REPÚBLICA DEL ECUADOR

"DISEÑO Y DESARROLLO DE UNA TIENDA VIRTUAL BÁSICA PARA PEQUEÑOS ARTESANOS CON CARRITO DE COMPRAS Y PASARELA DE PAGO SIMULADA"

MODALIDAD COMPLEXIVO PREVIO A LA OBTENCIÓN DEL TÍTULO DE TECNÓLOGO EN DESARROLLO DE SOFTWARE

AUTOR: ANDI CALAPUCHA RICARDO LEONEL

TUTOR: ING. GONZALO GUANIPATÍN

Tena - Ecuador

2025

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

APROBACIÓN DEL TUTOR

ING. AGUSTÍN GONZALO GUANIPATÍN RAMÍREZ

PROFESOR DEL INSTITUTO SUPERIOR TECNOLÓGICO TENA.

CERTIFICA:

En calidad de Tutor Examen de carácter complexivo práctico denominado: “Diseño y desarrollo de una tienda virtual básica con un carrito de compras y pasarela de pago simulada”, de autoría del señor Andi Calapucha Ricardo Leonel, con CC. 1501038408 estudiante de la Carrera de Tecnología Superior en Desarrollo de Software del Instituto Superior Tecnológico Tena, CERTIFICO que se ha realizado la revisión prolija del Examen de carácter complexivo práctico antes citado, cumple con los requisitos de fondo y de forma que exigen el respectivo reglamento e institución.

Tena, 14 de julio de 2025

Ing. Agustín Gonzalo Guanipatín Ramírez

TUTOR DEL EXAMEN DE CARACTER COMPLEXIVO PRÁCTICO

RESUMEN

Este proyecto de titulación aborda la brecha digital que enfrentan los pequeños artesanos mediante el diseño y desarrollo de una plataforma de comercio electrónico robusta, escalable y moderna. Se ha construido una aplicación web full-stack utilizando un stack tecnológico basado en Next.js, React y TypeScript, garantizando una experiencia de usuario fluida y un rendimiento optimizado para los motores de búsqueda (SEO). El sistema implementado ofrece un catálogo de productos dinámico, gestión de usuarios con un sistema de autenticación seguro basado JSON Web Tokens (JWT), un carrito de compras con persistencia de datos y un completo panel de administración para la gestión autónoma de la tienda. El objetivo fundamental es proporcionar una solución tecnológica de nivel profesional que no solo sirva como un prototipo funcional con pasarela de pago simulada, sino que también establezca una base arquitectónica sólida para su escalamiento a un entorno de producción con transacciones reales, impulsando así la transformación digital del comercio artesanal local.

Palabras clave: Tienda virtual, Next.js, React, TypeScript, JWT, MySQL, Tailwind CSS, comercio electrónico, desarrollo web full-stack, artesanía local.

ABSTRACT

This thesis project addresses the digital divide faced by small artisans by designing and developing a robust, scalable, and modern e-commerce platform. A full-stack web application was built using a technology stack based on Next.js, React, and TypeScript, ensuring a fluid user experience and optimized performance for search engines (SEO) The system offers the implementation of a dynamic product catalog, user management with a secure authentication system based on JSON Web Tokens (JWT), a shopping cart with data persistence, and a complete administration panel for autonomous store management. The fundamental objective is to provide a professional-grade technological solution that not only serves as a functional prototype with a simulated payment gateway, but also establishes a solid architectural foundation for scaling to a production environment with real transactions, thus driving the digital transformation of local artisan commerce.

Keywords: Virtual store, Next.js, React, TypeScript, JWT, MySQL, Tailwind CSS, e-comerce, full-stack web development, local crafts.

Revieward by

BA. Ana Carolina Romero Alava

C.I 1313245217

Language Center Professor

1. INTRODUCCIÓN

En una época donde el uso de la tecnología ha dejado de ser opcional para convertirse en una necesidad, incluso para actividades tradicionales como la artesanía. Muchos artesanos enfrentan una realidad desafiante: elaboran productos de gran valor cultural, pero no cuentan con las herramientas necesarias para comercializarlos en el entorno digital. Esta brecha tecnológica limita sus oportunidades de venta, visibilidad y crecimiento económico.

El desarrollo de una tienda básica se plantea como una respuesta concreta y accesible. La idea no es reemplazar el contacto humano propio de los mercados artesanales, sino complementarlo con una plataforma que les permita en un entorno seguro y controlado.

Esta propuesta construye desde la formación práctica en desarrollo de software y responde a la necesidad de diseñar soluciones tecnológicas aplicadas a la vida real. Su relevancia radica que no solo cumple un objetivo académico, sino que también busca generar un impacto social positivo en los pequeños emprendedores locales.

El objetivo general de este proyecto es desarrollar una aplicación web funcional que conecte a los artesanos con el mundo digital, permitiendo gestionar productos y pagos simulados de manera eficiente y amigable para el usuario.

2. ANÁLISIS

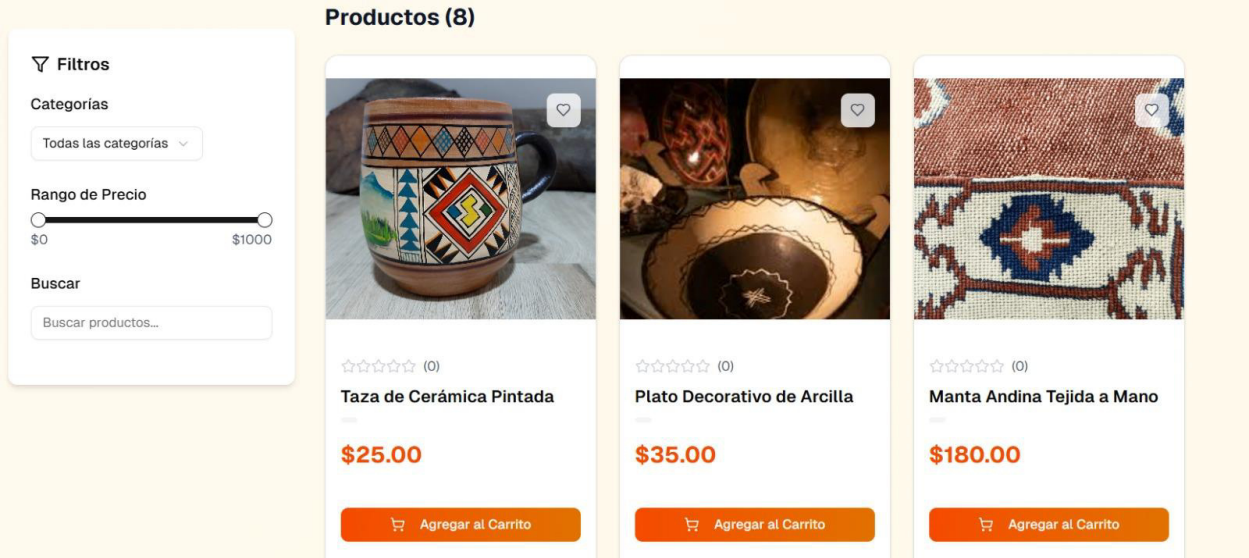
El sistema desarrollado, denominado "Artesian Store", es una tienda virtual concebida como una aplicación web full-stack. A diferencia de un enfoque tradicional con PHP y XAMPP, esta versión utiliza un stack de desarrollo moderno para ofrecer una solución más robusta y profesional. Durante la fase de pruebas funcionales, se realizaron evaluaciones exhaustivas para verificar el correcto desempeño de cada uno de los módulos implementados, logrando una plataforma cohesiva, eficiente y segura.

Durante la etapa de pruebas, se realizaron evaluaciones funcionales para verificar el correcto desempeño de cada uno de los módulos implementados. Entre los resultados más relevantes se encuentran:

2.1 MÓDULOS DESAROLLADOS Y VALIDADOS

- **Catálogo de productos:** Permite a los usuarios visualizar los productos de la tienda de forma dinámica. Se validó que las imágenes, nombres, precios y categorías se muestren correctamente, cargando la información de forma asíncrona desde la base de datos. La interfaz, desarrollada en el archivo app/products/page.tsx, incluye filtros funcionales que mejoran la experiencia de búsqueda del usuario.

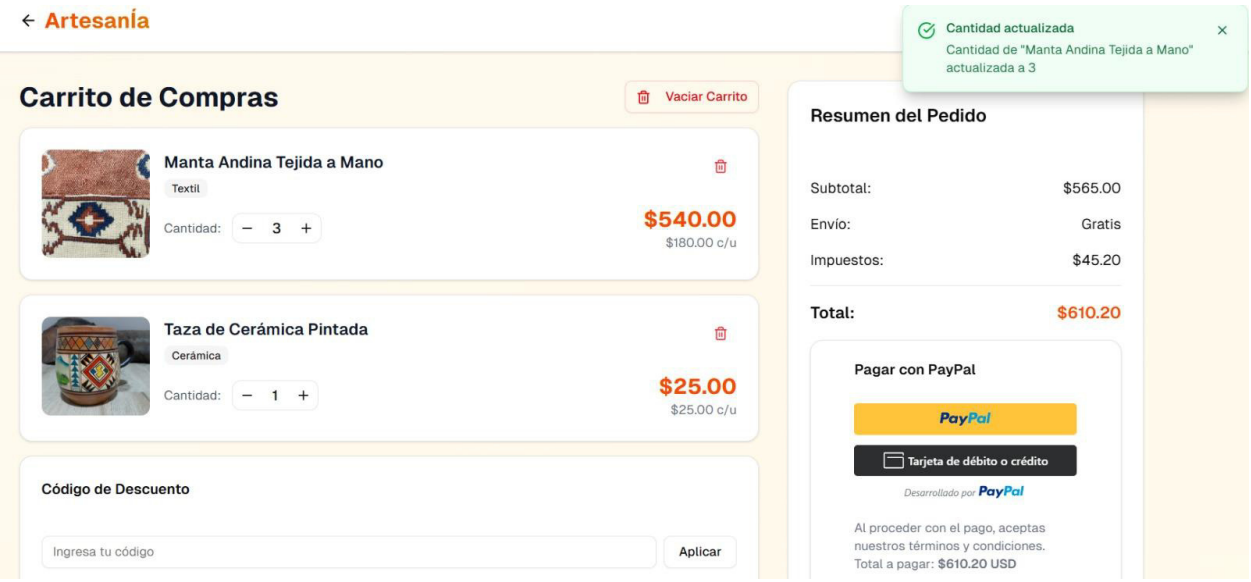
Figura 1*Vista del catálogo de productos.*



Nota 1 *La interfaz está construida con componentes de React y estilizada con Tailwind CSS, garantizando un diseño responsivo. Los datos de los productos, incluyendo las imágenes, se cargan dinámicamente desde la base de datos a través de la API RESTful implementada en app/api/products/route.ts.*

- **Carrito de compras**: Los usuarios autenticados pueden agregar productos al carrito, modificar las cantidades y ver el subtotal y total en tiempo real. Se validó que el estado del carrito es persistente entre sesiones, gracias a su gestión con React Context (contexts/CartContext.tsx) y su sincronización con la base de datos a través de los endpoints en app/api/cart/route.ts.

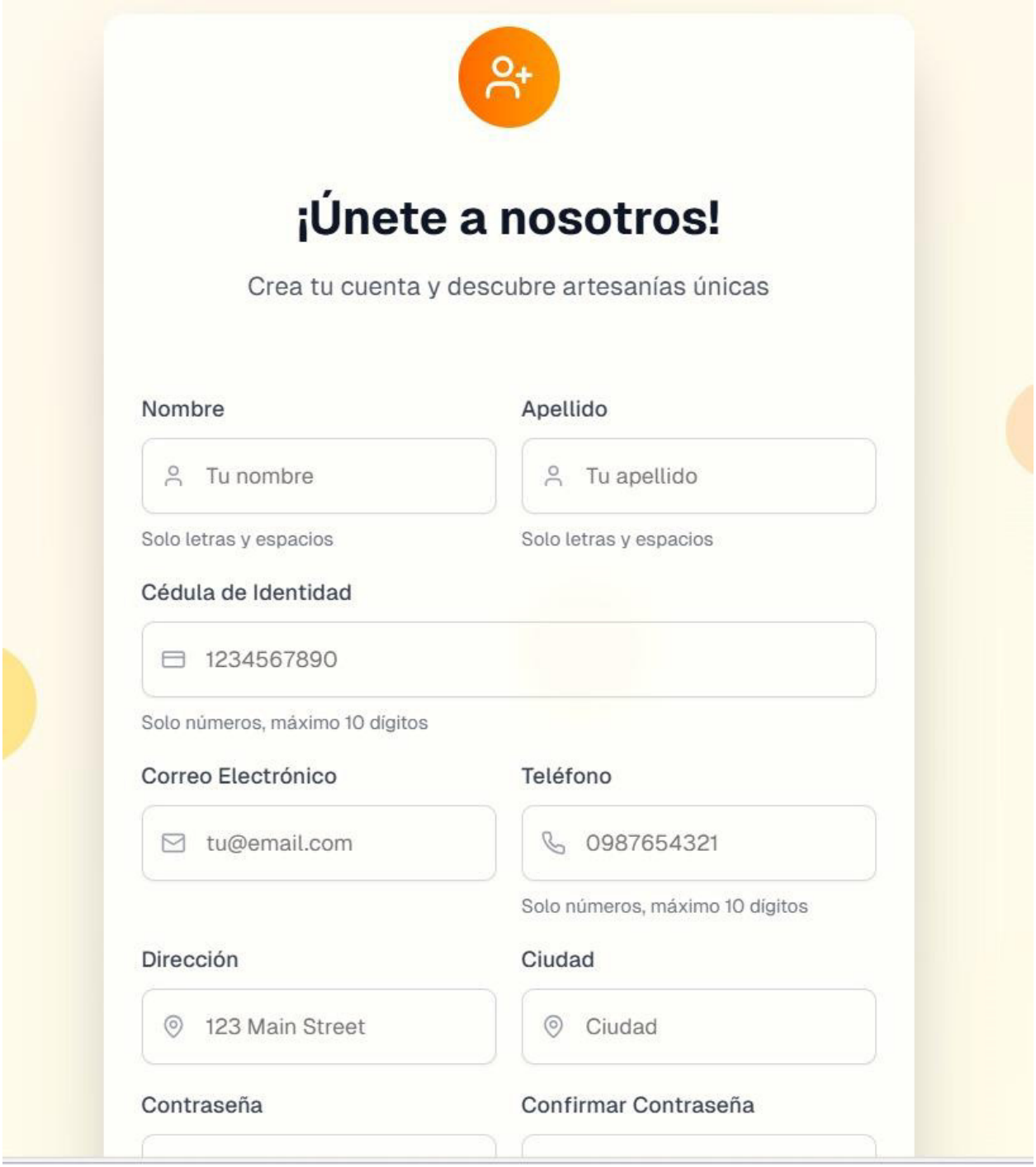
Figura 2 *Funcionalidad del carrito de compras.*



Nota 2 *El estado del carrito se gestiona globalmente, permitiendo que el contador en el header se actualice en tiempo real desde cualquier página, mejorando la experiencia de usuario.*

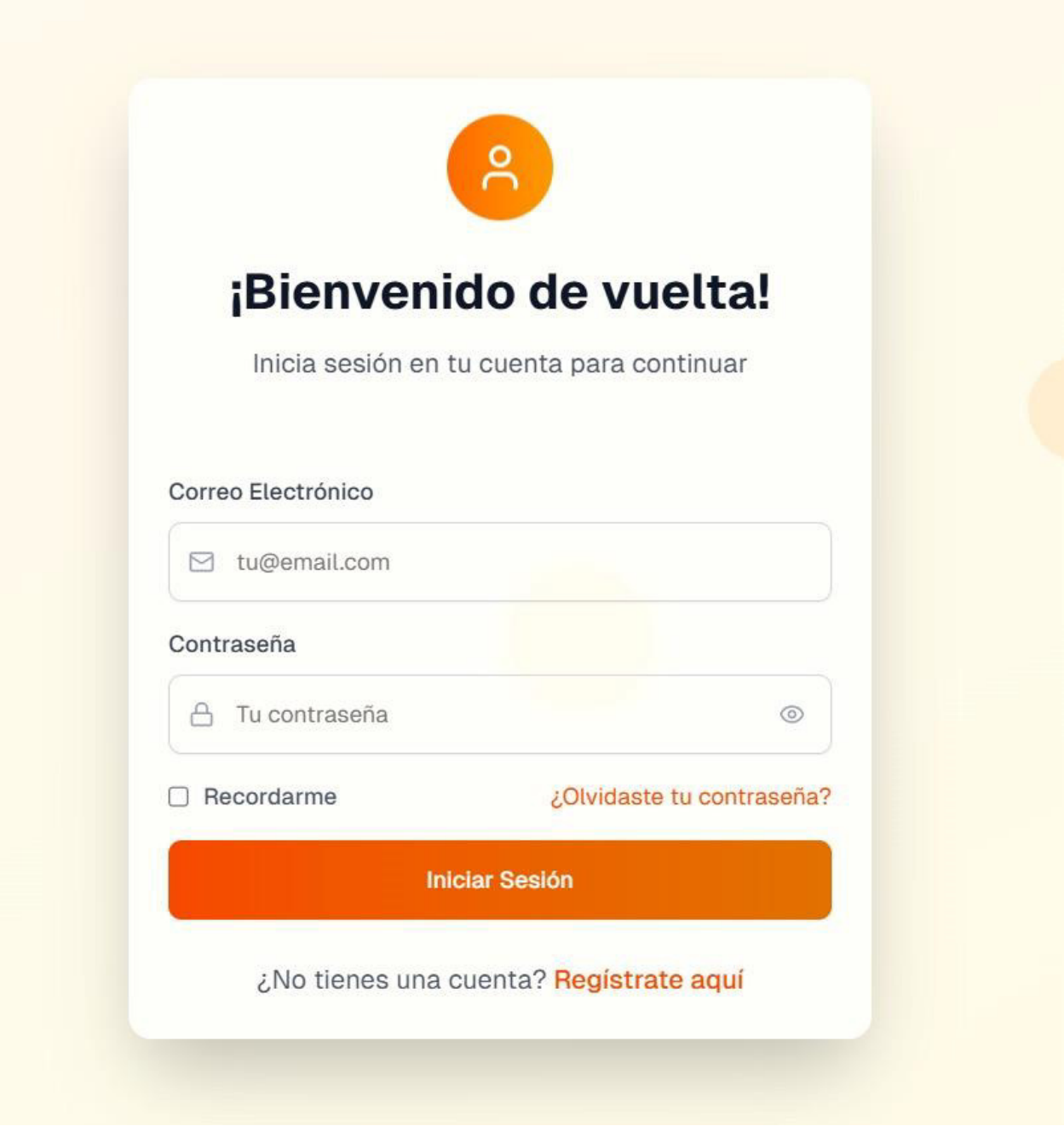
- **Registro e inicio de sesión:** Se implementó un sistema de autenticación seguro. Las pruebas confirmaron que los usuarios pueden registrarse, iniciar sesión y mantener su sesión activa.
- **Registro:** El formulario en app/auth/register/page.tsx captura y valida los datos, que son enviados a la API (app/api/auth/register/route.ts) para su procesamiento.
- **Inicio sesión:** Las credenciales son verificadas, y si son correctas, se genera un JSON Web Token (JWT) para gestionar la sesión.
- **Seguridad:** Las contraseñas se almacenan de forma segura utilizando el algoritmo de hashing bcrypt.js.

Figura 3 *Formulario de registro de nuevos usuarios.*



Nota 3 *El formulario incluye validaciones del lado del cliente y del servidor para asegurar la integridad de los datos. Los campos como cédula y teléfono tienen validaciones específicas para aceptar solo números y una longitud máxima de 10 dígitos.*

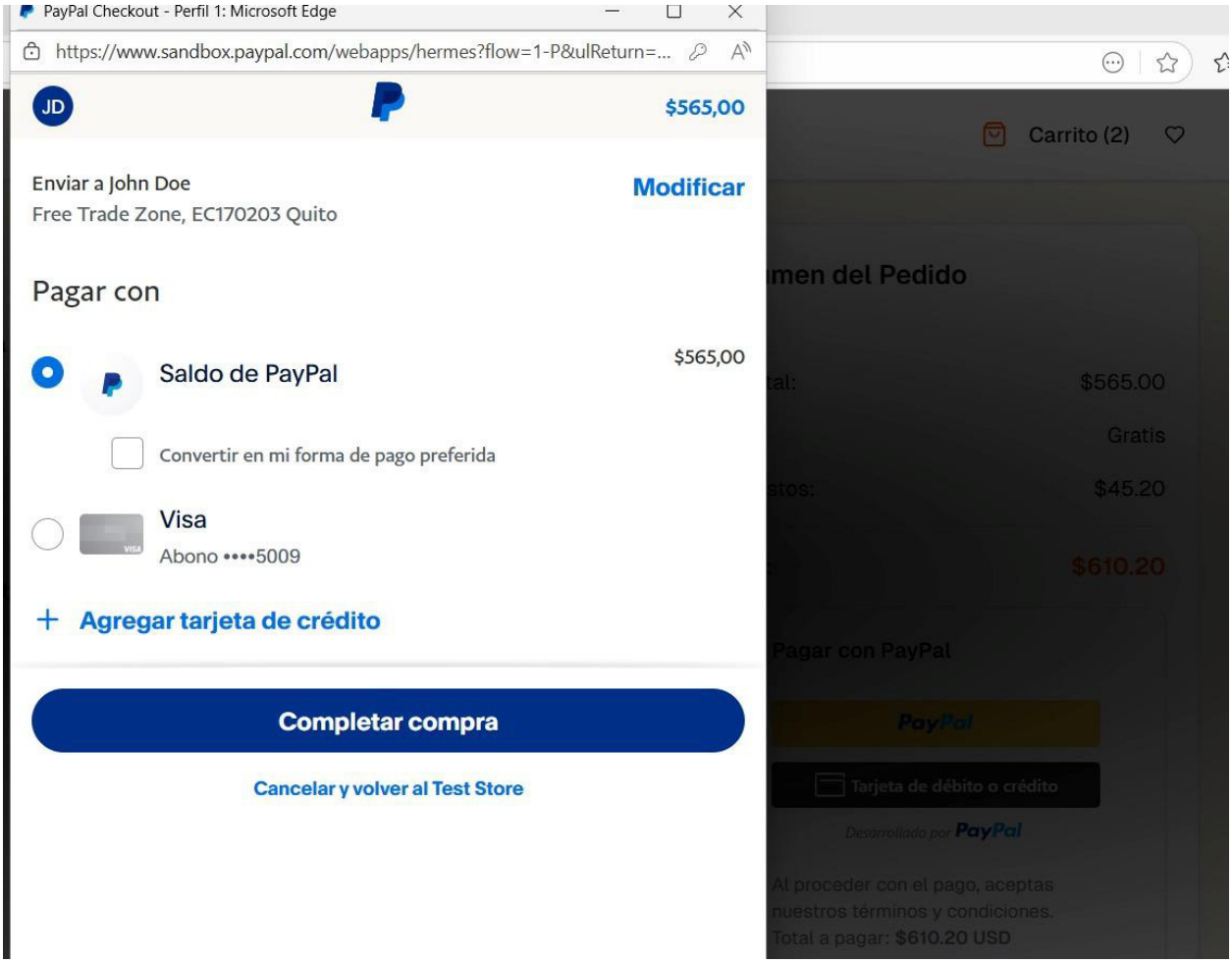
Figura 4 *Visualización del formulario de inicio de sesión.*



Nota 4 *El formulario incluye validaciones del lado del cliente y del servidor para asegurar la integridad de los datos.*

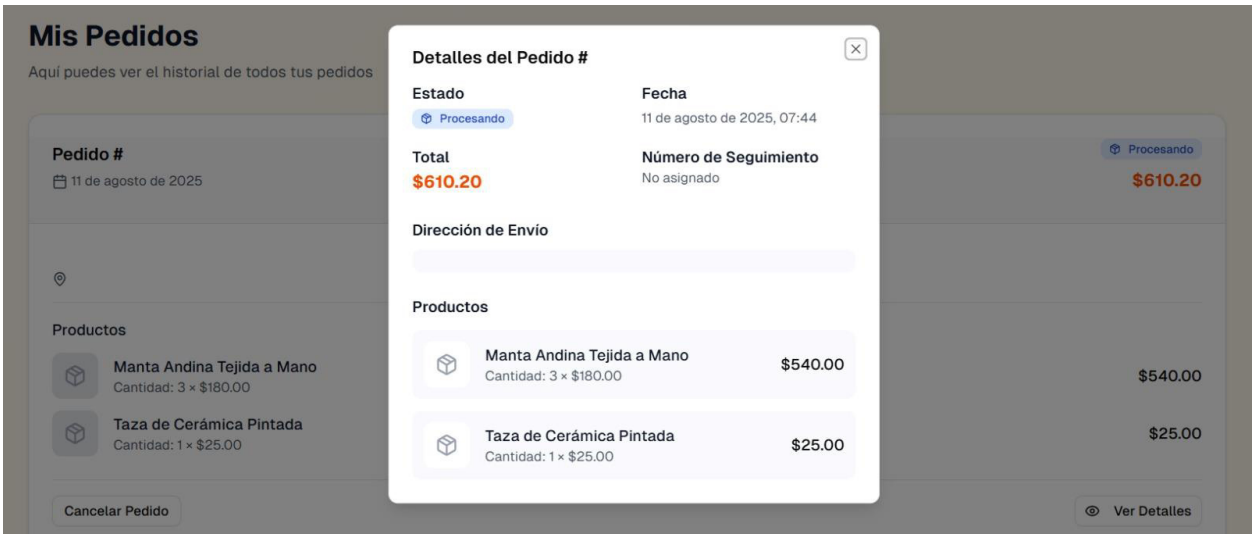
•**Simulación de Pago y Gestión de Pedidos**: Al finalizar la compra, el sistema simula el procesamiento del pago. Se validó que el pedido se almacena correctamente en la base de datos, incluyendo detalles y dirección de envío. Esta lógica reside en la API de pedidos (app/api/orders/route.ts).

Figura 5 *Confirmación de pedido.*



Nota 5 *Esta vista es generada dinámicamente por el frontend tras recibir una respuesta exitosa del backend, asegurando al usuario que su pedido ha sido procesado.*

Figura 6 *Visualización de la gestión de pedidos.*



- **Panel administrativo**: Se desarrolló un dashboard completo y protegido en app/admin/, validando que todas las operaciones CRUD funcionen correctamente.
- **Gestión de Productos**: (/admin/products) Permite añadir, editar y eliminar productos.
- **Configuración del Sistema**: (/admin/settings) Permite ajustar parámetros como la configuración de correo SMTP.

Figura 7 *Visualización del Dashboard.*

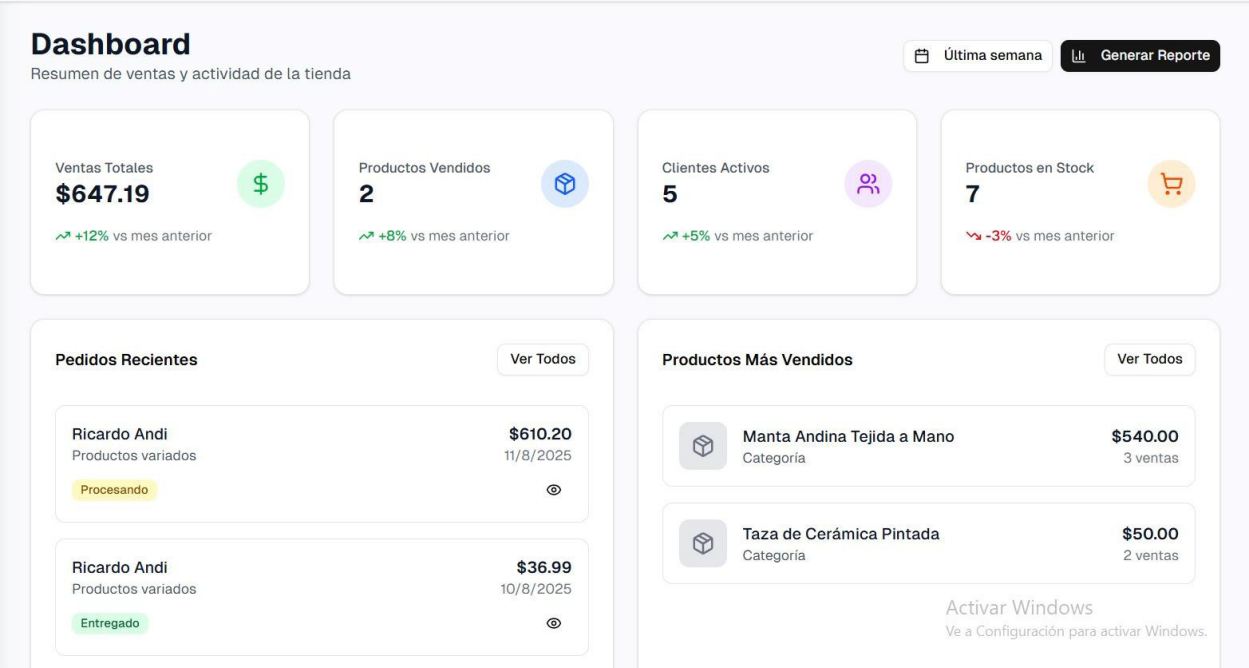
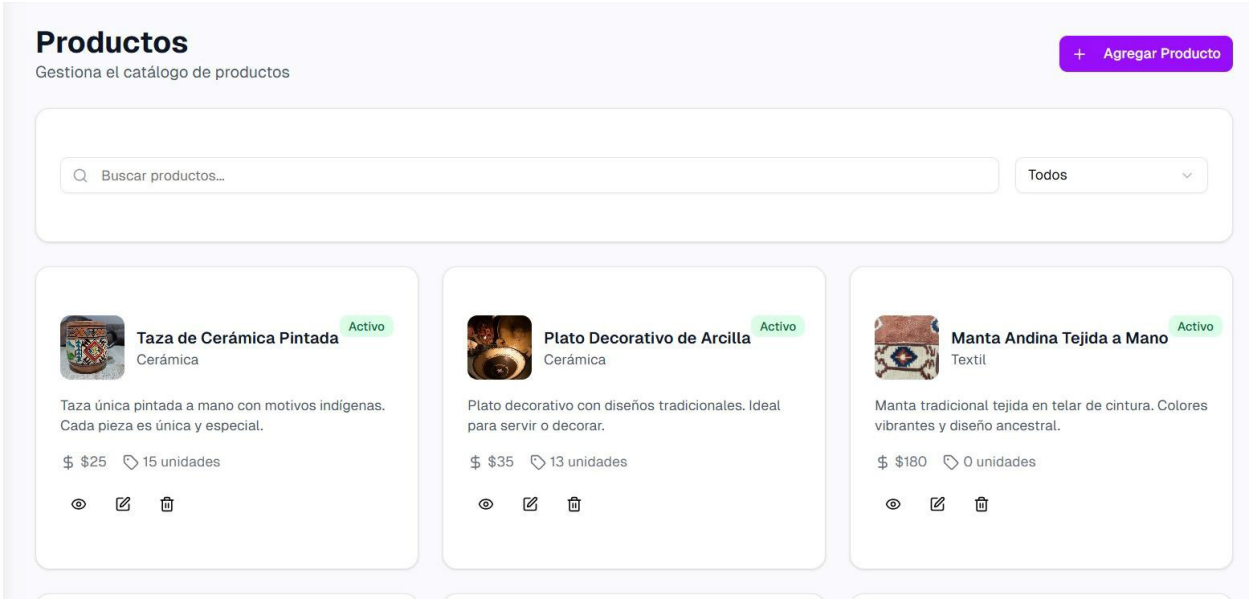
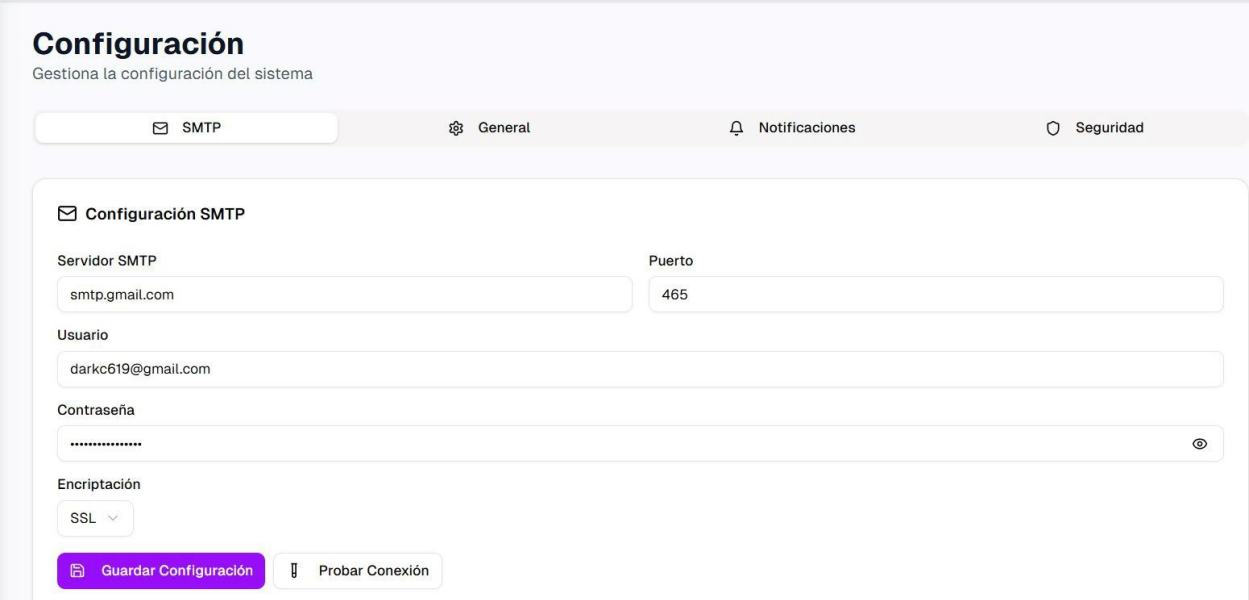


Figura 8 *Panel administrativo para gestión de productos.*



Nota 6 *Esta sección es una ruta protegida accesible solo para administradores. Las operaciones interactúan directamente con la API en app/api/admin/.*

Figura 9 *Visualización de la configuración del sistema SMTP.*



2.1.1 RESULTADOS TÉCNICOS

- **Arquitectura Full-stack con Next.js:** El uso de Next.js permitió unificar el desarrollo, mejorando la eficiencia y el rendimiento general.
- **Tipado Estático con TypeScript:** Garantizó un código más robusto, legible y con menos errores.
- **Gestión de Estado Centralizada:** El uso de React Context API simplificó el manejo de estados globales, haciendo el código más mantenible.
- **Diseño Responsivo y Moderno:** El uso de **Tailwind CSS** y shadcn/ui asegura una experiencia de usuario óptima en cualquier dispositivo.

2.1.2 EVALUACIÓN GENERAL DEL SISTEMA

La evaluación del sistema "Artesian Store" se ha llevado a cabo considerando múltiples criterios de calidad de software, incluyendo funcionalidad, arquitectura, experiencia de usuario, seguridad y rendimiento. El resultado es una plataforma que no solo cumple con los objetivos iniciales del proyecto, sino que los supera al entregar una solución tecnológicamente avanzada, robusta y preparada para un crecimiento futuro:

2.1.3 FUNCIONALIDAD Y CUMPLIMIENTO OBJETIVOS

El sistema es completamente funcional y satisface todos los requisitos definidos. Se ha validado con éxito que:

1. **Proporciona una Tienda Virtual Operativa**: La plataforma permite a los clientes navegar por un catálogo dinámico, ver detalles de productos, registrarse y realizar un ciclo de compra completo, desde la selección de artículos hasta la finalización de un pedido simulado.
2. **Facilita la Gestión Autónoma**: El panel administrativo es una herramienta integral que empodera al artesano, permitiéndole gestionar productos, categorías, inventario, clientes y pedidos sin necesidad de conocimientos técnicos avanzados.
3. **Simulación de Compra Exitosa**: El flujo transaccional, que incluye el carrito de compras persistente y la simulación de pago, funciona de manera correcta, registrando los pedidos en la base de datos y actualizando el stock de productos, lo que valida la lógica de negocio principal.

En resumen, la aplicación trasciende el concepto de un prototipo básico para posicionarse como un Producto Mínimo Viable (MVP) de alta calidad, listo para la siguiente fase de integración con servicios de pago reales.

2.1.4 CALIDAD DE LA ARQUITECTURA Y ESCABILIDAD

La elección de un stack tecnológico moderno es uno de los pilares fundamentales de la calidad del sistema:

- **Arquitectura Full-Stack con Next.js**: El uso de Next.js ha permitido construir una aplicación monolítica, pero con una clara separación de responsabilidades (frontend y backend). Esta arquitectura no solo simplifica el desarrollo y el despliegue, sino que también es inherentemente escalable. La capacidad de Next.js para funcionar en entornos de servidor sin estado (stateless) y su optimización para el despliegue en plataformas como Vercel garantizan que la aplicación puede manejar un crecimiento futuro en tráfico y datos.
- **Base de Datos Relacional Robusta**: La utilización de MySQL con un esquema de base de datos bien diseñado (database_schema.sql) asegura la integridad, consistencia y atomicidad de los datos (principios ACID). Las relaciones entre tablas como users, products y orders están correctamente establecidas, lo que permite realizar consultas complejas de manera eficiente.
- **Código Modular y Mantenible**: La estructuración del código en componentes reutilizables de React (components/), contextos para el estado global (contexts/) y una capa de acceso a datos bien definida (lib/db-utils.ts)

resulta en un código limpio, legible y fácil de mantener, lo cual es crucial para la longevidad y evolución del proyecto.

2.1.5 EXPERIENCIA DE USUARIO (UX) Y USABILIDAD

Se ha puesto un énfasis especial en la creación de una interfaz de usuario que sea tanto estéticamente agradable como funcionalmente intuitiva:

- **Interfaz Moderna e Interactiva**: Gracias a React, la interfaz es altamente dinámica. Acciones como agregar productos al carrito, filtrar el catálogo o recibir notificaciones ocurren de manera instantánea y sin necesidad de recargar la página, proporcionando una experiencia de usuario fluida y moderna.
- **Diseño Responsivo y Consistente**: El uso de Tailwind CSS y el sistema de componentes shadcn/ui garantiza que la aplicación se vea y funcione perfectamente en una amplia gama de dispositivos, desde teléfonos móviles hasta monitores de escritorio. La consistencia visual en todos los módulos (públicos y administrativos) refuerza la profesionalidad de la plataforma.
- **Flujos de Usuario Intuitivos**: Los procesos clave, como el registro, el inicio de sesión y el proceso de compra, han sido diseñados para ser sencillos y directos, minimizando la fricción y guiando al usuario de manera natural a través de la aplicación.

2.1.6 SEGURIDAD Y FIABILIDAD

La seguridad ha sido un aspecto prioritario durante todo el desarrollo:

- **Autenticación Segura**: La implementación de JSON Web Tokens (JWT) para la gestión de sesiones proporciona un mecanismo de autenticación estándar y seguro. Los endpoints de la API están debidamente protegidos, asegurando que solo los usuarios autenticados (y con los roles correctos) puedan acceder a recursos restringidos.
- **Protección de Datos Sensibles**: Todas las contraseñas de los usuarios se almacenan en la base de datos utilizando el algoritmo de hashing bcrypt.js. Esto garantiza que, incluso en el caso de una brecha de seguridad en la base de datos, las contraseñas originales no puedan ser recuperadas.
- **Prevención de Inyección SQL:** El uso de consultas parametrizadas a través de la librería mysql2 mitiga eficazmente el riesgo de ataques de inyección SQL, una de las vulnerabilidades más comunes en aplicaciones web.

En conclusión, la evaluación general del sistema "Artesian Store" es altamente positiva. La plataforma no solo es una herramienta funcional que resuelve la problemática planteada, sino que también representa una pieza de software bien diseñada, segura, escalable y con una experiencia de usuario de alta calidad, demostrando un dominio de las tecnologías y prácticas de desarrollo web modernas.

2.2 OBSERVACIONES Y MEJORAS IDENTIFICADAS

Durante el ciclo de desarrollo y la fase de pruebas funcionales, se realizó un análisis crítico del sistema "Artesian Store". Si bien la plataforma cumple con todos los objetivos planteados y presenta una base tecnológica sólida, se han identificado varias áreas de oportunidad para su optimización y evolución hacia un sistema de nivel de producción. Estas observaciones no representan deficiencias, sino una hoja de ruta estratégica para futuras iteraciones del proyecto, demostrando una comprensión profunda del ciclo de vida del software.

A continuación, se detallan las mejoras identificadas, agrupadas por áreas de impacto:

2.2.1 MEJORAS EN LA EXPERIENCIA DE USUARIO (UX) Y FRONTEND

- **Gestión de Estados de Carga Granulares:**
    - **Observación**: Actualmente, la aplicación muestra indicadores de carga generales para páginas completas (ej. "Cargando productos..."). En operaciones específicas, como agregar un producto al carrito, la retroalimentación es instantánea a través de notificaciones, pero el estado del botón que inicia la acción no cambia visualmente para indicar que una operación está en curso.
    - **Mejora Propuesta**: Implementar estados de carga a nivel de componente. Por ejemplo, al hacer clic en el botón "Agregar al Carrito", este podría mostrar un ícono de carga (spinner) y desactivarse temporalmente hasta que la operación se complete en

el servidor. Esto proporcionaría una retroalimentación visual más directa e inmediata al usuario, mejorando la percepción de interactividad y evitando clics duplicados.

- **Optimización de imágenes para la web:**
    - **Observación**: Las imágenes de los productos, aunque se cargan dinámicamente, no están optimizadas para diferentes tamaños de pantalla y resoluciones, lo que puede afectar los tiempos de carga en dispositivos con conexiones lentas.
    - **Mejora Propuesta**: Implementar srcset y tamaños de imagen responsivos utilizando las capacidades avanzadas del componente <Image> de Next.js. Adicionalmente, se podría integrar un servicio de optimización de imágenes en tiempo real o utilizar formatos de imagen modernos como WebP o AVIF, que ofrecen una compresión superior sin una pérdida significativa de calidad. Esto reduciría drásticamente el peso de las páginas y mejoraría la métrica de Largest Contentful Paint (LCP), crucial para el SEO.

2.2.2 MEJORAS EN RENDIMIENTO Y ESCALABILIDAD

- **Implementación de una Estrategia de Renderizado Híbrida:**
    - **Observación**: La aplicación funciona principalmente como una Single Page Application (SPA) con renderizado del lado del cliente (CSR), lo cual es ideal para interfaces altamente interactivas como el panel de administración.

- **Mejora Propuesta**: Aprovechar las poderosas estrategias de renderizado híbrido de Next.js para optimizar el rendimiento y el SEO de las páginas públicas:
- **Server-Side Rendering (SSR)**: Aplicar SSR a las páginas de detalle de productos (/product/[id]) para que el contenido se renderice en el servidor en cada solicitud. Esto garantiza que los motores de búsqueda siempre reciban el contenido actualizado y mejora el tiempo de carga inicial para el usuario.
- **Static Site Generation (SSG)**: Utilizar SSG para páginas cuyo contenido no cambia frecuentemente, como la página de inicio (/), la página de contacto (/contact) o el listado principal de productos (/products). Estas páginas se generarían en tiempo de compilación, permitiendo que se sirvan instantáneamente desde una red de distribución de contenidos (CDN), ofreciendo la máxima velocidad posible.

- **Gestión Escalable de Activos Digitales:**
  - **Observación**: El sistema actual sube y almacena las imágenes de los productos en el sistema de archivos local del servidor, dentro de la carpeta public/uploads/products/. Este enfoque, aunque funcional para el desarrollo, presenta serios problemas de escalabilidad en un entorno de producción, especialmente en despliegues sin estado (stateless) o con balanceo de carga.

- **Mejora Propuesta**: Integrar un servicio de almacenamiento en la nube, como Amazon S3, Google Cloud Storage o Cloudinary. Esto desacoplaría el almacenamiento de imágenes del servidor de la aplicación, mejoraría la velocidad de entrega a través de CDNs globales, facilitaría la escalabilidad horizontal del sistema y reduciría la carga del servidor principal.

2.2.3 MEJORAS EN LA SEGURIDAD DE LA APLICACIÓN

- **Protección contra Ataques CSRF (Cross-Site Request Forgery):**
  - **Observación**: Los formularios de la aplicación, especialmente aquellos que realizan acciones de modificación de estado (como actualizar el perfil o añadir un producto en el panel de administración), no implementan actualmente una protección contra ataques CSRF.
  - **Mejora Propuesta**: Implementar una estrategia de tokens anti-CSRF. Al renderizar un formulario, el servidor generaría un token único que se enviaría al cliente. Este token sería incluido en la solicitud posterior y verificado por el servidor, asegurando que la solicitud proviene legítimamente de la aplicación y no de un sitio malicioso de terceros.
- **Implementación de Políticas de Seguridad de Contenido (CSP):**
  - **Observación**: La aplicación no define una Política de Seguridad de Contenido (Content Security Policy) estricta, lo que la deja

vulnerable a ciertos tipos de ataques, como el Cross-Site Scripting (XSS).

- **Mejora Propuesta**: Configurar cabeceras de CSP en la configuración de Next.js (next.config.mjs). Esto permitiría definir explícitamente qué fuentes de contenido (scripts, estilos, imágenes, etc.) son confiables y pueden ser cargadas por el navegador, bloqueando la ejecución de cualquier recurso proveniente de fuentes no autorizadas.

2.2.4 MEJORAS EN LA CALIDAD DEL CODIGO Y MANTENIBILIDAD

- Implementación de Pruebas Automatizadas:
  - **Observación**: El proyecto carece de una suite de pruebas automatizadas. Las validaciones se han realizado de forma manual, lo cual es propenso a errores y difícil de escalar a medida que el proyecto crece.
  - **Mejora Propuesta**: Introducir un framework de pruebas como Jest junto con React Testing Library para desarrollar pruebas unitarias (para componentes individuales y funciones de utilidad) y pruebas de integración (para verificar la interacción entre múltiples componentes). Adicionalmente, se podrían implementar pruebas End-to-End (E2E) con herramientas como Cypress o Playwright para validar los flujos críticos de usuario de principio a fin, como el proceso de registro y compra.

- **Adopción de un ORM (Object-Relational Mapping):**
    - **Observación:** La capa de acceso a datos (lib/db-utils.ts) utiliza consultas SQL directas. Aunque son eficientes, pueden volverse difíciles de mantener y son propensas a errores a medida que la complejidad de las consultas aumenta.
    - **Mejora Propuesta**: Integrar un ORM moderno como Prisma o TypeORM. Esto proporcionaría una capa de abstracción sobre la base de datos, permitiendo escribir consultas de forma segura y tipada en TypeScript. Además, facilitaría enormemente la gestión de las migraciones del esquema de la base de datos, haciendo que las futuras actualizaciones sean más seguras y predecibles.

3 PROPUESTA

A partir del análisis de resultados y las observaciones identificadas durante el desarrollo, se propone una serie de mejoras estratégicas orientadas a fortalecer la funcionalidad, seguridad y escalabilidad del sistema "Artesian Store". Estas propuestas no solo buscan optimizar la plataforma actual, sino también prepararla para una transición exitosa hacia un entorno de producción real, convirtiéndola en una solución de comercio electrónico completa y competitiva.

3.1.1 INTEGRACIÓN DE UNA PASARELA DE PAGO REAL SIMULADO.

Justificación: La funcionalidad más crítica para transformar la plataforma de un prototipo a un negocio operativo es la capacidad de procesar transacciones

monetarias reales. La simulación de pago actual valida la lógica de negocio, pero la integración de una pasarela de pago real es indispensable para la comercialización.

Propuesta Técnica:

1. **Selección del Proveedor**: Se recomienda integrar Stripe debido a su robusta documentación, su excelente soporte para desarrolladores y sus librerías optimizadas para React y Next.js.

2. **Implementación en el Backend:**

   - Crear un nuevo endpoint en la API, por ejemplo, POST /api/payments/create-checkout-session.
   - Este endpoint recibirá los productos del carrito del usuario y utilizará el SDK de Stripe para crear una sesión de pago segura.
   - Tras una transacción exitosa, se configurará un Webhook (POST /api/payments/webhook) que escuchará los eventos de Stripe (ej. checkout.session.completed). Este webhook será el responsable de actualizar el estado del pedido en la base de datos de "pendiente" a "pagado", garantizando la fiabilidad del proceso incluso si el usuario cierra el navegador después de pagar.

3. **Implementación en el Frontemd:**

   - Al hacer clic en "Proceder al Pago", el cliente realizará una solicitud al nuevo endpoint para obtener la URL de la sesión de pago de Stripe.
   - Se redirigirá al usuario a la página de checkout segura de Stripe para completar la transacción.

3.1.2 SISTEMA DE NOTIFICACIONES TRANSACIONALES POR EMAIL

Justificación: Una comunicación clara y automatizada es fundamental para generar confianza en el cliente y reducir la carga operativa del administrador. El sistema ya cuenta con una base sólida para el envío de correos a través de la configuración SMTP (app/admin/settings/page.tsx).

Propuesta Técnica:

- **Crear un Módulo de Correo:** Desarrollar un servicio de notificaciones en lib/email-service.ts que utilice la librería Nodemailer (ya presente en el proyecto) para enviar correos electrónicos utilizando la configuración SMTP guardada en la base de datos.
- **Definir Plantillas de Correo:** Crear plantillas HTML dinámicas para diferentes eventos transaccionales, tales como:
    - **Confirmación de Registro:** Un correo de bienvenida para los nuevos usuarios.
    - **Confirmación de Pedido:** Un resumen detallado del pedido enviado al cliente tras una compra exitosa.
    - **Notificación de Nuevo Pedido:** Un aviso enviado al correo del administrador cada vez que se realice una nueva venta.
    - **Actualización de Envío:** Notificaciones sobre cambios en el estado del pedido (ej. "enviado", "en tránsito").
- **Integración con la Lógica de Negocio:** Invocar el servicio de correo en los puntos clave de la aplicación:

- Después de un registro exitoso en app/api/auth/register/route.ts.
- Después de procesar un pedido correctamente en el Webhook de la pasarela de pago.
- Cuando un administrador actualice el estado de un pedido desde el panel.

3.1.3 MEJORAS AVANZADAS EN LA SEGURIDAD DEL SISTEMA

Justificación: Para un entorno de producción, es imperativo reforzar la seguridad de la aplicación para proteger tanto los datos de los usuarios como la integridad del negocio.

Propuesta Técnica:

- **Protección contra Ataques CSRF (Cross-Site Request Forgery):**
  - Implementar una librería como csurf o una solución personalizada en un middleware de Next.js.
  - Esta capa de seguridad generará y validará tokens anti-CSRF en todos los formularios que realicen mutaciones de datos (ej. actualizar perfil, crear producto), asegurando que las solicitudes provengan únicamente de la propia aplicación.
- **Implementación de Cabeceras de Seguridad:**
  - Configurar cabeceras de seguridad HTTP en next.config.mjs para una protección adicional a nivel de navegador. Esto incluye:

- **Content-Security-Policy (CSP):** Para prevenir ataques de Cross-Site Scripting (XSS) al restringir las fuentes desde las cuales se puede cargar contenido.
- **Strict-Transport-Security (HSTS):** Para forzar el uso de HTTPS en todas las comunicaciones.
- **X-Content-Type-Options:** Para prevenir que los navegadores interpreten archivos con un tipo MIME incorrecto.

- **RATE LIMITING EN LA API:**
  - Integrar un middleware en los endpoints más sensibles (ej. login, register) para limitar el número de solicitudes que un cliente puede realizar en un período de tiempo determinado. Esto es crucial para mitigar ataques de fuerza bruta y de denegación de servicio (DoS).

3.1.4 ADOPTACIÓN DE UN ORM PARA LA GESTIÓN DE DATOS

Justificación: Aunque las consultas SQL directas (lib/db-utils.ts) son funcionales, un ORM (Object-Relational Mapper) como Prisma puede mejorar drásticamente la seguridad, la mantenibilidad y la experiencia del desarrollador.

Propuesta Técnica:

- **Migración a Prisma**:
  - Instalar y configurar Prisma en el proyecto.

    - Definir el esquema de la base de datos en el archivo schema.prisma, que servirá como la única fuente de verdad para la estructura de los datos.
    - Utilizar el cliente de Prisma, que es totalmente tipado, para reemplazar todas las consultas SQL directas. Esto eliminaría por completo el riesgo de inyección SQL y proporcionaría autocompletado y validación de tipos en el editor de código.
- **Gestión de Migraciones**: Aprovechar el sistema de migraciones de Prisma para gestionar los cambios en el esquema de la base de datos de una manera controlada y versionada, lo cual es fundamental para el trabajo en equipo y los despliegues a producción.

4 CONCLUSIONES

Se ha desarrollado con éxito una plataforma de comercio electrónico completa y funcional, superando significativamente los objetivos iniciales del proyecto. La transición de un prototipo basado en tecnologías tradicionales a una aplicación full-stack construida con un stack moderno (Next.js, React, TypeScript y MySQL) ha resultado en un producto final de calidad superior. El sistema no es meramente una "tienda virtual básica", sino un Producto Mínimo Viable (MVP) robusto, que demuestra la capacidad de aplicar metodologías y herramientas de vanguardia para resolver problemas del mundo real.

La arquitectura del sistema, basada en Next.js y sus API Routes, ha demostrado ser una elección acertada para garantizar la escalabilidad, el rendimiento y la mantenibilidad de la aplicación. La clara separación entre la lógica del frontend (gestionada por React y sus componentes reutilizables) y la del backend (manejada por los endpoints de la API) crea una base de código limpia y modular. Esta arquitectura no solo facilita futuras actualizaciones y la adición de nuevas funcionalidades, sino que también está inherentemente optimizada para el despliegue en infraestructuras de nube modernas.

La implementación de un sistema de autenticación seguro basado en JSON Web Tokens (JWT) y el hashing de contraseñas con bcrypt.js asegura la integridad y confidencialidad de los datos de los usuarios. Se ha validado que el sistema protege eficazmente las rutas y los recursos, diferenciando los roles de cliente y administrador. Esto es un pilar fundamental que otorga fiabilidad a la

plataforma, un requisito indispensable para cualquier aplicación de comercio electrónico que aspire a gestionar transacciones y datos personales.

El panel de administración integral representa una herramienta de empoderamiento tangible para los pequeños artesanos. A través de una interfaz intuitiva y amigable (app/admin/), los usuarios administradores tienen control total sobre los aspectos clave de su negocio en línea: gestión de inventario, visualización de pedidos, administración de clientes y configuración del sistema. Esto cumple con el objetivo central de proporcionar una solución autónoma que reduce la dependencia de intermediarios técnicos y facilita la transición del artesano al comercio digital.

La experiencia de usuario (UX) ha sido un factor prioritario y exitosamente logrado en el proyecto. Gracias a la interactividad de React, la gestión de estado centralizada con Context API y el diseño responsivo implementado con Tailwind CSS, la plataforma ofrece una navegación fluida, rápida y coherente en cualquier dispositivo. Acciones como agregar productos al carrito, filtrar el catálogo o recibir notificaciones en tiempo real se ejecutan sin interrupciones, elevando la calidad percibida y la usabilidad del sistema tanto para el cliente final como para el administrador.

El proyecto ha validado de manera práctica la viabilidad de construir soluciones de comercio electrónico de nivel profesional utilizando herramientas de código abierto. Al combinar la potencia de un framework como Next.js con la fiabilidad de una base de datos como MySQL, se ha demostrado que es posible crear

plataformas competitivas sin incurrir en altos costos de licencias, lo cual es un factor clave para la sostenibilidad de proyectos dirigidos a pequeños emprendedores.

5 REFERENCIAS BIBLIOGRÁFICA

Bootstrap. (2024). Documentación oficial. https://getbootstrap.com

GitHub Docs. (2024). Uso de repositorios para control de versiones. https://docs.github.com

ISO. (2020). Norma ISO 9241 sobre experiencia de usuario y ergonomía en la web.

JavaScript.info. (2024). Guía completa de JavaScript moderno. https://es.javascript.info/

MongoDB Manual. (2024). Bases de datos NoSQL para e commerce. https://www.mongodb.com/docs/

Node.js Docs. (2024). Documentación oficial. https://nodejs.org/es/docs/

PayPal Developer. (2023). Documentación para simulación de pagos. https://developer.paypal.com/

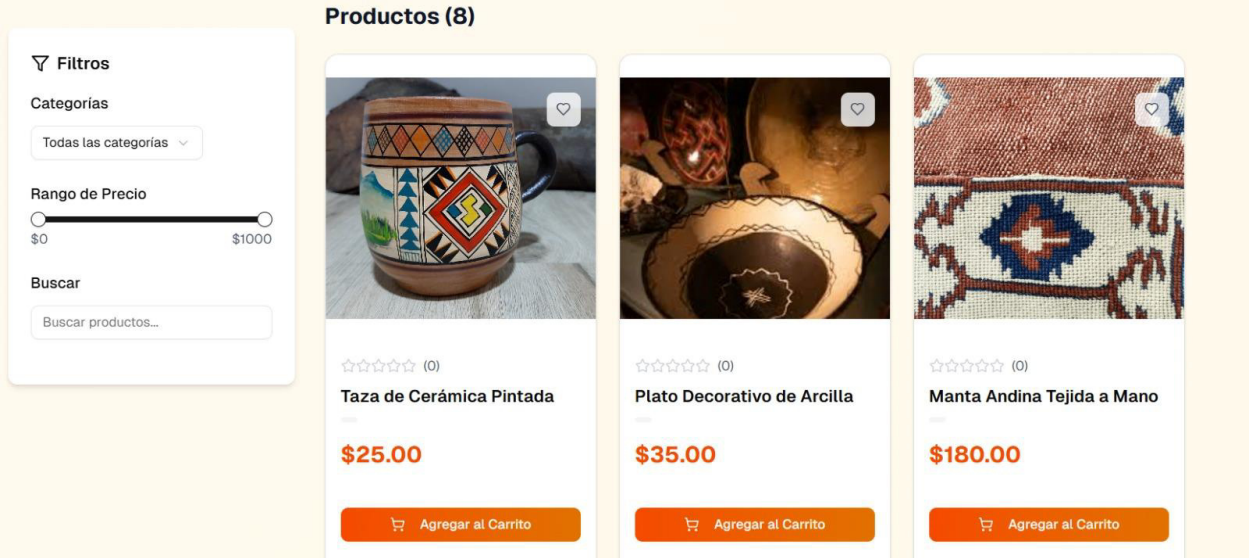
Ruiz, M. (2022). Diseño de tiendas virtuales con tecnologías web. Revista Digital de Comercio Electrónico, 11(2), 23 35.

Torres, A. (2021). PHP y MySQL en proyectos e-commerce. Alfaomega.

W3C. (2023). Estándares para formularios accesibles y seguros. https://www.w3.org/

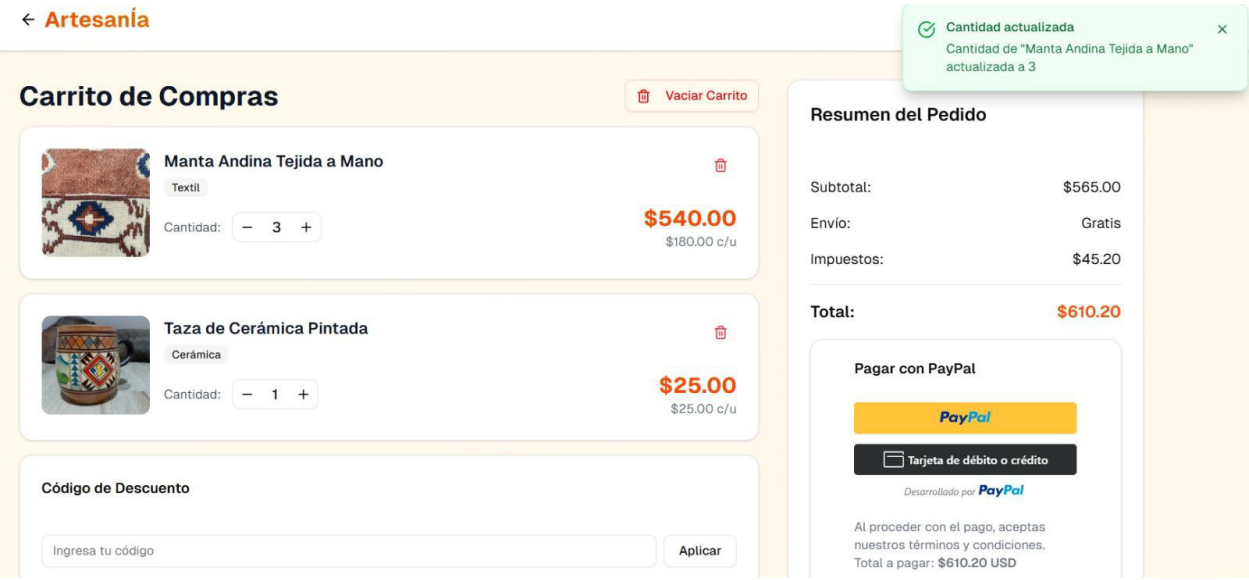
6 ANEXOS

Anexo 1. *Vista del catálogo de productos.*



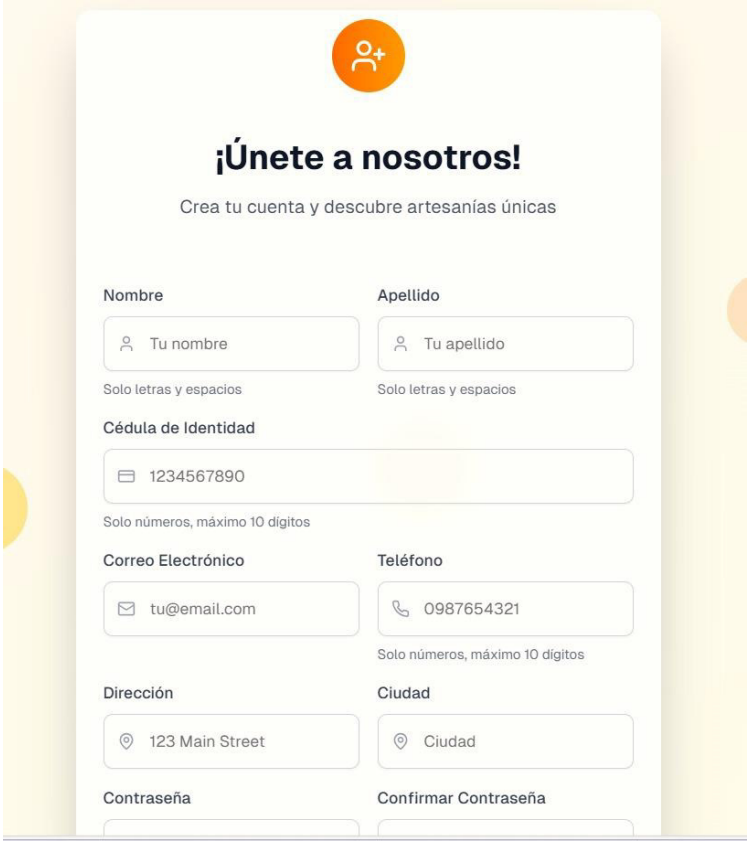
Nota 1: *La interfaz está construida con React y estilizada con Tailwind CSS. Los datos de los productos se cargan dinámicamente desde la base de datos a través de una API REST.*

Anexo 2. *Funcionalidad del carrito de compra.*



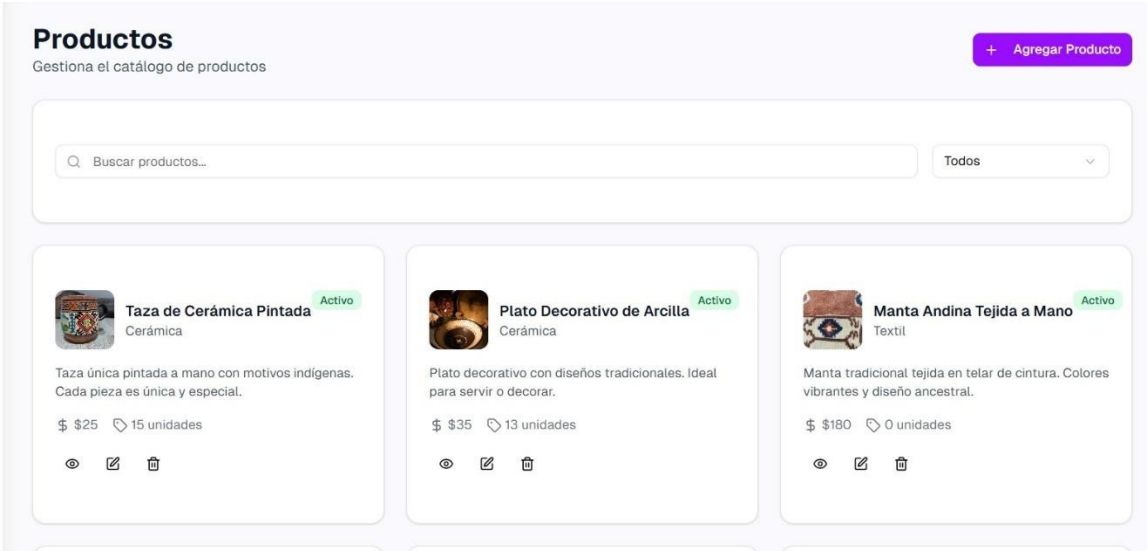
Nota 2: *El estado del carrito se gestiona globalmente mediante el CartContext de React, permitiendo que el contador en el header se actualice en tiempo real.*

Anexo 3. *Formulario de registro de nuevos usuarios.*



Nota 3: *El formulario incluye validaciones del lado del cliente y del servidor. Las contraseñas se almacenan hasheadas utilizando bcrypt.js.*

Anexo 4. *Panel administrativo para gestión de productos.*



Nota 4: *Esta sección es una ruta protegida accesible solo para administradores. Las operaciones CRUD interactúan directamente con la API en app/api/admin/products.*

Anexo 5. *Lógica de Autenticación en API Route (app/api/auth/login/route.ts).*

```
// Fragmento que muestra la validación de credenciales y generación de JWT
import { getUserByEmail } from '@/lib/db-utils';
import bcrypt from 'bcryptjs';
import jwt from 'jsonwebtoken';
// ... dentro de la función POST
const user = await getUserByEmail(email);
if (!user) {
  return NextResponse.json({ error: 'Credenciales inválidas' }, { status: 401 });
}
```

```
const isPasswordValid = await bcrypt.compare(password, user.password); if (!isPasswordValid) { return NextResponse.json({ error: 'Credenciales inválidas' }, { status: 401 }); } const token = jwt.sign( { userId: user.id, email: user.email, role: user.role }, process.env.JWT_SECRET!, { expiresIn: '7d' } ); // ...
```

Nota 5: *Este fragmento ilustra la implementación de la autenticación segura. Se obtiene el usuario por su email, se compara la contraseña proporcionada con el hash almacenado usando bcrypt, y finalmente se genera un JSON Web Token para gestionar la sesión del usuario.*

Anexo 6. *Gestión de Estado Global con React Context (contexts/CartContext.tsx)*

```
// Fragmento que muestra cómo se gestiona el estado del carrito import React, { createContext, useContext, useState, useEffect } from 'react'; // ... definición de interfaces const CartContext = createContext<CartContextType | undefined>(undefined); export function CartProvider({ children }) { const [cartItems, setCartItems] = useState<CartItem[]>([]); const { user, isAuthenticated } = useAuth(); const loadCart = async () => { if (!isAuthenticated || !user) return; try { const response = await fetch(`/api/cart?userId=${user.id}`); // ... lógica para actualizar el estado } catch (error) { console.error('Error cargando carrito:', error); } };
```

```
useEffect(() => { loadCart(); }, [user, isAuthenticated]); // ... otras funciones como addToCart, removeFromCart return ( <CartContext.Provider value={{ cartItems, ... }}> {children} </CartContext.Provider> ); }
```

Nota 6: *Este código muestra el uso de React Context para crear un proveedor de estado (CartProvider). La función loadCart se ejecuta cada vez que el estado de autenticación cambia, asegurando que el carrito del usuario se sincronice automáticamente con los datos del backend. Esto demuestra una gestión de estado moderna y eficiente en el frontend.*