator.pop(context);
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(
          content: Text('Error: ${e.toString()}'),
          backgroundColor: const Color(0xFFFF073A),
        ),
      );
    }
  }
}

// ✅ MÉTODO SIMPLIFICADO: Solo navega a CrearPedidoClienteScreen
Future<void> _crearPedido(
    BuildContext context, CarritoService carrito) async {
  final authService = context.read<AuthService>();
  final usuario = authService.usuarioActual;

  if (usuario == null || usuario.id == null) {
    ScaffoldMessenger.of(context).showSnackBar(
      const SnackBar(
        content: Text('Error: Usuario no autenticado'),
        backgroundColor: Color(0xFFFF073A),
      ),
```

```
      );
      return;
    }

    // Preparar items para la pantalla de crear pedido
    final itemsCarrito = carrito.items
        .map((item) => {
              'producto': item.producto,
              'cantidad': item.cantidad,
            })
        .toList();

    // ✅ Navegar a CrearPedidoClienteScreen con los items del carrito
    final resultado = await Navigator.push<bool>(
      context,
      MaterialPageRoute(
        builder: (context) => CrearPedidoClienteScreen(
          itemsCarrito: itemsCarrito,
          total: carrito.total,
        ),
      ),
    );

    // Si el pedido se creó exitosamente, vaciar el carrito
    if (resultado == true && context.mounted) {
      carrito.vaciar();
      ScaffoldMessenger.of(context).showSnackBar(
        const SnackBar(
          content: Row(
            children: [
              Icon(Icons.check_circle, color: Colors.white),
              SizedBox(width: 12),
              Text('Pedido creado exitosamente'),
            ],
          ),
          backgroundColor: Color(0xFF66BB6A),
        ),
      );
    }
  }
}
```

Mis_pedidos_tab.dart

```
// lib/views/cliente/mis_pedidos_tab.dart

import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import '../../viewmodels/pedido_viewmodel.dart';
import '../../services/auth_service.dart';
import '../../data/models/pedido.dart';

class MisPedidosTab extends StatefulWidget {
  final VoidCallback? onIrCatalogo;
  const MisPedidosTab({Key? key, this.onIrCatalogo}) : super(key: key);
```

```
  @override
  State<MisPedidosTab> createState() => _MisPedidosTabState();
}

class _MisPedidosTabState extends State<MisPedidosTab> {
  @override
  void initState() {
    super.initState();
    WidgetsBinding.instance.addPostFrameCallback((_) {
      _cargarPedidos();
    });
  }

  Future<void> _cargarPedidos() async {
    final authService = context.read<AuthService>();
    final clienteId = authService.usuarioActual?.id;

    if (clienteId != null) {
      context.read<PedidoViewModel>().cargarPorCliente(clienteId);
    }
  }

  @override
  Widget build(BuildContext context) {
    return Consumer<PedidoViewModel>(
      builder: (context, viewModel, child) {
        if (viewModel.cargando) {
          return const Center(
            child: CircularProgressIndicator(color: Color(0xFF4A9EFF)),
          );
        }

        if (viewModel.pedidos.isEmpty) {
          return Center(
            child: Column(
              mainAxisAlignment: MainAxisAlignment.center,
              children: [
                Icon(
                  Icons.inbox_outlined,
                  size: 80,
                  color: Colors.grey[600],
                ),
                const SizedBox(height: 16),
                Text(
                  'No tienes pedidos aún',
                  style: TextStyle(
                    color: Colors.grey[400],
                    fontSize: 18,
                  ),
                ),
                const SizedBox(
                  height: 24,
                ),
```

```
              //Agregamos el boton
              ElevatedButton.icon(
                onPressed: widget.onIrCatalogo, // Usar el callback
                icon: const Icon(Icons.store),
                label: const Text('Ir al Catálogo'),
                style: ElevatedButton.styleFrom(
                  backgroundColor: const Color(0xFF4A9EFF),
                  padding: const EdgeInsets.symmetric(
                    horizontal: 24,
                    vertical: 12,
                  ),
                  shape: RoundedRectangleBorder(
                    borderRadius: BorderRadius.circular(12),
                  ),
                ),
              ),
            ],
          ),
        );
      }

      return RefreshIndicator(
        onRefresh: _cargarPedidos,
        color: const Color(0xFF4A9EFF),
        child: ListView.builder(
          padding: const EdgeInsets.all(16),
          itemCount: viewModel.pedidos.length,
          itemBuilder: (context, index) {
            final pedido = viewModel.pedidos[index];
            return _buildPedidoCard(pedido, viewModel);
          },
        ),
      );
    },
  );
}

Widget _buildPedidoCard(Pedido pedido, PedidoViewModel viewModel) {
  return Container(
    margin: const EdgeInsets.only(bottom: 16),
    decoration: BoxDecoration(
      color: const Color(0xFF1A1A1A),
      borderRadius: BorderRadius.circular(16),
      border: Border.all(color: Colors.white.withOpacity(0.1)),
    ),
    child: Column(
      children: [
        ListTile(
          contentPadding: const EdgeInsets.all(16),
          leading: _buildEstadoIcon(pedido.estado.name),
          title: Text(
            pedido.numeroPedido,
            style: const TextStyle(
              color: Colors.white,
```

```
                fontWeight: FontWeight.bold,
              ),
            ),
            subtitle: Text(
              _formatearFecha(pedido.fechaPedido),
              style: TextStyle(color: Colors.grey[500]),
            ),
            trailing: _buildEstadoChip(pedido.estado.name),
          ),
          Divider(color: Colors.white.withOpacity(0.1), height: 1),
          Padding(
            padding: const EdgeInsets.all(16),
            child: Row(
              mainAxisAlignment: MainAxisAlignment.spaceBetween,
              children: [
                Column(
                  crossAxisAlignment: CrossAxisAlignment.start,
                  children: [
                    Text(
                      'Total',
                      style: TextStyle(
                        color: Colors.grey[500],
                        fontSize: 12,
                      ),
                    ),
                    Text(
                      '\$${pedido.total.toStringAsFixed(2)}',
                      style: const TextStyle(
                        color: Color(0xFF66BB6A),
                        fontSize: 20,
                        fontWeight: FontWeight.bold,
                      ),
                    ),
                  ],
                ),
                Row(
                  children: [
                    IconButton(
                      onPressed: () => _verDetalle(pedido, viewModel),
                      icon: const Icon(Icons.visibility),
                      color: const Color(0xFF4A9EFF),
                      tooltip: 'Ver detalles',
                    ),
                    if (pedido.estado == 'pendiente')
                      IconButton(
                        onPressed: () => _confirmarCancelar(pedido,
viewModel),
                        icon: const Icon(Icons.cancel),
                        color: const Color(0xFFFF073A),
                        tooltip: 'Cancelar pedido',
                      ),
                  ],
                ),
              ],
```

```
          ),
        ),
      ],
    ),
  );
}

Widget _buildEstadoIcon(String estado) {
  IconData icon;
  Color color;

  switch (estado.toLowerCase()) {
    case 'pendiente':
      icon = Icons.schedule;
      color = const Color(0xFFFFA726);
      break;
    case 'procesando':
      icon = Icons.autorenew;
      color = const Color(0xFF4A9EFF);
      break;
    case 'enviado':
      icon = Icons.local_shipping;
      color = const Color(0xFF9C27B0);
      break;
    case 'completado':
    case 'entregado':
      icon = Icons.check_circle;
      color = const Color(0xFF66BB6A);
      break;
    case 'cancelado':
      icon = Icons.cancel;
      color = const Color(0xFFFF073A);
      break;
    default:
      icon = Icons.help;
      color = Colors.grey;
  }

  return Container(
    padding: const EdgeInsets.all(12),
    decoration: BoxDecoration(
      color: color.withOpacity(0.2),
      borderRadius: BorderRadius.circular(12),
    ),
    child: Icon(icon, color: color, size: 28),
  );
}

Widget _buildEstadoChip(String estado) {
  Color color;
  switch (estado.toLowerCase()) {
    case 'pendiente':
      color = const Color(0xFFFFA726);
      break;
```

```
    case 'procesando':
      color = const Color(0xFF4A9EFF);
      break;
    case 'enviado':
      color = const Color(0xFF9C27B0);
      break;
    case 'completado':
    case 'entregado':
      color = const Color(0xFF66BB6A);
      break;
    case 'cancelado':
      color = const Color(0xFFFF073A);
      break;
    default:
      color = Colors.grey;
  }

  return Container(
    padding: const EdgeInsets.symmetric(horizontal: 12, vertical: 6),
    decoration: BoxDecoration(
      color: color.withOpacity(0.2),
      borderRadius: BorderRadius.circular(20),
      border: Border.all(color: color),
    ),
    child: Text(
      estado.toUpperCase(),
      style: TextStyle(
        color: color,
        fontSize: 11,
        fontWeight: FontWeight.bold,
      ),
    ),
  );
}

void _verDetalle(Pedido pedido, PedidoViewModel viewModel) async {
  final pedidoDetalle = await viewModel.obtenerPedidoDetalle(pedido.id!);
  if (!mounted || pedidoDetalle == null) return;

  showDialog(
    context: context,
    builder: (context) => Dialog(
      backgroundColor: const Color(0xFF1A1A1A),
      shape: RoundedRectangleBorder(
        borderRadius: BorderRadius.circular(20),
      ),
      child: SingleChildScrollView(
        padding: const EdgeInsets.all(20),
        child: Column(
          mainAxisSize: MainAxisSize.min,
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            Row(
              mainAxisAlignment: MainAxisAlignment.spaceBetween,
```

```
  children: [
    const Text(
      'Detalle del Pedido',
      style: TextStyle(
        color: Colors.white,
        fontSize: 20,
        fontWeight: FontWeight.bold,
      ),
    ),
    IconButton(
      onPressed: () => Navigator.pop(context),
      icon: const Icon(Icons.close, color: Colors.white),
    ),
  ],
),
const SizedBox(height: 20),
Text(
  'Número: ${pedido.numeroPedido}',
  style: const TextStyle(color: Colors.white),
),
const SizedBox(height: 8),
Text(
  'Fecha: ${_formatearFecha(pedido.fechaPedido)}',
  style: TextStyle(color: Colors.grey[400]),
),
const SizedBox(height: 8),
if (pedido.direccionEntrega != null) ...[
  Text(
    'Dirección: ${pedido.direccionEntrega}',
    style: TextStyle(color: Colors.grey[400]),
  ),
  const SizedBox(height: 8),
],
Text(
  'Estado: ${pedido.estado.name.toUpperCase()}',
  style: TextStyle(
    color: _getEstadoColor(pedido.estado.name),
    fontWeight: FontWeight.bold,
  ),
),
const SizedBox(height: 20),
const Text(
  'Productos:',
  style: TextStyle(
    color: Colors.white,
    fontSize: 16,
    fontWeight: FontWeight.bold,
  ),
),
const SizedBox(height: 12),
if (pedidoDetalle.detalles != null)
  ...pedidoDetalle.detalles!.map((detalle) {
    return Container(
      margin: const EdgeInsets.only(bottom: 8),
```

```
        padding: const EdgeInsets.all(12),
        decoration: BoxDecoration(
          color: const Color(0xFF0F0F0F),
          borderRadius: BorderRadius.circular(8),
        ),
        child: Row(
          mainAxisAlignment: MainAxisAlignment.spaceBetween,
          children: [
            Expanded(
              child: Column(
                crossAxisAlignment: CrossAxisAlignment.start,
                children: [
                  Text(
                    detalle.nombreProducto ?? 'Producto',
                    style: const TextStyle(
                      color: Colors.white,
                      fontWeight: FontWeight.w600,
                    ),
                  ),
                  Text(
                    'Cant: ${detalle.cantidad}',
                    style: TextStyle(color: Colors.grey[500]),
                  ),
                ],
              ),
            ),
            Text(
              '\$${detalle.subtotal.toStringAsFixed(2)}',
              style: const TextStyle(
                color: Color(0xFF66BB6A),
                fontWeight: FontWeight.bold,
              ),
            ),
          ],
        ),
      );
    }),
  const SizedBox(height: 20),
  Container(
    padding: const EdgeInsets.all(16),
    decoration: BoxDecoration(
      color: const Color(0xFF66BB6A).withOpacity(0.2),
      borderRadius: BorderRadius.circular(12),
    ),
    child: Row(
      mainAxisAlignment: MainAxisAlignment.spaceBetween,
      children: [
        const Text(
          'TOTAL',
          style: TextStyle(
            color: Colors.white,
            fontWeight: FontWeight.bold,
          ),
        ),
```

```
                  Text(
                    '\$${pedido.total.toStringAsFixed(2)}',
                    style: const TextStyle(
                      color: Color(0xFF66BB6A),
                      fontSize: 20,
                      fontWeight: FontWeight.bold,
                    ),
                  ),
                ],
              ),
            ),
          ],
        ),
      ),
    ),
  );
}

void _confirmarCancelar(Pedido pedido, PedidoViewModel viewModel) {
  showDialog(
    context: context,
    builder: (context) => AlertDialog(
      backgroundColor: const Color(0xFF1A1A1A),
      title: const Text(
        'Cancelar Pedido',
        style: TextStyle(color: Colors.white),
      ),
      content: Text(
        '¿Estás seguro de cancelar el pedido ${pedido.numeroPedido}?',
        style: TextStyle(color: Colors.grey[400]),
      ),
      actions: [
        TextButton(
          onPressed: () => Navigator.pop(context),
          child: const Text('No'),
        ),
        ElevatedButton(
          onPressed: () async {
            Navigator.pop(context);
            final success = await viewModel.cancelarPedido(pedido.id!);
            if (mounted && success) {
              ScaffoldMessenger.of(context).showSnackBar(
                const SnackBar(
                  content: Text('Pedido cancelado'),
                  backgroundColor: Color(0xFFFFA726),
                ),
              );
            }
          },
          style: ElevatedButton.styleFrom(
            backgroundColor: const Color(0xFFFF073A),
          ),
          child: const Text('Sí, cancelar'),
        ),
```

```
        ],
      ),
    );
  }

  Color _getEstadoColor(String estado) {
    switch (estado.toLowerCase()) {
      case 'pendiente':
        return const Color(0xFFFFA726);
      case 'procesando':
        return const Color(0xFF4A9EFF);
      case 'enviado':
        return const Color(0xFF9C27B0);
      case 'completado':
      case 'entregado':
        return const Color(0xFF66BB6A);
      case 'cancelado':
        return const Color(0xFFFF073A);
      default:
        return Colors.grey;
    }
  }

  String _formatearFecha(DateTime fecha) {
    final meses = [
      'Ene',
      'Feb',
      'Mar',
      'Abr',
      'May',
      'Jun',
      'Jul',
      'Ago',
      'Sep',
      'Oct',
      'Nov',
      'Dic'
    ];
    return '${fecha.day} ${meses[fecha.month - 1]} ${fecha.year}';
  }
}
```

12 SEGURIDAD

12.1 Permission Service

12.1.1 Control de permisos por rol.

```
class PermissionService {

  // Roles del sistema

  static const String ADMIN = 'admin';
```

static const String EMPLEADO = 'empleado';

static const String Cliente = 'cliente';

}

// Verificar si el usuario puede gest

12.2 Input Validators

12.2.1 Validación de entradas de usuario.

```
// utils/validators.dart
class Validadores {
  // Validar email
  static String? email(String? value) {
    if (value == null || value.isEmpty) {
      return 'El email es requerido';
    }
    final emailRegex = RegExp(
      r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$',
    );
    if (!emailRegex.hasMatch(value)) {
      return 'Ingrese un email válido';
    }
    return null;
  }
  // Validar contraseña
  static String? password(String? value) {
    if (value == null || value.isEmpty) {
      return 'La contraseña es requerida';
    }
    if (value.length < 6) {
      return 'La contraseña debe tener al menos 6 caracteres';
    }
    return null;
  }
  // Validar contraseña fuerte
  static String? passwordFuerte(String? value) {
    if (value == null || value.isEmpty) {
      return 'La contraseña es requerida';
    }
    if (value.length < 8) {
      return 'La contraseña debe tener al menos 8 caracteres';
    }
    if (!value.contains(RegExp(r'[A-Z]'))) {
      return 'Debe contener al menos una mayúscula';
    }
    if (!value.contains(RegExp(r'[a-z]'))) {
      return 'Debe contener al menos una minúscula';
    }
    if (!value.contains(RegExp(r'[0-9]'))) {
      return 'Debe contener al menos un número';
    }
    return null;
```

```
}
// Confirmar contraseña
static String? confirmarPassword(String? value, String password) {
  if (value == null || value.isEmpty) {
    return 'Confirme su contraseña';
  }
  if (value != password) {
    return 'Las contraseñas no coinciden';
  }
  return null;
}
// Validar nombre
static String? nombre(String? value) {
  if (value == null || value.isEmpty) {
    return 'El nombre es requerido';
  }
  if (value.length < 3) {
    return 'El nombre debe tener al menos 3 caracteres';
  }
  return null;
}
// Validar teléfono
static String? telefono(String? value) {
  if (value == null || value.isEmpty) {
    return null; // Opcional
  }
  final phoneRegex = RegExp(r'^\d{10}$');
  if (!phoneRegex.hasMatch(value.replaceAll(RegExp(r'[\s\-\(\)]'), ''))) {
    return 'Ingrese un teléfono válido (10 dígitos)';
  }

  return null;
}

// Validar campo requerido
static String? requerido(String? value, [String? nombreCampo]) {
  if (value == null || value.isEmpty) {
    return '${nombreCampo ?? 'Este campo'} es requerido';
  }
  return null;
}

// Validar número
static String? numero(String? value) {
  if (value == null || value.isEmpty) {
    return 'Este campo es requerido';
  }

  final numero = double.tryParse(value);
  if (numero == null) {
    return 'Ingrese un número válido';
  }

  return null;
```

```
}

// Validar número positivo
static String? numeroPositivo(String? value) {
  if (value == null || value.isEmpty) {
    return 'Este campo es requerido';
  }

  final numero = double.tryParse(value);
  if (numero == null) {
    return 'Ingrese un número válido';
  }

  if (numero <= 0) {
    return 'Debe ser mayor a 0';
  }

  return null;
}

// Validar entero
static String? entero(String? value) {
  if (value == null || value.isEmpty) {
    return 'Este campo es requerido';
  }

  final numero = int.tryParse(value);
  if (numero == null) {
    return 'Ingrese un número entero válido';
  }

  return null;
}

// Validar entero positivo
static String? enteroPositivo(String? value) {
  if (value == null || value.isEmpty) {
    return 'Este campo es requerido';
  }

  final numero = int.tryParse(value);
  if (numero == null) {
    return 'Ingrese un número entero válido';
  }

  if (numero <= 0) {
    return 'Debe ser mayor a 0';
  }

  return null;
}

// Validar longitud mínima
static String? longitudMinima(String? value, int minimo) {
```

```
    if (value == null || value.isEmpty) {
      return 'Este campo es requerido';
    }

    if (value.length < minimo) {
      return 'Debe tener al menos $minimo caracteres';
    }

    return null;
  }

  // Validar longitud máxima
  static String? longitudMaxima(String? value, int maximo) {
    if (value == null || value.isEmpty) {
      return null;
    }

    if (value.length > maximo) {
      return 'No debe exceder $maximo caracteres';
    }

    return null;
  }

  // Validar rango numérico
  static String? rangoNumerico(String? value, double min, double max) {
    if (value == null || value.isEmpty) {
      return 'Este campo es requerido';
    }

    final numero = double.tryParse(value);
    if (numero == null) {
      return 'Ingrese un número válido';
    }

    if (numero < min || numero > max) {
      return 'Debe estar entre $min y $max';
    }

    return null;
  }

  // Validar URL
  static String? url(String? value) {
    if (value == null || value.isEmpty) {
      return null; // Opcional
    }

    final urlRegex = RegExp(
      r'^https?:\/\/(www\.)?[-a-zA-Z0-9@:%._\+~#=]{1,256}\.[a-zA-Z0-
9()]{1,6}\b([-a-zA-Z0-9()@:%_\+.~#?&//=]*)$',
    );

    if (!urlRegex.hasMatch(value)) {
```

```
      return 'Ingrese una URL válida';
    }

    return null;
  }

  // Validar fecha
  static String? fecha(String? value) {
    if (value == null || value.isEmpty) {
      return 'La fecha es requerida';
    }

    try {
      DateTime.parse(value);
      return null;
    } catch (e) {
      return 'Ingrese una fecha válida';
    }
  }

  // Validar código (alfanumérico)
  static String? codigo(String? value) {
    if (value == null || value.isEmpty) {
      return 'El código es requerido';
    }

    final codigoRegex = RegExp(r'^[a-zA-Z0-9]+$');
    if (!codigoRegex.hasMatch(value)) {
      return 'Solo se permiten letras y números';
    }

    return null;
  }

  // Validador personalizado combinado
  static String? Function(String?) combinar(
    List<String? Function(String?)> validadores,
  ) {
    return (String? value) {
      for (final validador in validadores) {
        final error = validador(value);
        if (error != null) return error;
      }
      return null;
    };
  }
}
```

12.3 **Session Timeout**

```
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import 'dart:async';
import '../services/auth_service.dart';

/// Servicio para manejar el timeout automático de sesión
/// Cierra la sesión automáticamente después de un período de inactividad
class SessionTimeoutService {
  static const int _TIMEOUT_DURATION = 5 * 60 * 1000; // 5 minutos en
milisegundos

  Timer? _timer;
  BuildContext? _context;

  /// Inicializa el servicio con el contexto necesario
  void initialize(BuildContext context) {
    _context = context;
    _resetTimer();
    _setupUserActivityListener();
  }

  /// Reinicia el temporizador de timeout
  void _resetTimer() {
    _timer?.cancel();
    _timer = Timer(const Duration(milliseconds: _TIMEOUT_DURATION),
_handleTimeout);
  }

  /// Configura listeners para detectar actividad del usuario
  void _setupUserActivityListener() {
    if (_context == null) return;

    // Escuchar gestos del usuario para reiniciar el temporizador
    final binding = WidgetsBinding.instance;
    binding.addObserver(_UserActivityObserver(_resetTimer));
  }

  /// Maneja el timeout de sesión
  void _handleTimeout() {
    if (_context?.mounted == true) {
      final authService = _context!.read<AuthService>();
      authService.cerrarSesion();

      // Mostrar mensaje de timeout
      if (_context!.mounted) {
        ScaffoldMessenger.of(_context!).showSnackBar(
          const SnackBar(
            content: Text(
              'Sesión cerrada por inactividad (5 minutos)',
              style: TextStyle(color: Colors.white),
            ),
            backgroundColor: Color(0xFFE74C3C),
            behavior: SnackBarBehavior.floating,
```

```
            duration: Duration(seconds: 3),
            shape: RoundedRectangleBorder(
              borderRadius: BorderRadius.all(Radius.circular(12)),
            ),
          ),
        );

        // Redirigir al login
        Navigator.pushNamedAndRemoveUntil(
          _context!,
          '/login',
          (route) => false,
        );
      }
    }
  }

  /// Cancela el temporizador (usar al cerrar sesión manualmente)
  void cancel() {
    _timer?.cancel();
    _timer = null;
    _context = null;
  }
}

/// Observer para detectar actividad del usuario
class _UserActivityObserver extends WidgetsBindingObserver {
  final VoidCallback onUserActivity;

  _UserActivityObserver(this.onUserActivity);

  @override
  void didChangeAppLifecycleState(AppLifecycleState state) {
    super.didChangeAppLifecycleState(state);
    if (state == AppLifecycleState.resumed) {
      onUserActivity();
    }
  }

  @override
  void didHaveMemoryPressure() {
    super.didHaveMemoryPressure();
    onUserActivity();
  }
}
```

13 Utilidades

Formatters.dart

```
// utils/formatters.dart
import 'package:intl/intl.dart';

class Formateadores {
  // Formatear fecha
  static String fecha(DateTime fecha) {
```

```
    return DateFormat('dd/MM/yyyy').format(fecha);
  }

  // Formatear fecha con hora
  static String fechaHora(DateTime fecha) {
    return DateFormat('dd/MM/yyyy HH:mm').format(fecha);
  }

  // Formatear moneda
  static String moneda(double monto) {
    return NumberFormat.currency(
      locale: 'es_EC',
      symbol: '\$',
      decimalDigits: 2,
    ).format(monto);
  }

  // Formatear número
  static String numero(double numero, {int decimales = 2}) {
    return NumberFormat.decimalPattern('es_EC').format(numero);
  }
}
```

Responsive_helper.dart

```
import 'package:flutter/material.dart';

/// 📱 Ayudante responsivo simplificado
/// Uso: final resp = ResponsiveHelper(context);
/// resp.isMobile, resp.isTablet, resp.isDesktop
/// resp.paddingSmall, resp.paddingMedium, resp.paddingLarge
/// resp.fontSmall, resp.fontMedium, resp.fontLarge

class ResponsiveHelper {
  final BuildContext context;

  ResponsiveHelper(this.context);

  /// Ancho de pantalla
  double get screenWidth => MediaQuery.of(context).size.width;
  double get screenHeight => MediaQuery.of(context).size.height;

  /// Detectar dispositivo
  bool get isMobile => screenWidth < 600;
  bool get isTablet => screenWidth >= 600 && screenWidth < 1024;
  bool get isDesktop => screenWidth >= 1024;

  /// Padding según dispositivo
  double get paddingSmall => isMobile ? 8.0 : 12.0;
  double get paddingMedium => isMobile ? 16.0 : 24.0;
  double get paddingLarge => isMobile ? 24.0 : 32.0;

  /// Font sizes según dispositivo
  double get fontSmall => isMobile ? 12.0 : 13.0;
  double get fontMedium => isMobile ? 14.0 : 15.0;
```

```
  double get fontLarge => isMobile ? 16.0 : 18.0;
  double get fontXLarge => isMobile ? 18.0 : 22.0;
  double get fontTitle => isMobile ? 22.0 : 28.0;

  /// Espaciado entre elementos
  double get spacing => isMobile ? 8.0 : 12.0;
  double get spacingMedium => isMobile ? 12.0 : 16.0;
  double get spacingLarge => isMobile ? 16.0 : 24.0;

  /// Altura de botones
  double get buttonHeight => isMobile ? 44.0 : 48.0;
  double get buttonHeightSmall => isMobile ? 36.0 : 40.0;

  /// Tamaño de iconos
  double get iconSmall => isMobile ? 20.0 : 24.0;
  double get iconMedium => isMobile ? 24.0 : 28.0;
  double get iconLarge => isMobile ? 32.0 : 40.0;

  /// Border radius
  double get borderRadius => isMobile ? 8.0 : 12.0;
  double get borderRadiusSmall => isMobile ? 4.0 : 6.0;
  double get borderRadiusLarge => isMobile ? 12.0 : 16.0;

  /// Ancho máximo para contenidos
  double get maxWidth => isMobile ? double.infinity : (isTablet ? 600.0 :
1000.0);

  /// Grid columns
  int get gridColumns => isMobile ? 2 : (isTablet ? 3 : 4);

  /// Orientación
  bool get isLandscape => screenWidth > screenHeight;
  bool get isPortrait => screenWidth < screenHeight;
}

/// 🎯 Constructor rápido para widgets responsivos
extension ResponsiveExtension on BuildContext {
  ResponsiveHelper get responsive => ResponsiveHelper(this);
}
```

Responsive_util.dart

```
import 'package:flutter/material.dart';

///  ResponsiveUtil - Utilidad centralizada para diseño responsivo
///
/// Uso:
/// ```dart
/// final sizes = ResponsiveUtil.getSizes(context);
/// padding: EdgeInsets.all(sizes.padding),
/// fontSize: sizes.fontSize,
/// ```
class ResponsiveUtil {
  /// Obtener tamaños responsivos basado en ancho de pantalla
```

```
static ResponsiveSizes getSizes(BuildContext context) {
  final width = MediaQuery.of(context).size.width;
  final height = MediaQuery.of(context).size.height;
  final isTablet = width >= 600 && width < 1024;
  final isDesktop = width >= 1024;

  if (isDesktop) {
    return ResponsiveSizes(
      isMobile: false,
      isTablet: false,
      isDesktop: true,
      width: width,
      height: height,
      padding: 32.0,
      paddingSmall: 16.0,
      paddingLarge: 48.0,
      spacing: 24.0,
      spacingSmall: 12.0,
      spacingLarge: 32.0,
      fontSize: 16.0,
      fontSizeSmall: 12.0,
      fontSizeLarge: 20.0,
      fontSizeTitle: 36.0,
      fontSizeHeading: 28.0,
      fontSizeSubheading: 18.0,
      buttonHeight: 56.0,
      buttonHeightSmall: 44.0,
      iconSize: 32.0,
      iconSizeSmall: 24.0,
      iconSizeLarge: 48.0,
      borderRadius: 12.0,
      containerMaxWidth: 1200.0,
      cardMaxWidth: 600.0,
    );
  } else if (isTablet) {
    return ResponsiveSizes(
      isMobile: false,
      isTablet: true,
      isDesktop: false,
      width: width,
      height: height,
      padding: 24.0,
      paddingSmall: 12.0,
      paddingLarge: 36.0,
      spacing: 18.0,
      spacingSmall: 10.0,
      spacingLarge: 24.0,
      fontSize: 15.0,
      fontSizeSmall: 11.0,
      fontSizeLarge: 18.0,
      fontSizeTitle: 32.0,
      fontSizeHeading: 24.0,
      fontSizeSubheading: 16.0,
      buttonHeight: 52.0,
```

```
      buttonHeightSmall: 42.0,
      iconSize: 28.0,
      iconSizeSmall: 20.0,
      iconSizeLarge: 40.0,
      borderRadius: 10.0,
      containerMaxWidth: 800.0,
      cardMaxWidth: 500.0,
    );
  } else {
    // Mobile
    return ResponsiveSizes(
      isMobile: true,
      isTablet: false,
      isDesktop: false,
      width: width,
      height: height,
      padding: 16.0,
      paddingSmall: 8.0,
      paddingLarge: 24.0,
      spacing: 12.0,
      spacingSmall: 6.0,
      spacingLarge: 16.0,
      fontSize: 14.0,
      fontSizeSmall: 10.0,
      fontSizeLarge: 16.0,
      fontSizeTitle: 26.0,
      fontSizeHeading: 20.0,
      fontSizeSubheading: 14.0,
      buttonHeight: 48.0,
      buttonHeightSmall: 40.0,
      iconSize: 24.0,
      iconSizeSmall: 16.0,
      iconSizeLarge: 36.0,
      borderRadius: 8.0,
      containerMaxWidth: 500.0,
      cardMaxWidth: 100.0,
    );
  }
}

/// Colores responsivos (algunos deben ser más oscuros en mobile)
static Color getBackgroundColor(BuildContext context) {
  return const Color(0xFF0F0F0F);
}

static Color getCardColor(BuildContext context) {
  return const Color(0xFF1A1A1A);
}

static Color getPrimaryColor(BuildContext context) {
  return const Color(0xFFFF6B6B);
}

static Color getSecondaryColor(BuildContext context) {
```

```
    return const Color(0xFFFF8E53);
  }

  static Color getTextColor(BuildContext context) {
    return Colors.white;
  }

  static Color getTextSecondaryColor(BuildContext context) {
    return Colors.grey[400]!;
  }

  /// Alineación responsiva
  static MainAxisAlignment getMainAxisAlignment(BuildContext context) {
    final sizes = getSizes(context);
    return sizes.isMobile ? MainAxisAlignment.start :
MainAxisAlignment.center;
  }

  static CrossAxisAlignment getCrossAxisAlignment(BuildContext context) {
    final sizes = getSizes(context);
    return sizes.isMobile ? CrossAxisAlignment.start :
CrossAxisAlignment.center;
  }

  /// Grid columns responsivos
  static int getGridColumns(BuildContext context) {
    final sizes = getSizes(context);
    if (sizes.isDesktop) return 4;
    if (sizes.isTablet) return 3;
    return 2; // mobile
  }

  /// Aspect ratio responsivo para imágenes
  static double getImageAspectRatio(BuildContext context) {
    final sizes = getSizes(context);
    if (sizes.isDesktop) return 16 / 9;
    if (sizes.isTablet) return 4 / 3;
    return 1 / 1; // mobile - cuadrado
  }

  /// Obtener EdgeInsets responsivos
  static EdgeInsets getPadding(BuildContext context) {
    final sizes = getSizes(context);
    return EdgeInsets.all(sizes.padding);
  }

  static EdgeInsets getSymmetricPadding(BuildContext context,
      {bool vertical = true}) {
    final sizes = getSizes(context);
    if (vertical) {
      return EdgeInsets.symmetric(vertical: sizes.padding);
    } else {
      return EdgeInsets.symmetric(horizontal: sizes.padding);
    }
```

```
  }

  /// Altura mínima para formularios
  static double getFormMinHeight(BuildContext context) {
    final height = MediaQuery.of(context).size.height;
    return height * 0.7;
  }

  /// Ancho máximo para contenedores
  static double getMaxWidth(BuildContext context) {
    final sizes = getSizes(context);
    return sizes.containerMaxWidth;
  }
}

/// 🎁 Clase para almacenar tamaños responsivos
class ResponsiveSizes {
  final bool isMobile;
  final bool isTablet;
  final bool isDesktop;
  final double width;
  final double height;
  final double padding;
  final double paddingSmall;
  final double paddingLarge;
  final double spacing;
  final double spacingSmall;
  final double spacingLarge;
  final double fontSize;
  final double fontSizeSmall;
  final double fontSizeLarge;
  final double fontSizeTitle;
  final double fontSizeHeading;
  final double fontSizeSubheading;
  final double buttonHeight;
  final double buttonHeightSmall;
  final double iconSize;
  final double iconSizeSmall;
  final double iconSizeLarge;
  final double borderRadius;
  final double containerMaxWidth;
  final double cardMaxWidth;

  ResponsiveSizes({
    required this.isMobile,
    required this.isTablet,
    required this.isDesktop,
    required this.width,
    required this.height,
    required this.padding,
    required this.paddingSmall,
    required this.paddingLarge,
    required this.spacing,
    required this.spacingSmall,
```

```
    required this.spacingLarge,
    required this.fontSize,
    required this.fontSizeSmall,
    required this.fontSizeLarge,
    required this.fontSizeTitle,
    required this.fontSizeHeading,
    required this.fontSizeSubheading,
    required this.buttonHeight,
    required this.buttonHeightSmall,
    required this.iconSize,
    required this.iconSizeSmall,
    required this.iconSizeLarge,
    required this.borderRadius,
    required this.containerMaxWidth,
    required this.cardMaxWidth,
  });
}
```

Validators.dart

```
// utils/validators.dart
class Validadores {
  // Validar email
  static String? email(String? value) {
    if (value == null || value.isEmpty) {
      return 'El email es requerido';
    }
    final emailRegex = RegExp(
      r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$',
    );
    if (!emailRegex.hasMatch(value)) {
      return 'Ingrese un email válido';
    }
    return null;
  }
  // Validar contraseña
  static String? password(String? value) {
    if (value == null || value.isEmpty) {
      return 'La contraseña es requerida';
    }
    if (value.length < 6) {
      return 'La contraseña debe tener al menos 6 caracteres';
    }
    return null;
  }
  // Validar contraseña fuerte
  static String? passwordFuerte(String? value) {
    if (value == null || value.isEmpty) {
      return 'La contraseña es requerida';
    }
    if (value.length < 8) {
      return 'La contraseña debe tener al menos 8 caracteres';
    }
    if (!value.contains(RegExp(r'[A-Z]'))) {
      return 'Debe contener al menos una mayúscula';
```

```
  }
  if (!value.contains(RegExp(r'[a-z]'))) {
    return 'Debe contener al menos una minúscula';
  }
  if (!value.contains(RegExp(r'[0-9]'))) {
    return 'Debe contener al menos un número';
  }
  return null;
}
// Confirmar contraseña
static String? confirmarPassword(String? value, String password) {
  if (value == null || value.isEmpty) {
    return 'Confirme su contraseña';
  }
  if (value != password) {
    return 'Las contraseñas no coinciden';
  }
  return null;
}
// Validar nombre
static String? nombre(String? value) {
  if (value == null || value.isEmpty) {
    return 'El nombre es requerido';
  }
  if (value.length < 3) {
    return 'El nombre debe tener al menos 3 caracteres';
  }
  return null;
}
// Validar teléfono
static String? telefono(String? value) {
  if (value == null || value.isEmpty) {
    return null; // Opcional
  }
  final phoneRegex = RegExp(r'^\d{10}$');
  if (!phoneRegex.hasMatch(value.replaceAll(RegExp(r'[\s\-\(\)]'), ''))) {
    return 'Ingrese un teléfono válido (10 dígitos)';
  }

  return null;
}

// Validar campo requerido
static String? requerido(String? value, [String? nombreCampo]) {
  if (value == null || value.isEmpty) {
    return '${nombreCampo ?? 'Este campo'} es requerido';
  }
  return null;
}

// Validar número
static String? numero(String? value) {
  if (value == null || value.isEmpty) {
    return 'Este campo es requerido';
```

```
  }

  final numero = double.tryParse(value);
  if (numero == null) {
    return 'Ingrese un número válido';
  }

  return null;
}

// Validar número positivo
static String? numeroPositivo(String? value) {
  if (value == null || value.isEmpty) {
    return 'Este campo es requerido';
  }

  final numero = double.tryParse(value);
  if (numero == null) {
    return 'Ingrese un número válido';
  }

  if (numero <= 0) {
    return 'Debe ser mayor a 0';
  }

  return null;
}

// Validar entero
static String? entero(String? value) {
  if (value == null || value.isEmpty) {
    return 'Este campo es requerido';
  }

  final numero = int.tryParse(value);
  if (numero == null) {
    return 'Ingrese un número entero válido';
  }

  return null;
}

// Validar entero positivo
static String? enteroPositivo(String? value) {
  if (value == null || value.isEmpty) {
    return 'Este campo es requerido';
  }

  final numero = int.tryParse(value);
  if (numero == null) {
    return 'Ingrese un número entero válido';
  }

  if (numero <= 0) {
```

```
    return 'Debe ser mayor a 0';
  }

  return null;
}

// Validar longitud mínima
static String? longitudMinima(String? value, int minimo) {
  if (value == null || value.isEmpty) {
    return 'Este campo es requerido';
  }

  if (value.length < minimo) {
    return 'Debe tener al menos $minimo caracteres';
  }

  return null;
}

// Validar longitud máxima
static String? longitudMaxima(String? value, int maximo) {
  if (value == null || value.isEmpty) {
    return null;
  }

  if (value.length > maximo) {
    return 'No debe exceder $maximo caracteres';
  }

  return null;
}

// Validar rango numérico
static String? rangoNumerico(String? value, double min, double max) {
  if (value == null || value.isEmpty) {
    return 'Este campo es requerido';
  }

  final numero = double.tryParse(value);
  if (numero == null) {
    return 'Ingrese un número válido';
  }

  if (numero < min || numero > max) {
    return 'Debe estar entre $min y $max';
  }

  return null;
}

// Validar URL
static String? url(String? value) {
  if (value == null || value.isEmpty) {
    return null; // Opcional
```

```
    }

    final urlRegex = RegExp(
      r'^https?:\/\/(www\.)?[-a-zA-Z0-9@:%._\+~#=]{1,256}\.[a-zA-Z0-
9()]{1,6}\b([-a-zA-Z0-9()@:%_\+.~#?&//=]*)$',
    );

    if (!urlRegex.hasMatch(value)) {
      return 'Ingrese una URL válida';
    }

    return null;
  }

  // Validar fecha
  static String? fecha(String? value) {
    if (value == null || value.isEmpty) {
      return 'La fecha es requerida';
    }

    try {
      DateTime.parse(value);
      return null;
    } catch (e) {
      return 'Ingrese una fecha válida';
    }
  }

  // Validar código (alfanumérico)
  static String? codigo(String? value) {
    if (value == null || value.isEmpty) {
      return 'El código es requerido';
    }

    final codigoRegex = RegExp(r'^[a-zA-Z0-9]+$');
    if (!codigoRegex.hasMatch(value)) {
      return 'Solo se permiten letras y números';
    }

    return null;
  }

  // Validador personalizado combinado
  static String? Function(String?) combinar(
    List<String? Function(String?)> validadores,
  ) {
    return (String? value) {
      for (final validador in validadores) {
        final error = validador(value);
        if (error != null) return error;
      }
      return null;
    };
  }
```

```
}
```

14 Navegación de Rutas

App_routes.dart

```
// core/app_routes.dart
// ✅ Rutas organizadas por roles y funcionalidades

class AppRoutes {
  // ======================================
  // RUTAS DE AUTENTICACIÓN
  // ======================================

  static const String login = '/login';
  static const String registro = '/registro';
  static const String forgotPassword = '/forgot-password';
  static const String resetPassword = '/reset-password';

  // ======================================
  // RUTAS PRINCIPALES - DASHBOARDS POR ROL
  // ======================================

  static const String home = '/'; // Redirige según rol
  static const String adminDashboard = '/admin-dashboard';
  static const String vendedorDashboard = '/vendedor-dashboard';
  static const String clienteDashboard = '/cliente-dashboard';

  // ======================================
  // RUTAS DE ADMIN
  // ======================================

  // Tabs del admin
  static const String adminPedidos = '/admin-pedidos';
  static const String adminProductos = '/admin-productos';
  static const String adminReportes = '/admin-reportes';
  static const String adminUsuarios = '/admin-usuarios';
  static const String adminVentas = '/admin-ventas';

  // Gestión de productos (admin)
  static const String productoDetalle = '/producto-detalle';
  static const String productoNuevo = '/producto-nuevo';
  static const String productoEditar = '/producto-editar';

  // Gestión de ventas (admin)
  static const String ventaDetalle = '/venta-detalle';
  static const String ventaNueva = '/venta-nueva';

  // Gestión de usuarios (admin)
  static const String usuarioDetalle = '/usuario-detalle';
  static const String usuarioNuevo = '/usuario-nuevo';
  static const String usuarioEditar = '/usuario-editar';

  // Reportes
  static const String reporteVentas = '/reporte-ventas';
  static const String reporteInventario = '/reporte-inventario';
```

```
// ======================================
// RUTAS DE VENDEDOR
// ======================================

// Tabs del vendedor
static const String vendedorClientes = '/vendedor-clientes';
static const String vendedorCrearVenta = '/vendedor-crear-venta';
static const String vendedorProductos = '/vendedor-productos';
static const String vendedorVentas = '/vendedor-ventas';
static const String vendedorPedidos = '/vendedor-pedidos';

// Gestión de clientes (vendedor)
static const String clienteDetalle = '/cliente-detalle';
static const String clienteNuevo = '/cliente-nuevo';
static const String clienteEditar = '/cliente-editar';

// ======================================
// RUTAS DE CLIENTE
// ======================================

static const String catalogo = '/catalogo';
static const String carrito = '/carrito';
static const String misPedidos = '/mis-pedidos';

// Detalle de pedido (compartido entre roles)
static const String pedidoDetalle = '/pedido-detalle';

// ======================================
// RUTAS DE PERFIL (Todos los roles)
// ======================================

static const String perfilCliente = '/perfil-cliente';
static const String perfilUsuario = '/perfil-usuario';

// ======================================
// RUTAS DE CONFIGURACIÓN
// ======================================

static const String configuracion = '/configuracion';
static const String configuracionUsuarios = '/configuracion-usuarios';

// ======================================
// HELPERS - Navegación según rol
// ======================================

/// Retorna el dashboard correspondiente según el rol
static String getDashboardByRole(String rol) {
  switch (rol.toLowerCase()) {
    case 'admin':
    case 'administrador':
      return adminDashboard;
    case 'vendedor':
      return vendedorDashboard;
```

```
    case 'cliente':
      return clienteDashboard;
    default:
      return clienteDashboard;
  }
}

/// Rutas exclusivas de admin
static bool requiresAdmin(String route) {
  return [
    adminDashboard,
    adminPedidos,
    adminProductos,
    adminReportes,
    adminUsuarios,
    adminVentas,
    usuarioNuevo,
    usuarioEditar,
    usuarioDetalle,
    reporteVentas,
    reporteInventario,
    configuracion,
    configuracionUsuarios,
  ].contains(route);
}

/// Rutas para admin o vendedor
static bool requiresAdminOrVendedor(String route) {
  return [
    vendedorClientes,
    vendedorCrearVenta,
    vendedorProductos,
    vendedorVentas,
    vendedorPedidos,
    clienteNuevo,
    clienteEditar,
    clienteDetalle,
    productoDetalle,
    productoNuevo,
    productoEditar,
    ventaNueva,
    ventaDetalle,
  ].contains(route);
}

/// Rutas públicas/cliente
static bool isClienteRoute(String route) {
  return [
    clienteDashboard,
    catalogo,
    carrito,
    misPedidos,
    pedidoDetalle,
    perfilCliente,
```

```
    ].contains(route);
  }

  /// Obtiene la ruta de perfil según el rol
  static String getPerfilByRole(String rol) {
    if (rol.toLowerCase() == 'cliente') {
      return perfilCliente;
    }
    return perfilUsuario;
  }
}

// ========================================
// ENUMS PARA ROLES
// ========================================

enum UserRole {
  admin,
  vendedor,
  cliente;

  String get displayName {
    switch (this) {
      case UserRole.admin:
        return 'Administrador';
      case UserRole.vendedor:
        return 'Vendedor';
      case UserRole.cliente:
        return 'Cliente';
    }
  }

  String get dashboardRoute {
    switch (this) {
      case UserRole.admin:
        return AppRoutes.adminDashboard;
      case UserRole.vendedor:
        return AppRoutes.vendedorDashboard;
      case UserRole.cliente:
        return AppRoutes.clienteDashboard;
    }
  }
}
```

App_theme.dart

```
import 'package:aztecafest1/core/constants.dart';
import 'package:flutter/material.dart';

class AppTheme {
  // ========================================
  // TEMA OSCURO (Estilo Admin Dashboard)
  // ========================================
  static ThemeData get temaOscuro {
    return ThemeData(
```

```
      brightness: Brightness.dark,
      primaryColor: AppConstants.colorPrimario,
      scaffoldBackgroundColor: AppConstants.fondoOscuro,
      colorScheme: const ColorScheme.dark(
        primary: AppConstants.colorPrimario,
        secondary: AppConstants.colorSecundario,
        error: AppConstants.colorError,
        surface: AppConstants.fondoTarjeta,
        background: AppConstants.fondoOscuro,
      ),

      // AppBar
      appBarTheme: const AppBarTheme(
        backgroundColor: AppConstants.fondoMedio,
        foregroundColor: AppConstants.textoBlanco,
        elevation: 0,
        centerTitle: true,
        titleTextStyle: TextStyle(
          fontSize: AppConstants.fuenteGrande,
          fontWeight: FontWeight.bold,
          color: AppConstants.textoBlanco,
        ),
      ),

      // Tarjetas
      cardTheme: CardThemeData(
        color: AppConstants.fondoTarjeta,
        elevation: 2,
        margin: const EdgeInsets.symmetric(
          horizontal: AppConstants.espaciadoMedio,
          vertical: AppConstants.espaciadoPequeno,
        ),
        shape: RoundedRectangleBorder(
          borderRadius:
BorderRadius.circular(AppConstants.bordeRedondeadoMedio),
        ),
      ),

      // Botones elevados
      elevatedButtonTheme: ElevatedButtonThemeData(
        style: ElevatedButton.styleFrom(
          backgroundColor: AppConstants.colorPrimario,
          foregroundColor: Colors.white,
          padding: const EdgeInsets.symmetric(
            horizontal: AppConstants.espaciadoGrande,
            vertical: AppConstants.espaciadoMedio,
          ),
          shape: RoundedRectangleBorder(
            borderRadius:
BorderRadius.circular(AppConstants.bordeRedondeadoMedio),
          ),
          elevation: 2,
        ),
      ),
```

```
      // Campos de texto
      inputDecorationTheme: InputDecorationTheme(
        filled: true,
        fillColor: AppConstants.fondoInput,
        border: OutlineInputBorder(
          borderRadius:
BorderRadius.circular(AppConstants.bordeRedondeadoMedio),
          borderSide: const BorderSide(color: AppConstants.textoGrisOscuro),
        ),
        enabledBorder: OutlineInputBorder(
          borderRadius:
BorderRadius.circular(AppConstants.bordeRedondeadoMedio),
          borderSide: const BorderSide(color: AppConstants.textoGrisOscuro),
        ),
        focusedBorder: OutlineInputBorder(
          borderRadius:
BorderRadius.circular(AppConstants.bordeRedondeadoMedio),
          borderSide: const BorderSide(color: AppConstants.colorPrimario,
width: 2),
        ),
        errorBorder: OutlineInputBorder(
          borderRadius:
BorderRadius.circular(AppConstants.bordeRedondeadoMedio),
          borderSide: const BorderSide(color: AppConstants.colorError),
        ),
        contentPadding: const EdgeInsets.symmetric(
          horizontal: AppConstants.espaciadoMedio,
          vertical: AppConstants.espaciadoMedio,
        ),
        hintStyle: const TextStyle(color: AppConstants.textoGrisMedio),
        labelStyle: const TextStyle(color: AppConstants.textoGrisClaro),
      ),

      // Texto
      textTheme: const TextTheme(
        displayLarge: TextStyle(
          fontSize: AppConstants.fuenteTitulo,
          fontWeight: FontWeight.bold,
          color: AppConstants.textoBlanco,
        ),
        titleLarge: TextStyle(
          fontSize: AppConstants.fuenteGrande,
          fontWeight: FontWeight.w600,
          color: AppConstants.textoBlanco,
        ),
        bodyLarge: TextStyle(
          fontSize: AppConstants.fuenteNormal,
          color: AppConstants.textoGrisClaro,
        ),
        bodyMedium: TextStyle(
          fontSize: AppConstants.fuenteNormal,
          color: AppConstants.textoGrisMedio,
        ),
```

```
      ),
    );
  }

  // ======================================
  // TEMA CLARO (Para compatibilidad)
  // ======================================
  static ThemeData get temaClaro {
    return ThemeData(
      primaryColor: AppConstants.colorPrimario,
      scaffoldBackgroundColor: AppConstants.fondoClaro,
      colorScheme: ColorScheme.light(
        primary: AppConstants.colorPrimario,
        secondary: AppConstants.colorSecundario,
        error: AppConstants.colorError,
      ),

      // AppBar
      appBarTheme: const AppBarTheme(
        backgroundColor: AppConstants.colorPrimario,
        foregroundColor: Colors.white,
        elevation: 2,
        centerTitle: true,
        titleTextStyle: TextStyle(
          fontSize: AppConstants.fuenteGrande,
          fontWeight: FontWeight.bold,
          color: Colors.white,
        ),
      ),

      // Tarjetas
      cardTheme: CardThemeData(
        color: AppConstants.fondoTarjeta,
        elevation: 2,
        margin: const EdgeInsets.symmetric(
          horizontal: AppConstants.espaciadoMedio,
          vertical: AppConstants.espaciadoPequeno,
        ),
        shape: RoundedRectangleBorder(
          borderRadius:
BorderRadius.circular(AppConstants.bordeRedondeadoMedio),
        ),
      ),

      // Botones elevados
      elevatedButtonTheme: ElevatedButtonThemeData(
        style: ElevatedButton.styleFrom(
          backgroundColor: AppConstants.colorPrimario,
          foregroundColor: Colors.white,
          padding: const EdgeInsets.symmetric(
            horizontal: AppConstants.espaciadoGrande,
            vertical: AppConstants.espaciadoMedio,
          ),
          shape: RoundedRectangleBorder(
```

```
            borderRadius:
BorderRadius.circular(AppConstants.bordeRedondeadoMedio),
          ),
          elevation: 2,
        ),
      ),

      // Campos de texto
      inputDecorationTheme: InputDecorationTheme(
        filled: true,
        fillColor: Colors.white,
        border: OutlineInputBorder(
          borderRadius:
BorderRadius.circular(AppConstants.bordeRedondeadoMedio),
          borderSide: const BorderSide(color: AppConstants.textoClaro),
        ),
        enabledBorder: OutlineInputBorder(
          borderRadius:
BorderRadius.circular(AppConstants.bordeRedondeadoMedio),
          borderSide: const BorderSide(color: AppConstants.textoClaro),
        ),
        focusedBorder: OutlineInputBorder(
          borderRadius:
BorderRadius.circular(AppConstants.bordeRedondeadoMedio),
          borderSide: const BorderSide(color: AppConstants.colorPrimario,
width: 2),
        ),
        errorBorder: OutlineInputBorder(
          borderRadius:
BorderRadius.circular(AppConstants.bordeRedondeadoMedio),
          borderSide: const BorderSide(color: AppConstants.colorError),
        ),
        contentPadding: const EdgeInsets.symmetric(
          horizontal: AppConstants.espaciadoMedio,
          vertical: AppConstants.espaciadoMedio,
        ),
      ),

      // Texto
      textTheme: const TextTheme(
        displayLarge: TextStyle(
          fontSize: AppConstants.fuenteTitulo,
          fontWeight: FontWeight.bold,
          color: AppConstants.textoOscuro,
        ),
        titleLarge: TextStyle(
          fontSize: AppConstants.fuenteGrande,
          fontWeight: FontWeight.w600,
          color: AppConstants.textoOscuro,
        ),
        bodyLarge: TextStyle(
          fontSize: AppConstants.fuenteNormal,
          color: AppConstants.textoOscuro,
        ),
```

```
        bodyMedium: TextStyle(
          fontSize: AppConstants.fuenteNormal,
          color: AppConstants.textoMedio,
        ),
      ),
    );
  }
}
```

Constants.dart

```
import 'package:flutter/material.dart';

class AppConstants {
  // ======================================
  // COLORES PRINCIPALES (Estilo Admin Dashboard)
  // ======================================
  // Gradiente principal
  static const Color colorPrimario = Color(0xFFFF6B6B);
  static const Color colorSecundario = Color(0xFFFF8E53);

  // Colores de estado
  static const Color colorExito = Color(0xFF4CAF50);
  static const Color colorError = Color(0xFFF44336);
  static const Color colorAdvertencia = Color(0xFFFF9800);
  static const Color colorInfo = Color(0xFF2196F3);

  // ======================================
  // COLORES DE FONDO (Dark Theme)
  // ======================================
  static const Color fondoOscuro = Color(0xFF0F0F0F);        // Fondo
principal oscuro
  static const Color fondoMedio = Color(0xFF1A1A1A);         // AppBar, cards
principales
  static const Color fondoTarjeta = Color(0xFF2A2A2A);       // Tarjetas y
contenedores
  static const Color fondoInput = Color(0xFF1E1E1E);         // Campos de
texto

  // Compatibilidad (para pantallas que aún no se migraron)
  static const Color fondoClaro = Color(0xFFF5F5F5);
  static const Color fondoBlanco = Colors.white;

  // ======================================
  // COLORES DE TEXTO (Dark Theme)
  // ======================================
  static const Color textoBlanco = Colors.white;             // Texto
principal en dark
  static const Color textoGrisClaro = Color(0xFFE0E0E0);     // Texto
secundario
  static const Color textoGrisMedio = Color(0xFFB0B0B0);     // Texto
deshabilitado/hints
  static const Color textoGrisOscuro = Color(0xFF757575);    // Texto
terciario
```

```
// Compatibilidad (para pantallas claras)
static const Color textoOscuro = Color(0xFF212121);
static const Color textoMedio = Color(0xFF757575);
static const Color textoClaro = Color(0xFFBDBDBD);

// ======================================
// INFORMACIÓN DE LA APLICACIÓN
// ======================================
static const String nombreApp = 'AztecaFest';
static const String versionApp = '1.0.0';

// ======================================
// MENSAJES GENERALES
// ======================================
static const String mensajeCargando = 'Cargando...';
static const String mensajeError = 'Ha ocurrido un error';
static const String mensajeExito = 'Operación exitosa';
static const String mensajeSinDatos = 'No hay datos disponibles';
static const String mensajeConfirmar = '¿Está seguro?';

// ======================================
// VALIDACIONES
// ======================================
static const int longitudMinimaPassword = 6;
static const int longitudMaximaNombre = 100;
static const int longitudMaximaDescripcion = 500;

// ======================================
// CONFIGURACIÓN DE PAGINACIÓN
// ======================================
static const int elementosPorPagina = 20;

// ======================================
// ROLES DE USUARIO
// ✅ VERIFICADO: Consistente con base de datos real
// ======================================

// Roles principales (valores exactos de BD)
static const String rolAdmin = 'admin';
static const String rolVendedor = 'vendedor';
static const String rolCliente = 'cliente';

// Alias para compatibilidad (opcional)
static const String rolAdministrador = 'admin';

// Lista de roles válidos
static const List<String> rolesValidos = [
  rolAdmin,
  rolVendedor,
  rolCliente,
];

// Lista de roles de usuario (admin y vendedor)
static const List<String> rolesUsuario = [
```

```
  rolAdmin,
  rolVendedor,
];

// ======================================
// MÉTODOS HELPER PARA ROLES
// ======================================

/// Verifica si un rol es de administrador
static bool esAdmin(String? rol) {
  if (rol == null) return false;
  final rolLower = rol.toLowerCase();
  return rolLower == 'admin' || rolLower == 'administrador';
}

/// Verifica si un rol es de vendedor
static bool esVendedor(String? rol) {
  if (rol == null) return false;
  return rol.toLowerCase() == 'vendedor';
}

/// Verifica si un rol es de cliente
static bool esCliente(String? rol) {
  if (rol == null) return false;
  return rol.toLowerCase() == 'cliente';
}

/// Verifica si un rol tiene permisos de gestión (admin o vendedor)
static bool tienePermisosGestion(String? rol) {
  return esAdmin(rol) || esVendedor(rol);
}

/// Normaliza el rol a su formato estándar
static String normalizarRol(String rol) {
  final rolLower = rol.toLowerCase();
  if (rolLower == 'admin' || rolLower == 'administrador') {
    return rolAdmin;
  }
  if (rolLower == 'vendedor') {
    return rolVendedor;
  }
  return rolCliente;
}

/// Obtiene el nombre de visualización del rol
static String getNombreRol(String? rol) {
  if (rol == null) return 'Desconocido';

  if (esAdmin(rol)) return 'Administrador';
  if (esVendedor(rol)) return 'Vendedor';
  if (esCliente(rol)) return 'Cliente';

  return rol; // Retorna el rol original si no coincide
}
```

```
// ======================================
// ESTADOS DE VENTA
// ======================================
static const String ventaCompletada = 'completada';
static const String ventaCancelada = 'cancelada';
static const String ventaPendiente = 'pendiente';

static const List<String> estadosVenta = [
  ventaPendiente,
  ventaCompletada,
  ventaCancelada,
];

// ======================================
// ESTADOS DE PEDIDO
// ======================================
static const String pedidoPendiente = 'pendiente';
static const String pedidoProcesando = 'procesando';
static const String pedidoEnviado = 'enviado';
static const String pedidoEntregado = 'entregado';
static const String pedidoCancelado = 'cancelado';

static const List<String> estadosPedido = [
  pedidoPendiente,
  pedidoProcesando,
  pedidoEnviado,
  pedidoEntregado,
  pedidoCancelado,
];

// ======================================
// MÉTODOS DE PAGO
// ======================================
static const String pagoEfectivo = 'efectivo';
static const String pagoTarjeta = 'tarjeta';
static const String pagoTransferencia = 'transferencia';

static const List<String> metodosPago = [
  pagoEfectivo,
  pagoTarjeta,
  pagoTransferencia,
];

// ======================================
// ESPACIADO
// ======================================
static const double espaciadoPequeno = 8.0;
static const double espaciadoMedio = 16.0;
static const double espaciadoGrande = 24.0;
static const double espaciadoExtraGrande = 32.0;

// ======================================
// BORDES REDONDEADOS
```

```
// ======================================
static const double bordeRedondeadoPequeno = 4.0;
static const double bordeRedondeadoMedio = 8.0;
static const double bordeRedondeadoGrande = 12.0;
static const double bordeRedondeadoExtraGrande = 16.0;

// ======================================
// TAMAÑOS DE FUENTE
// ======================================
static const double fuentePequena = 12.0;
static const double fuenteNormal = 14.0;
static const double fuenteMediana = 16.0;
static const double fuenteGrande = 18.0;
static const double fuenteTitulo = 24.0;
static const double fuenteEncabezado = 32.0;

// ======================================
// ICONOS POR ROL
// ======================================
static IconData getIconoRol(String? rol) {
  if (esAdmin(rol)) return Icons.admin_panel_settings;
  if (esVendedor(rol)) return Icons.point_of_sale;
  if (esCliente(rol)) return Icons.person;
  return Icons.help_outline;
}

// ======================================
// COLORES POR ROL
// ======================================
static Color getColorRol(String? rol) {
  if (esAdmin(rol)) return const Color(0xFFE74C3C); // Rojo
  if (esVendedor(rol)) return const Color(0xFF3498DB); // Azul
  if (esCliente(rol)) return const Color(0xFF2ECC71); // Verde
  return Colors.grey;
}

// ======================================
// IMPUESTOS
// ======================================
static const double ivaEcuador = 0.12; // 12% IVA en Ecuador
static const double ivaDefault = 0.16; // 16% IVA por defecto

// ======================================
// LÍMITES Y RESTRICCIONES
// ======================================
static const int stockMinimo = 0;
static const int stockMaximo = 9999;
static const double precioMinimo = 0.01;
static const double precioMaximo = 999999.99;
static const int cantidadMinimaCarrito = 1;
static const int cantidadMaximaCarrito = 100;

// ======================================
// FORMATOS
```

```
// ======================================
static const String formatoFecha = 'dd/MM/yyyy';
static const String formatoFechaHora = 'dd/MM/yyyy HH:mm';
static const String formatoMoneda = '\$';

// ======================================
// RESPONSIVIDAD (Breakpoints)
// ======================================
static const double breakpointMobile = 600.0;     // < 600 = Móvil
static const double breakpointTablet = 800.0;     // 600-800 = Tablet
static const double breakpointDesktop = 1200.0;   // > 800 = Escritorio

/// Helper para obtener tamaños responsivos
/// Retorna un Map con: titleSize, iconSize, fontSize, avatarSize, padding
static Map<String, dynamic> getResponsiveSizes(double width) {
  if (width > breakpointTablet) {
    // Desktop
    return {
      'titleSize': 22.0,
      'iconSize': 26.0,
      'fontSize': 16.0,
      'avatarSize': 44.0,
      'padding': 16.0,
      'cardPadding': 20.0,
      'buttonHeight': 50.0,
    };
  } else if (width > breakpointMobile) {
    // Tablet
    return {
      'titleSize': 20.0,
      'iconSize': 24.0,
      'fontSize': 15.0,
      'avatarSize': 40.0,
      'padding': 14.0,
      'cardPadding': 16.0,
      'buttonHeight': 46.0,
    };
  } else {
    // Móvil
    return {
      'titleSize': 18.0,
      'iconSize': 22.0,
      'fontSize': 14.0,
      'avatarSize': 36.0,
      'padding': 12.0,
      'cardPadding': 12.0,
      'buttonHeight': 44.0,
    };
  }
}

/// Verifica si es móvil
static bool isMobile(double width) => width < breakpointMobile;
```

```
  /// Verifica si es tablet
  static bool isTablet(double width) => width >= breakpointMobile && width <
breakpointTablet;

  /// Verifica si es escritorio
  static bool isDesktop(double width) => width >= breakpointTablet;
}
```

Responsive_helper.dart

```
import 'package:flutter/material.dart';
import 'constants.dart';

/// 📱 Helper para construir UIs responsivas
/// Evita advertencias de overflow y adapta elementos según el dispositivo
class ResponsiveHelper {
  /// Obtiene el ancho de la pantalla
  static double getWidth(BuildContext context) {
    return MediaQuery.of(context).size.width;
  }

  /// Obtiene el alto de la pantalla
  static double getHeight(BuildContext context) {
    return MediaQuery.of(context).size.height;
  }

  /// Verifica si es móvil
  static bool isMobile(BuildContext context) {
    return getWidth(context) < AppConstants.breakpointMobile;
  }

  /// Verifica si es tablet
  static bool isTablet(BuildContext context) {
    final width = getWidth(context);
    return width >= AppConstants.breakpointMobile &&
        width < AppConstants.breakpointTablet;
  }

  /// Verifica si es escritorio
  static bool isDesktop(BuildContext context) {
    return getWidth(context) >= AppConstants.breakpointTablet;
  }

  /// Obtiene tamaños responsivos
  static Map<String, dynamic> getSizes(BuildContext context) {
    return AppConstants.getResponsiveSizes(getWidth(context));
  }

  /// Padding responsivo horizontal
  static EdgeInsets horizontalPadding(BuildContext context) {
    final sizes = getSizes(context);
    return EdgeInsets.symmetric(horizontal: sizes['padding']);
  }

  /// Padding responsivo vertical
```

```
static EdgeInsets verticalPadding(BuildContext context) {
  final sizes = getSizes(context);
  return EdgeInsets.symmetric(vertical: sizes['padding']);
}

/// Padding responsivo completo
static EdgeInsets allPadding(BuildContext context) {
  final sizes = getSizes(context);
  return EdgeInsets.all(sizes['padding']);
}

/// Tamaño de fuente responsivo
static double fontSize(BuildContext context, {double? custom}) {
  if (custom != null) return custom;
  final sizes = getSizes(context);
  return sizes['fontSize'];
}

/// Tamaño de título responsivo
static double titleSize(BuildContext context) {
  final sizes = getSizes(context);
  return sizes['titleSize'];
}

/// Tamaño de icono responsivo
static double iconSize(BuildContext context) {
  final sizes = getSizes(context);
  return sizes['iconSize'];
}

/// Container/Card con padding responsivo
static Widget responsiveContainer({
  required BuildContext context,
  required Widget child,
  Color? color,
  EdgeInsets? customPadding,
  double? borderRadius,
}) {
  final sizes = getSizes(context);
  return Container(
    padding: customPadding ?? EdgeInsets.all(sizes['cardPadding']),
    decoration: BoxDecoration(
      color: color ?? AppConstants.fondoTarjeta,
      borderRadius: BorderRadius.circular(
        borderRadius ?? AppConstants.bordeRedondeadoMedio,
      ),
    ),
    child: child,
  );
}

/// TextField con constraints responsivos
static Widget responsiveTextField({
  required BuildContext context,
```

```
  required TextEditingController controller,
  String? labelText,
  String? hintText,
  IconData? prefixIcon,
  bool obscureText = false,
  TextInputType? keyboardType,
  String? Function(String?)? validator,
  int maxLines = 1,
}) {
  final sizes = getSizes(context);
  return ConstrainedBox(
    constraints: BoxConstraints(
      maxWidth: isDesktop(context) ? 400 : double.infinity,
    ),
    child: TextFormField(
      controller: controller,
      obscureText: obscureText,
      keyboardType: keyboardType,
      validator: validator,
      maxLines: maxLines,
      style: TextStyle(
        fontSize: sizes['fontSize'],
        color: AppConstants.textoBlanco,
      ),
      decoration: InputDecoration(
        labelText: labelText,
        hintText: hintText,
        prefixIcon: prefixIcon != null
            ? Icon(prefixIcon, size: sizes['iconSize'])
            : null,
      ),
    ),
  );
}

/// Botón responsivo
static Widget responsiveButton({
  required BuildContext context,
  required String text,
  required VoidCallback onPressed,
  Color? backgroundColor,
  Color? textColor,
  IconData? icon,
  bool fullWidth = true,
}) {
  final sizes = getSizes(context);
  final button = ElevatedButton(
    onPressed: onPressed,
    style: ElevatedButton.styleFrom(
      backgroundColor: backgroundColor ?? AppConstants.colorPrimario,
      foregroundColor: textColor ?? Colors.white,
      padding: EdgeInsets.symmetric(
        horizontal: sizes['padding'] * 2,
        vertical: sizes['padding'],
```

```
      ),
      minimumSize: Size(0, sizes['buttonHeight']),
    ),
    child: Row(
      mainAxisSize: fullWidth ? MainAxisSize.max : MainAxisSize.min,
      mainAxisAlignment: MainAxisAlignment.center,
      children: [
        if (icon != null) ...[
          Icon(icon, size: sizes['iconSize']),
          const SizedBox(width: 8),
        ],
        Text(
          text,
          style: TextStyle(
            fontSize: sizes['fontSize'],
            fontWeight: FontWeight.bold,
          ),
        ),
      ],
    ),
  );

  if (fullWidth) {
    return SizedBox(width: double.infinity, child: button);
  }
  return button;
}

/// AppBar responsiva con gradiente
static PreferredSizeWidget responsiveAppBar({
  required BuildContext context,
  required String title,
  List<Widget>? actions,
  Widget? leading,
  bool showBackButton = true,
}) {
  final sizes = getSizes(context);
  return AppBar(
    backgroundColor: AppConstants.fondoMedio,
    elevation: 0,
    automaticallyImplyLeading: showBackButton,
    leading: leading,
    title: Text(
      title,
      style: TextStyle(
        fontSize: sizes['titleSize'],
        fontWeight: FontWeight.bold,
        color: AppConstants.textoBlanco,
      ),
    ),
    actions: actions,
  );
}
```

```
  /// Espaciado responsivo
  static SizedBox verticalSpace(BuildContext context, {double multiplier =
1}) {
    final sizes = getSizes(context);
    return SizedBox(height: sizes['padding'] * multiplier);
  }

  static SizedBox horizontalSpace(BuildContext context,
      {double multiplier = 1}) {
    final sizes = getSizes(context);
    return SizedBox(width: sizes['padding'] * multiplier);
  }

  /// Grid responsivo (número de columnas según dispositivo)
  static int getGridCrossAxisCount(BuildContext context) {
    if (isDesktop(context)) return 4;
    if (isTablet(context)) return 3;
    return 2;
  }

  /// Ancho máximo para formularios (evita que se estiren demasiado en
desktop)
  static double getMaxFormWidth(BuildContext context) {
    return isDesktop(context) ? 600 : double.infinity;
  }
}
```

15 Migración de Datos

Migrattions Service

```
// lib/services/migration_service.dart
import 'dart:convert';
import 'dart:io';
import 'package:path_provider/path_provider.dart';
import 'package:sqflite/sqflite.dart';
import '../data/database_helper.dart';

class MigrationService {
  static final MigrationService instance = MigrationService._init();
  MigrationService._init();

  /// Exporta TODOS los datos de la base de datos a un archivo JSON
  Future<File> exportarDatos({String? nombreArchivo}) async {
    try {
      final db = await DatabaseHelper.instance.database;

      // Obtener todos los datos de todas las tablas
      final Map<String, dynamic> datosExportacion = {
        'version': 6,
        'fecha_exportacion': DateTime.now().toIso8601String(),
        'datos': {
          'usuarios': await db.query('usuarios'),
          'productos': await db.query('productos'),
          'ventas': await db.query('ventas'),
          'detalle_ventas': await db.query('detalle_ventas'),
```

```
          'pedidos': await db.query('pedidos'),
          'detalle_pedidos': await db.query('detalle_pedidos'),
        },
        'metadatos': {
          'total_usuarios': await _contarRegistros(db, 'usuarios'),
          'total_productos': await _contarRegistros(db, 'productos'),
          'total_ventas': await _contarRegistros(db, 'ventas'),
          'total_pedidos': await _contarRegistros(db, 'pedidos'),
        }
      };

      // Convertir a JSON
      final jsonString = JsonEncoder.withIndent('
').convert(datosExportacion);

      // Guardar en archivo
      final directorio = await getApplicationDocumentsDirectory();
      final nombre = nombreArchivo ??
'backup_aztecafest_${DateTime.now().millisecondsSinceEpoch}.json';
      final archivo = File('${directorio.path}/$nombre');

      await archivo.writeAsString(jsonString);

      print('✅ Exportación exitosa: ${archivo.path}');
      print('📊 Total de registros exportados:');
      print('   - Usuarios:
${datosExportacion['metadatos']['total_usuarios']}');
      print('   - Productos:
${datosExportacion['metadatos']['total_productos']}');
      print('   - Ventas: ${datosExportacion['metadatos']['total_ventas']}');
      print('   - Pedidos:
${datosExportacion['metadatos']['total_pedidos']}');

      return archivo;
    } catch (e) {
      print('❌ Error al exportar datos: $e');
      rethrow;
    }
  }

  /// Importa datos desde un archivo JSON
  Future<bool> importarDatos(String rutaArchivo, {bool reemplazarExistentes =
false}) async {
    try {
      final archivo = File(rutaArchivo);

      if (!await archivo.exists()) {
        throw Exception('El archivo no existe: $rutaArchivo');
      }

      // Leer archivo JSON
      final jsonString = await archivo.readAsString();
      final Map<String, dynamic> datosImportados = json.decode(jsonString);
```

```
      // Validar versión
      final versionImportada = datosImportados['version'] as int?;
      print('🎁 Versión de datos importados: $versionImportada');

      final db = await DatabaseHelper.instance.database;

      // Si se desea reemplazar datos existentes, limpiar tablas
      if (reemplazarExistentes) {
        await _limpiarTablas(db);
      }

      // Importar datos en orden correcto (respetando foreign keys)
      final datos = datosImportados['datos'] as Map<String, dynamic>;

      await db.transaction((txn) async {
        // 1. Usuarios (no tiene dependencias)
        await _importarTabla(txn, 'usuarios', datos['usuarios'] as List);

        // 2. Productos (no tiene dependencias)
        await _importarTabla(txn, 'productos', datos['productos'] as List);

        // 3. Ventas (depende de usuarios)
        await _importarTabla(txn, 'ventas', datos['ventas'] as List);

        // 4. Detalle ventas (depende de ventas y productos)
        await _importarTabla(txn, 'detalle_ventas', datos['detalle_ventas']
as List);

        // 5. Pedidos (depende de usuarios)
        await _importarTabla(txn, 'pedidos', datos['pedidos'] as List);

        // 6. Detalle pedidos (depende de pedidos y productos)
        await _importarTabla(txn, 'detalle_pedidos', datos['detalle_pedidos']
as List);
      });

      print('✅ Importación completada exitosamente');
      print('📊 Resumen:');
      print('   - Usuarios importados: ${(datos['usuarios'] as
List).length}');
      print('   - Productos importados: ${(datos['productos'] as
List).length}');
      print('   - Ventas importadas: ${(datos['ventas'] as List).length}');
      print('   - Pedidos importados: ${(datos['pedidos'] as List).length}');

      return true;
    } catch (e) {
      print('❌ Error al importar datos: $e');
      return false;
    }
  }

  /// Importa tabla específica
  Future<void> _importarTabla(DatabaseExecutor db, String tabla, List
```

```
registros) async {
    int importados = 0;
    int omitidos = 0;

    for (var registro in registros) {
      try {
        final Map<String, dynamic> datos = Map<String,
dynamic>.from(registro);
        await db.insert(
          tabla,
          datos,
          conflictAlgorithm: ConflictAlgorithm.replace,
        );
        importados++;
      } catch (e) {
        print('⚠ Error al importar registro en $tabla: $e');
        omitidos++;
      }
    }

    print('   ✓ $tabla: $importados importados, $omitidos omitidos');
  }

  /// Limpia todas las tablas antes de importar
  Future<void> _limpiarTablas(Database db) async {
    print('🗑 Limpiando tablas existentes...');

    await db.transaction((txn) async {
      // Eliminar en orden inverso a las dependencias
      await txn.delete('detalle_pedidos');
      await txn.delete('detalle_ventas');
      await txn.delete('pedidos');
      await txn.delete('ventas');
      await txn.delete('productos');
      await txn.delete('usuarios');
    });

    print('✅ Tablas limpiadas');
  }

  /// Cuenta registros en una tabla
  Future<int> _contarRegistros(Database db, String tabla) async {
    final resultado = await db.rawQuery('SELECT COUNT(*) as count FROM
$tabla');
    return Sqflite.firstIntValue(resultado) ?? 0;
  }

  /// Obtiene información del último backup
  Future<Map<String, dynamic>?> obtenerInfoBackup(String rutaArchivo) async {
    try {
      final archivo = File(rutaArchivo);

      if (!await archivo.exists()) {
        return null;
```

```
      }

      final jsonString = await archivo.readAsString();
      final Map<String, dynamic> datos = json.decode(jsonString);

      return {
        'version': datos['version'],
        'fecha_exportacion': datos['fecha_exportacion'],
        'metadatos': datos['metadatos'],
        'tamano_bytes': await archivo.length(),
      };
    } catch (e) {
      print('❌ Error al leer info del backup: $e');
      return null;
    }
  }

  /// Lista todos los archivos de backup disponibles
  Future<List<FileSystemEntity>> listarBackups() async {
    try {
      final directorio = await getApplicationDocumentsDirectory();
      final archivos = directorio.listSync()
          .where((f) => f.path.contains('backup_aztecafest') &&
f.path.endsWith('.json'))
          .toList();

      // Ordenar por fecha (más reciente primero)
      archivos.sort((a, b) => b.path.compareTo(a.path));

      return archivos;
    } catch (e) {
      print('❌ Error al listar backups: $e');
      return [];
    }
  }

  /// Crea un backup automático con nombre basado en fecha
  Future<String> crearBackupAutomatico() async {
    final ahora = DateTime.now();
    final nombre = 'backup_aztecafest_'
        '${ahora.year}${ahora.month.toString().padLeft(2,
'0')}${ahora.day.toString().padLeft(2, '0')}_'
        '${ahora.hour.toString().padLeft(2,
'0')}${ahora.minute.toString().padLeft(2, '0')}.json';

    final archivo = await exportarDatos(nombreArchivo: nombre);
    return archivo.path;
  }

  /// Exporta solo datos específicos (por ejemplo, solo productos)
  Future<File> exportarTablaEspecifica(String nombreTabla, {String?
nombreArchivo}) async {
    try {
      final db = await DatabaseHelper.instance.database;
```

```
      final datos = await db.query(nombreTabla);

      final Map<String, dynamic> exportacion = {
        'tabla': nombreTabla,
        'fecha_exportacion': DateTime.now().toIso8601String(),
        'registros': datos,
        'total': datos.length,
      };

      final jsonString = JsonEncoder.withIndent('  ').convert(exportacion);

      final directorio = await getApplicationDocumentsDirectory();
      final nombre = nombreArchivo ??
'export_${nombreTabla}_${DateTime.now().millisecondsSinceEpoch}.json';
      final archivo = File('${directorio.path}/$nombre');

      await archivo.writeAsString(jsonString);

      print('✅ Exportación de $nombreTabla exitosa: ${archivo.path}');
      print('📊 Total de registros: ${datos.length}');

      return archivo;
    } catch (e) {
      print('❌ Error al exportar tabla $nombreTabla: $e');
      rethrow;
    }
  }

  /// Valida la integridad de un archivo de backup
  Future<bool> validarBackup(String rutaArchivo) async {
    try {
      final archivo = File(rutaArchivo);

      if (!await archivo.exists()) {
        print('❌ El archivo no existe');
        return false;
      }

      final jsonString = await archivo.readAsString();
      final datos = json.decode(jsonString);

      // Validar estructura básica
      if (!datos.containsKey('version') ||
          !datos.containsKey('datos') ||
          !datos.containsKey('metadatos')) {
        print('❌ Estructura de backup inválida');
        return false;
      }

      final datosTablas = datos['datos'] as Map<String, dynamic>;
      final tablasRequeridas = ['usuarios', 'productos', 'ventas',
'detalle_ventas', 'pedidos', 'detalle_pedidos'];
```

```
      for (var tabla in tablasRequeridas) {
        if (!datosTablas.containsKey(tabla)) {
          print('❌ Falta tabla: $tabla');
          return false;
        }
      }

      print('✅ Backup válido');
      return true;
    } catch (e) {
      print('❌ Error al validar backup: $e');
      return false;
    }
  }
}
```