Evidencia GA6-220501123-AA4-EV01

Proyecto Final

Aprendiz: Juan Carlos Lopez Moreno

Instructor: Alvaro Esteban Betancourt Matoma

Área Técnica

Servicio Nacional de Aprendizaje – SENA

Técnico en Programación de Aplicaciones y Servicios Para la Nube

Código: 3186263

Diciembre 2025

Link del repositorio con el código fuente: https://github.com/juancholopes/cafe-yamara

Link del proyecto desplegado: https://cafe-yamara.vercel.app/

1. Introducción

El sector cafetero colombiano representa uno de los pilares fundamentales de la economía nacional, constituyendo una fuente de ingresos para más de 540,000 familias productoras distribuidas en 23 departamentos del país. Colombia ocupa el tercer lugar a nivel mundial en la producción de café arábica, reconocido internacionalmente por sus características organolépticas excepcionales derivadas de condiciones geográficas privilegiadas que incluyen altitudes entre 1,200 y 2,000 metros sobre el nivel del mar, temperaturas promedias entre 17 y 23 grados centígrados, y precipitaciones bien distribuidas durante el año. El departamento de Santander, específicamente, aporta aproximadamente el 7% de la producción nacional y se ha consolidado como una región productora de cafés especiales con denominaciones de origen que garantizan estándares de calidad superiores.

En este contexto, Finca Miraflores emerge como un emprendimiento familiar ubicado en el municipio de Guadalupe, Santander, dedicado a la producción de café arábica orgánico bajo el modelo de agricultura sostenible y comercialización bajo la marca Café Yamara. La finca implementa prácticas agroecológicas que prescinden de insumos químicos sintéticos, promoviendo la biodiversidad y la conservación del ecosistema montañoso santandereano. Sin embargo, a pesar de la alta calidad del producto y el compromiso ambiental demostrado, la operación enfrenta limitaciones significativas en materia de comercialización directa debido a la ausencia de canales digitales que permitan conectar eficientemente con consumidores finales interesados en productos premium con trazabilidad garantizada.

La transformación digital del sector agroalimentario ha demostrado ser un catalizador fundamental para mejorar la competitividad de pequeños y medianos productores. Las plataformas de comercio electrónico especializadas no solo eliminan intermediarios innecesarios que erosionan los márgenes de rentabilidad, sino que también facilitan la construcción de relaciones directas con consumidores cada vez más conscientes del origen, proceso productivo y prácticas sostenibles asociadas a los productos que adquieren. En este sentido, el desarrollo de una solución tecnológica integral que combine aspectos institucionales, educativos y transaccionales representa una oportunidad estratégica para fortalecer la cadena de valor de Café Yamara.

El presente proyecto aborda esta necesidad mediante el diseño, desarrollo e implementación de una plataforma web completa que integra funcionalidades de escaparate digital, sistema de comercio electrónico y panel de administración empresarial. La solución propuesta se fundamenta en tecnologías modernas de desarrollo web que priorizan el rendimiento, la experiencia de usuario, la seguridad de las transacciones y la escalabilidad del sistema ante crecimientos futuros. La arquitectura seleccionada responde a principios de ingeniería de software contemporáneos que enfatizan la separación de responsabilidades, la reutilización de componentes y la mantenibilidad del código a largo plazo.

La metodología de desarrollo adoptada combina prácticas ágiles con enfoque iterativo e incremental, permitiendo entregas funcionales tempranas y ajustes continuos basados en retroalimentación de usuarios reales. Este enfoque resulta particularmente apropiado para emprendimientos donde los requerimientos pueden evolucionar conforme se comprende mejor el comportamiento del mercado objetivo y las capacidades operativas de la organización. La

documentación técnica exhaustiva garantiza que futuras ampliaciones o modificaciones puedan ser abordadas eficientemente por desarrolladores que se integren al proyecto.

El documento se estructura en secciones que abordan sistemáticamente el análisis del problema, la fundamentación teórica de las decisiones tecnológicas, la descripción detallada de la arquitectura implementada, los procesos de prueba y validación realizados, y finalmente los resultados obtenidos junto con conclusiones y recomendaciones para evoluciones futuras del sistema. Los anexos complementan el cuerpo principal con diagramas técnicos, fragmentos de código relevantes, capturas de pantalla de la interfaz y enlaces a los repositorios de código fuente y aplicación desplegada.

---

2. Planteamiento del Problema

La comercialización de café en Colombia opera tradicionalmente mediante cadenas de intermediación que involucran cooperativas de productores, empresas procesadoras, distribuidores mayoristas y minoristas antes de alcanzar al consumidor final. Este modelo de múltiples eslabones genera ineficiencias económicas que afectan desproporcionadamente a los pequeños productores, quienes frecuentemente reciben apenas entre el 10% y el 15% del precio final de venta al público por su café tostado. Para Finca Miraflores, productora de café orgánico premium con certificaciones de calidad que justificarían precios superiores en el mercado, esta estructura representa una barrera significativa para capturar el valor agregado de sus prácticas diferenciadas.

El problema se ve agravado por la creciente demanda de consumidores urbanos educados que buscan activamente productos con historias auténticas, transparencia en el origen y garantías de

sostenibilidad ambiental y social. Estudios recientes del mercado de cafés especiales demuestran que segmentos importantes de consumidores están dispuestos a pagar sobreprecio de hasta 40% respecto a café commodity cuando se les garantiza trazabilidad completa y se les comunica efectivamente la propuesta de valor diferencial. Sin embargo, los canales tradicionales de distribución no facilitan esta comunicación directa entre productor y consumidor, resultando en una desconexión que impide la materialización de oportunidades comerciales mutuamente beneficiosas.

Desde la perspectiva del productor, la ausencia de presencia digital representa una desventaja competitiva crítica en un mercado donde la visibilidad online se ha convertido en requisito fundamental para la viabilidad comercial. Finca Miraflores carece de mecanismos para comunicar su historia familiar, explicar su proceso productivo artesanal, mostrar su ubicación geográfica privilegiada en las montañas santandereanas, o comercializar sus productos directamente a través de canales digitales. Esta situación limita drásticamente el alcance geográfico de sus operaciones al entorno local inmediato, desaprovechando mercados potenciales en ciudades principales donde la concentración de consumidores objetivo es significativamente mayor.

Adicionalmente, la gestión operativa de la finca presenta ineficiencias derivadas de sistemas manuales o semi-automatizados para control de inventarios, seguimiento de pedidos y administración de información de clientes. La ausencia de un sistema integrado genera riesgos de errores humanos, duplicación de esfuerzos, dificultades para generar reportes de desempeño y limitaciones para tomar decisiones estratégicas basadas en datos. Conforme crece el volumen de operaciones, estas deficiencias se amplifican exponencialmente, amenazando la capacidad de escalar el negocio sosteniblemente.

El contexto competitivo añade presión adicional dado que otras fincas cafeteras y marcas comercializadoras han comenzado a desarrollar presencias digitales sofisticadas, implementando estrategias de contenido, comercio electrónico y marketing digital que les permiten capturar participación de mercado aceleradamente. La demora en adoptar estas capacidades digitales coloca a Café Yamara en desventaja frente a competidores menos diferenciados en términos de calidad del producto pero más avanzados en infraestructura digital.

Desde el punto de vista técnico, cualquier solución debe considerar restricciones presupuestarias propias de un emprendimiento familiar, lo que descarta alternativas tecnológicas costosas que requieran licencias empresariales o infraestructuras complejas. Simultáneamente, la solución debe cumplir estándares profesionales de seguridad para proteger información sensible de clientes y transacciones financieras, así como garantizar disponibilidad y rendimiento adecuados para soportar tráfico variable sin degradación de la experiencia de usuario. La tensión entre limitaciones de recursos y requerimientos técnicos exigentes constituye un desafío de diseño que debe ser resuelto mediante selección inteligente de tecnologías y arquitecturas.

Finalmente, existe una dimensión de sostenibilidad de largo plazo que debe ser considerada. La solución tecnológica no puede depender de un único desarrollador individual cuyo conocimiento especializado se convierta en punto único de falla para el mantenimiento y evolución del sistema. La arquitectura debe facilitar que otros desarrolladores puedan comprender la estructura del código, realizar modificaciones seguras y agregar nuevas funcionalidades conforme evolucionan las necesidades del negocio. Esto implica adherencia a convenciones establecidas de la industria, documentación exhaustiva y selección de tecnologías con comunidades activas y abundante material de referencia.

3. Objetivos

3.1 Objetivo General

Desarrollar e implementar una plataforma web integral que permita a Finca Miraflores comercializar directamente café orgánico premium bajo la marca Café Yamara, integrando funcionalidades institucionales, transaccionales y administrativas mediante una arquitectura escalable, segura y mantenible que utilice tecnologías modernas de desarrollo web y mejores prácticas de ingeniería de software.

3.2 Objetivos Específicos

El primer objetivo específico consiste en diseñar y construir un módulo institucional que comunique efectivamente la propuesta de valor diferencial de Café Yamara mediante contenido narrativo sobre la historia de Finca Miraflores, visualización detallada del proceso productivo desde el cultivo hasta el empaque, y presentación de la ubicación geográfica a través de mapas interactivos que permitan a los usuarios potenciales comprender el contexto montañoso santandereano donde se produce el café. Este módulo debe cumplir estándares de diseño moderno, ser completamente responsivo para adaptarse a dispositivos móviles y de escritorio, y optimizarse para motores de búsqueda mediante implementación correcta de técnicas de SEO que incluyan renderizado del lado del servidor, metadatos apropiados y estructura semántica del contenido.

El segundo objetivo específico implica implementar un sistema de comercio electrónico completo que incluya catálogo de productos con información detallada de precios, presentaciones disponibles, descripciones y fotografías de alta calidad, funcionalidad de carrito

de compras con persistencia de estado mediante almacenamiento local del navegador, integración con pasarela de pagos segura que procese transacciones con tarjetas de crédito y débito cumpliendo estándares PCI DSS sin exponer información sensible al servidor de la aplicación, y generación automática de órdenes de compra que registren todos los detalles transaccionales para seguimiento posterior. El sistema debe manejar apropiadamente casos edge como productos agotados, aplicación de descuentos promocionales y cálculos correctos de totales incluyendo impuestos y costos de envío cuando apliquen.

El tercer objetivo específico se enfoca en desarrollar un panel administrativo protegido mediante autenticación robusta que permita a usuarios autorizados realizar operaciones CRUD completas sobre el catálogo de productos, incluyendo carga de imágenes optimizadas automáticamente para web, gestión de inventarios con alertas de stock bajo, visualización y actualización de estados de órdenes recibidas, y acceso a métricas clave de desempeño del negocio mediante dashboards que presenten información consolidada sobre ventas, productos más demandados y comportamiento temporal de las transacciones. La interfaz administrativa debe priorizar eficiencia operativa mediante flujos de trabajo intuitivos que minimicen la cantidad de clics necesarios para completar tareas frecuentes.

El cuarto objetivo específico aborda la implementación de una arquitectura de software que separe claramente responsabilidades mediante capas bien definidas, incluyendo capa de presentación con componentes reutilizables de interfaz de usuario, capa de lógica de negocio encapsulada en servicios especializados, capa de acceso a datos que abstraiga interacciones con la base de datos y servicios externos, y capa de infraestructura que maneje aspectos transversales como autenticación, autorización, manejo de errores y logging. Esta arquitectura debe

documentarse exhaustivamente mediante diagramas UML, descripciones textuales de patrones aplicados y comentarios inline en el código que expliquen decisiones de diseño no obvias.

El quinto objetivo específico consiste en establecer un pipeline de integración y despliegue continuo que automatice pruebas, construcción de artefactos y promoción a ambientes de producción cada vez que se integran cambios al repositorio principal de código. Este pipeline debe incluir verificaciones automáticas de estilo de código mediante linters, ejecución de suite de pruebas unitarias y de integración, análisis estático de seguridad que identifique vulnerabilidades conocidas en dependencias, y generación de previsualizaciones desplegables para cada pull request que faciliten revisiones de código contextualizado. El objetivo final es minimizar el riesgo de introducir regresiones y reducir el tiempo entre identificación de necesidad y disponibilidad de funcionalidad en producción.

El sexto objetivo específico se relaciona con garantizar seguridad integral del sistema mediante implementación de autenticación basada en tokens JWT con rotación automática, autorización granular mediante políticas de Row Level Security en la base de datos que impidan acceso no autorizado incluso si las credenciales del servidor son comprometidas, validación exhaustiva de todas las entradas de usuario tanto en cliente como en servidor mediante esquemas tipados que prevengan ataques de inyección, sanitización apropiada de contenido generado por usuarios antes de renderizado para prevenir XSS, y configuración de headers de seguridad HTTP que mitiguen vectores de ataque comunes. Adicionalmente, debe implementarse un sistema de respaldo automático de la base de datos con retención configurable que permita recuperación ante corrupción de datos.

4. Justificación

La realización de este proyecto se fundamenta en múltiples dimensiones que abarcan aspectos económicos, sociales, técnicos y académicos, conformando un conjunto coherente de razones que validan la inversión de recursos y esfuerzo en su ejecución. Desde la perspectiva económica, el desarrollo de capacidades de comercialización directa para productores de café especial representa una intervención con alto potencial de impacto en la rentabilidad del emprendimiento. Investigaciones del sector demuestran que la eliminación de intermediarios puede incrementar los márgenes de ganancia del productor entre 200% y 400%, transformando radicalmente la viabilidad financiera de operaciones cafeteras pequeñas que típicamente operan con márgenes extremadamente ajustados bajo modelos tradicionales de comercialización.

El mercado objetivo para cafés orgánicos premium ha experimentado crecimiento sostenido durante la última década, impulsado por tendencias de consumo consciente que priorizan aspectos de sostenibilidad ambiental, comercio justo y trazabilidad de alimentos. Reportes de la Specialty Coffee Association indican que el segmento de cafés especiales crece a tasas anuales superiores al 12%, significativamente por encima del crecimiento del mercado general de café que se mantiene relativamente estancado. Este contexto macroeconómico favorable sugiere que inversiones en infraestructura comercial para este segmento tienen alta probabilidad de generar retornos positivos en horizontes temporales razonables.

Desde la dimensión social, el proyecto contribuye al fortalecimiento del tejido empresarial regional mediante habilitación de capacidades competitivas para emprendimientos familiares que constituyen la base de la estructura económica santandereana. El sector cafetero es intensivo en mano de obra y genera empleo directo e indirecto para comunidades rurales con opciones

limitadas de fuentes alternativas de ingreso. Mejorar la rentabilidad de fincas cafeteras mediante mejor acceso al mercado tiene efectos multiplicadores que se traducen en mejores condiciones de vida para familias productoras, incentivos para permanencia en zonas rurales evitando migración hacia centros urbanos, y preservación de conocimientos tradicionales de agricultura sostenible que son patrimonio cultural de las regiones cafeteras colombianas.

El proyecto se alinea con políticas públicas nacionales y departamentales orientadas a promover la transformación digital del sector agropecuario como estrategia para cerrar brechas de productividad y competitividad que limitan el potencial de desarrollo del país. Iniciativas gubernamentales como la Agenda de Transformación Digital del Ministerio TIC han identificado el comercio electrónico agroalimentario como área prioritaria para inversión y fomento, ofreciendo recursos y programas de acompañamiento para proyectos que demuestren viabilidad técnica y comercial. Este contexto de política pública favorable incrementa las probabilidades de acceso a líneas de financiamiento, subsidios o incentivos tributarios que pueden facilitar la escalabilidad del emprendimiento.

Desde el punto de vista técnico, el proyecto ofrece la oportunidad de aplicar tecnologías y metodologías contemporáneas de desarrollo web en un caso de uso real con restricciones y requerimientos auténticos, trascendiendo el ámbito puramente académico de ejercicios teóricos para generar aprendizajes profundos derivados de enfrentar complejidades inherentes a sistemas de producción. La selección de Next.js como framework base responde a su creciente adopción en la industria y su diseño arquitectónico que facilita optimizaciones de rendimiento críticas para experiencia de usuario. Supabase como plataforma de backend-as-a-service permite concentrar esfuerzos de desarrollo en lógica de negocio específica de la aplicación en lugar de invertir tiempo en configuración de infraestructura básica, acelerando significativamente time-to-market.

La arquitectura propuesta implementa patrones de diseño ampliamente validados en la industria como separación de responsabilidades, inversión de dependencias y programación orientada a interfaces, todos los cuales contribuyen a mantenibilidad, testabilidad y escalabilidad del código. La adopción de TypeScript sobre JavaScript introduce verificación estática de tipos que previene categorías enteras de errores comunes en tiempo de desarrollo, mejorando significativamente la confiabilidad del sistema y reduciendo costos de debugging en fases posteriores. Estas decisiones técnicas no solo benefician el proyecto específico sino que establecen capacidades transferibles a futuros desarrollos de los autores.

El componente académico del proyecto es particularmente relevante considerando que constituye una aplicación práctica integrada de conocimientos adquiridos a lo largo del programa de formación en áreas diversas como bases de datos relacionales, programación orientada a objetos, arquitectura de software, diseño de interfaces de usuario, seguridad informática y gestión de proyectos tecnológicos. La naturaleza multidisciplinaria del sistema requiere articulación coherente de conceptos provenientes de múltiples cursos, demostrando competencias de síntesis e integración que son objetivos fundamentales de programas de educación superior en tecnología.

La documentación exhaustiva generada como parte del proyecto constituye un activo valioso que trasciende la entrega académica inmediata. La arquitectura detallada facilita futuras ampliaciones del sistema por parte de otros desarrolladores, el análisis de requerimientos sirve como referencia para proyectos similares en contextos comparables, y los desafíos técnicos documentados junto con sus soluciones contribuyen al acervo de conocimiento de la comunidad de desarrollo. En este sentido, el proyecto tiene un componente de contribución al conocimiento colectivo que justifica su realización desde una perspectiva de construcción de bienes públicos intelectuales.

5. Marco Teórico

5.1 Comercio Electrónico y Transformación Digital del Sector Agroalimentario

El comercio electrónico se define como la realización de transacciones comerciales mediante redes electrónicas, siendo internet el medio más prominente en la actualidad. Turban et al. (2018) caracterizan el comercio electrónico como un ecosistema que trasciende la simple digitación de procesos transaccionales para abarcar transformaciones fundamentales en modelos de negocio, relaciones con clientes y cadenas de suministro. La evolución del comercio electrónico se conceptualiza típicamente en generaciones sucesivas, siendo la primera centrada en presencia básica mediante sitios web informativos, la segunda en transacciones B2C mediante tiendas online, la tercera en personalización y recomendaciones algorítmicas, y la cuarta emergente enfocada en experiencias omnicanal que integran perfectamente interacciones digitales y físicas.

El sector agroalimentario ha experimentado adopción relativamente tardía de comercio electrónico comparado con industrias como retail de productos electrónicos o entretenimiento digital. Esta lentitud se atribuye a características inherentes de los productos agrícolas como perecibilidad, variabilidad de calidad, dependencia de cadenas de frío y regulaciones sanitarias complejas que complican la logística de comercialización directa. Sin embargo, investigaciones recientes documentan aceleración significativa en adopción durante el período 2020-2025, catalizad

a por disrupciones en canales tradicionales de distribución y cambios en preferencias de consumidores que valoran cada vez más conveniencia y transparencia en el origen de alimentos.

5.2 Arquitectura de Aplicaciones Web Modernas

La arquitectura de aplicaciones web contemporáneas ha evolucionado dramáticamente desde modelos monolíticos tradicionales donde toda la lógica residía en un único servidor hacia patrones distribuidos que separan frontend, backend y servicios especializados. Fielding (2000) formalizó el estilo arquitectónico REST que fundamenta la mayoría de APIs web actuales, definiendo restricciones como statelessness, cachabilidad y separación cliente-servidor que facilitan escalabilidad y mantenibilidad. La progresión hacia arquitecturas de microservicios descompone aplicaciones en servicios pequeños e independientemente desplegables, cada uno responsable de un dominio de negocio acotado.

Sin embargo, para proyectos con equipos pequeños o recursos limitados, Newman (2021) argumenta que arquitecturas monolíticas modulares representan un punto óptimo que captura beneficios de separación lógica sin incurrir en complejidad operacional de sistemas distribuidos. Este enfoque organiza el código en módulos cohesivos con interfaces bien definidas pero despliega todo como una unidad, simplificando debugging, transacciones y consistencia de datos. Next.js ejemplifica esta filosofía al proveer un framework que facilita modularización mediante su sistema de routing basado en filesystem y separación entre server components y client components.

5.3 Renderizado del Lado del Servidor y Optimización de Performance

El debate entre renderizado del lado del cliente y del servidor representa una tensión fundamental en desarrollo web moderno. Las aplicaciones de página única tradicionales delegan todo el renderizado al navegador, resultando en carga inicial lenta mientras JavaScript se descarga y ejecuta. Grigorik (2013) documenta que cada segundo adicional de tiempo de carga incrementa

tasas de abandono entre 7% y 10%, estableciendo performance como factor crítico de éxito. El renderizado del lado del servidor genera HTML completo en el servidor antes de enviarlo al cliente, mejorando drásticamente First Contentful Paint y facilitando indexación por motores de búsqueda.

Next.js implementa un modelo híbrido donde páginas pueden elegir estrategias de renderizado apropiadas a sus características. Páginas con contenido altamente dinámico utilizan Server-Side Rendering generando HTML fresco en cada request, páginas con contenido relativamente estático emplean Static Site Generation compilando HTML en tiempo de build, y páginas requiriendo interactividad compleja en cliente combinan renderizado inicial en servidor con hidratación posterior de interactividad. Esta flexibilidad permite optimizar el balance entre performance, frescura de datos y complejidad de implementación según requerimientos específicos de cada ruta.

5.4 Gestión de Estado en Aplicaciones React

La gestión de estado constituye uno de los desafíos centrales en aplicaciones React debido a la naturaleza unidireccional de flujo de datos. Redux emergió como solución dominante basada en arquitectura Flux con store centralizado, actions tipadas y reducers puros. Sin embargo, la verbosidad de Redux ha motivado alternativas como Zustand que simplifican drásticamente la API manteniendo predictibilidad. Daishi (2021) argumenta que stores minimalistas como Zustand son suficientes para la mayoría de aplicaciones que no requieren características avanzadas como time-travel debugging o middleware complejo.

La distinción entre estado del servidor y estado del cliente es conceptualmente importante. Tanstack Query popularizó el concepto de caching declarativo donde datos del servidor se

gestionan mediante queries con invalidación automática, separando esta responsabilidad del estado local de UI. Next.js App Router lleva este concepto más allá mediante Server Components que eliminan la necesidad de queries explícitas para datos que no cambian durante la sesión del usuario, simplificando significativamente la arquitectura de aplicaciones que priorizan server-side rendering.

5.5 Seguridad en Aplicaciones Web

La seguridad de aplicaciones web abarca múltiples vectores de ataque que deben mitigarse sistemáticamente. OWASP (2021) documenta las diez vulnerabilidades más críticas incluyendo inyección SQL, cross-site scripting, autenticación quebrada y exposición de datos sensibles. La defensa en profundidad requiere múltiples capas de protección reconociendo que ninguna medida individual es infalible. La validación de entradas debe ocurrir tanto en cliente como en servidor, siendo la validación del servidor la única considerada confiable desde perspectiva de seguridad dado que código cliente puede ser manipulado arbitrariamente.

Row Level Security en bases de datos PostgreSQL representa un enfoque particularmente robusto donde políticas de acceso se definen directamente en el esquema de base de datos, ejecutándose automáticamente para cada query independientemente de la aplicación que las emite. Este modelo previene escalación de privilegios incluso si la capa de aplicación es completamente comprometida, contrastando con arquitecturas donde la aplicación asume responsabilidad exclusiva de autorización. Supabase aprovecha RLS extensivamente, permitiendo políticas declarativas que verifican ownership de recursos o membership en roles antes de permitir lecturas o escrituras.

6. Metodología de Desarrollo

La metodología de desarrollo adoptada para este proyecto combina principios ágiles con prácticas de ingeniería de software disciplinadas, reconociendo que equipos pequeños requieren balance entre flexibilidad y rigor. El marco metodológico se estructura en cinco fases iterativas que se ejecutan en ciclos de dos semanas, permitiendo entregas incrementales de funcionalidad mientras se incorpora retroalimentación continua. Esta sección describe sistemáticamente cada fase junto con sus entregables específicos, criterios de aceptación y métricas de progreso.

La fase de descubrimiento y planificación establece fundamentos del proyecto mediante elicitación exhaustiva de requerimientos, análisis de stakeholders y definición de alcance. Esta fase involucra entrevistas semi-estructuradas con propietarios de Finca Miraflores para comprender flujos de trabajo actuales, puntos de dolor operativos y expectativas respecto al sistema. Las entrevistas se complementan con observación directa de procesos como recepción de pedidos, actualización de inventarios y preparación de despachos, identificando oportunidades de automatización que podrían no ser evidentes mediante entrevistas únicamente. Los requerimientos capturados se documentan mediante historias de usuario siguiendo el formato estándar de Cohn (2004) que especifica quién necesita la funcionalidad, qué necesitan lograr y por qué es valioso, facilitando priorización posterior.

El análisis competitivo examina sistemáticamente plataformas de comercio electrónico de cafés especiales tanto nacionales como internacionales, identificando mejores prácticas en diseño de interfaz, flujos de checkout, presentación de información del producto y estrategias de contenido. Este benchmarking informa decisiones de diseño asegurando que la solución desarrollada cumple o excede expectativas establecidas por competidores directos. Los hallazgos se sintetizan

en un documento de requisitos del sistema que sirve como contrato implícito entre desarrolladores y stakeholders, estableciendo claramente qué construir y con qué criterios se evaluará el éxito.

La fase de diseño arquitectónico traduce requerimientos funcionales en estructuras técnicas concretas mediante múltiples artefactos de diseño. El diagrama de arquitectura de alto nivel mapea componentes principales del sistema y sus relaciones, identificando fronteras entre módulos y especificando interfaces de comunicación. Este diagrama facilita razonamiento sobre escalabilidad, identificación de cuellos de botella potenciales y planificación de estrategias de caching. El diseño de base de datos normaliza el esquema hasta tercera forma normal para eliminar redundancia mientras introduce desnormalizaciones estratégicas donde consultas frecuentes se benefician de datos duplicados que evitan joins complejos.

Los wireframes de baja fidelidad esbozan layouts de páginas clave priorizando estructura informacional y flujos de usuario sobre detalles visuales. Estos wireframes se validan mediante sesiones de feedback con usuarios representativos del segmento objetivo, identificando tempranamente confusiones de navegación o expectativas no cumplidas. Los mockups de alta fidelidad incorporan paleta de colores, tipografía, espaciado y componentes visuales finales, sirviendo como especificación detallada para implementación. La guía de estilo documenta decisiones de diseño incluyendo jerarquía tipográfica, uso de color, espaciado consistente y comportamientos de interacción estándar, asegurando coherencia visual a lo largo de la aplicación.

La fase de implementación ejecuta la construcción del sistema mediante sprints de dos semanas con entregables funcionales al final de cada sprint. El desarrollo sigue principios de test-driven development donde pruebas se escriben antes que código de producción, forzando diseño

modular y facilitando refactoring posterior con confianza. Los componentes de interfaz se construyen en aislamiento utilizando Storybook, permitiendo iteración rápida sobre variantes visuales sin necesidad de ejecutar la aplicación completa. La integración continua mediante GitHub Actions ejecuta automáticamente pruebas en cada push al repositorio, detectando regresiones inmediatamente.

Las revisiones de código se realizan sistemáticamente mediante pull requests que requieren aprobación de al menos un revisor antes de fusión al branch principal. Estas revisiones verifican adherencia a estándares de código, identifican potenciales bugs lógicos, sugieren mejoras de rendimiento y aseguran que el código es comprensible para otros desarrolladores. Los comentarios en revisiones se documentan públicamente en GitHub, construyendo un historial de decisiones de diseño que facilita comprensión futura del razonamiento detrás de implementaciones específicas.

La fase de pruebas abarca múltiples niveles de verificación incluyendo pruebas unitarias de funciones y componentes individuales, pruebas de integración de flujos que atraviesan múltiples módulos, y pruebas end-to-end que simulan interacciones completas de usuario desde autenticación hasta completar una compra. Las pruebas de performance utilizan Lighthouse para auditar métricas de Web Vitals incluyendo Largest Contentful Paint, First Input Delay y Cumulative Layout Shift, estableciendo umbrales mínimos que cada página debe cumplir. Las pruebas de accesibilidad verifican navegación mediante teclado, compatibilidad con lectores de pantalla y contraste de color suficiente, asegurando que la aplicación es utilizable por personas con discapacidades.

Las pruebas de seguridad incluyen análisis estático mediante herramientas como ESLint security plugins que detectan patrones de código peligrosos, análisis de dependencias con npm audit que

identifica vulnerabilidades conocidas en paquetes de terceros, y penetration testing manual que intenta explotar vectores de ataque comunes como inyección SQL, XSS y CSRF. Los hallazgos de seguridad se priorizan según severidad utilizando el sistema CVSS, abordando inmediatamente vulnerabilidades críticas mientras se planifica remediación de issues menos severos.

La fase de despliegue y monitoreo establece la infraestructura de producción y sistemas de observabilidad necesarios para operar confiablemente el sistema. El despliegue se realiza mediante Vercel que provee edge network global, scaling automático y rollbacks instantáneos en caso de detectarse problemas. La configuración de variables de entorno segrega secretos de producción del código fuente, utilizando el sistema de secrets de Vercel que encripta valores sensibles en reposo. La base de datos de producción se configura con respaldos automáticos diarios con retención de 30 días, permitiendo recuperación ante corrupción de datos o errores operativos.

El monitoreo en producción captura múltiples señales de salud del sistema incluyendo logs de errores, métricas de rendimiento, traces de transacciones y analytics de uso. Los logs se agregan en Vercel Analytics que provee dashboards de errores por frecuencia, permitiendo priorización de correcciones basada en impacto a usuarios. Las métricas de rendimiento se monitoran mediante Real User Monitoring que captura tiempos de carga experimentados por usuarios reales bajo condiciones diversas de red y dispositivo, contrastando con pruebas sintéticas que no reflejan necesariamente experiencia en producción. Los analytics de uso rastrean conversiones de funnel de compra, identificando puntos de abandono que requieren optimización.

7. Arquitectura del Sistema

La arquitectura del sistema implementa un patrón monolítico modular optimizado para despliegue en plataformas serverless, balanceando la simplicidad operativa de un monolito con los beneficios de organización de microservicios. Esta sección describe exhaustivamente cada capa arquitectónica, las tecnologías seleccionadas para implementarlas, los patrones de diseño aplicados y las justificaciones técnicas de decisiones estructurales.

7.1 Vista de Alto Nivel

El sistema se estructura en tres capas principales siguiendo el patrón clásico de separación de responsabilidades. La capa de presentación implementa toda la lógica de interfaz de usuario incluyendo componentes React, layouts, páginas y lógica de enrutamiento. Esta capa se ejecuta tanto en el servidor durante renderizado inicial como en el cliente después de hidratación, aprovechando las capacidades híbridas de Next.js App Router. La separación entre Server Components y Client Components permite optimización donde componentes sin interactividad se renderizan exclusivamente en servidor, reduciendo el bundle de JavaScript enviado al cliente.

La capa de lógica de negocio encapsula operaciones del dominio de la aplicación incluyendo validación de reglas de negocio, transformaciones de datos y orquestación de flujos que involucran múltiples recursos. Esta capa se implementa mediante servicios especializados que exponen interfaces tipadas y abstraen detalles de implementación como fuentes de datos específicas. La organización por servicios facilita testing mediante mocking de dependencias y permite que múltiples consumidores reutilicen la misma lógica sin duplicación.

La capa de acceso a datos gestiona todas las interacciones con sistemas persistentes incluyendo la base de datos PostgreSQL, el storage de archivos y servicios externos como Stripe. Esta capa

expone interfaces consistentes que ocultan complejidades de queries SQL, manejo de conexiones y transformación de formatos de datos externos a representaciones internas. La centralización de acceso a datos facilita auditoría de operaciones sensibles, implementación de caching y migración futura a tecnologías alternativas sin impactar capas superiores.

7.2 Tecnologías y Justificaciones

Next.js fue seleccionado como framework principal fundamentándose en su arquitectura que optimiza automáticamente muchos aspectos de performance que requerirían configuración manual en alternativas. El App Router introduce Server Components que renderiza

n exclusivamente en servidor sin enviar JavaScript al cliente, reduciendo drásticamente tamaños de bundles. El sistema de routing basado en filesystem elimina configuración de routes explícita mientras provee características avanzadas como layouts anidados, loading states y error boundaries. La integración nativa con Vercel facilita despliegues con zero-configuration mientras habilita características como Edge Functions y Middleware que ejecutan lógica geográficamente cerca de usuarios.

TypeScript se adoptó en modo estricto para introducir verificación estática de tipos que previene categorías enteras de errores. La inferencia de tipos permite que gran parte del código se beneficie de type safety sin anotaciones explícitas verbosas, mientras que tipos complejos de dominio se definen explícitamente mediante interfaces que sirven como documentación viva del sistema. La integración entre TypeScript y IDEs modernos habilita autocompletado inteligente, refactoring seguro y navegación de código que incrementan productividad de desarrollo significativamente.

Supabase provee una suite integrada de servicios backend que aceleran desarrollo al eliminar necesidad de configurar infraestructura básica. La base de datos PostgreSQL ofrece características empresariales como transacciones ACID, índices complejos, triggers y funciones almacenadas. El sistema de autenticación maneja flujos completos incluyendo registro, login, recuperación de contraseña y gestión de sesiones mediante tokens JWT. El storage de archivos provee URLs firmadas temporalmente que permiten uploads directos desde cliente sin exponer credenciales del servidor.

Stripe se integró como pasarela de pagos por su API bien diseñada, documentación exhaustiva y manejo robusto de casos edge en procesamiento de pagos. El modo de Checkout hosteado externaliza la interfaz de captura de información de pago, reduciendo scope de cumplimiento PCI DSS requerido. Los webhooks permiten que el sistema reaccione a eventos como confirmaciones de pago asíncronamente sin mantener conexiones abiertas durante procesamiento que podría tomar varios segundos. La reconciliación de órdenes mediante idempotency keys previene duplicación de cargos en caso de retries de red.

7.3 Patrones de Diseño Aplicados

El patrón Repository abstrae acceso a datos mediante interfaces que exponen operaciones de negocio sin revelar detalles de persistencia. Cada entidad del dominio como Product, Order y User tiene un servicio correspondiente que encapsula todas las operaciones relacionadas. Esta organización facilita testing mediante implementaciones mock que no requieren base de datos real y permite optimizaciones como caching transparente sin modificar código de consumidores.

El patrón Strategy se aplica en validación de datos donde esquemas Zod definen reglas de validación que pueden intercambiarse sin modificar lógica de formularios. La misma estructura de datos puede validarse con reglas diferentes según contexto, por ejemplo requerimientos más

estrictos en creación versus actualización. Esta flexibilidad facilita evolución de reglas de negocio sin refactoring extensivo.

El patrón Observer se manifiesta en el sistema de estado de Zustand donde componentes se suscriben a slices específicos del store, re-renderizando únicamente cuando datos relevantes cambian. Esta suscripción selectiva previene cascadas de re-renderizado innecesarias que degradarían performance. Los componentes no necesitan conocimiento de otros suscriptores, facilitando composición de UI independiente.

El patrón Facade simplifica interacciones con subsistemas complejos exponiendo interfaces de alto nivel. Por ejemplo, el servicio de órdenes orquesta múltiples operaciones incluyendo creación del registro de orden, actualización de inventarios, generación de sesión de Stripe y envío de emails de confirmación, encapsulando esta complejidad detrás de una única función que los componentes invocan.

---

8. Análisis de Requerimientos

El análisis de requerimientos establece el contrato fundamental entre stakeholders y el equipo de desarrollo, especificando exhaustivamente qué debe hacer el sistema, bajo qué condiciones opera y con qué criterios se mide el éxito. Esta sección categoriza requerimientos en funcionales que describen comportamientos observables del sistema, no funcionales que especifican atributos de calidad, y restricciones que limitan el espacio de soluciones viables.

8.1 Requerimientos Funcionales

El sistema debe permitir a usuarios no autenticados navegar el catálogo completo de productos, visualizando para cada producto su nombre, descripción detallada, precio unitario, opciones de

presentación disponibles, fotografías en alta resolución desde múltiples ángulos, disponibilidad de stock y calificaciones de clientes anteriores. La navegación debe soportar filtrado por categorías, ordenamiento por criterios como precio ascendente o descendente, popularidad y fecha de adición, y búsqueda textual que localiza productos cuyos nombres o descripciones contienen términos especificados. Los productos agotados deben indicarse claramente imposibilitando adición al carrito hasta que stock sea repuesto.

El sistema debe implementar funcionalidad de carrito de compras que persiste localmente en el navegador, permitiendo que usuarios agreguen productos con cantidades específicas, modifiquen cantidades posteriormente, remuevan items individualmente o vacíen completamente el carrito. El carrito debe calcular automáticamente subtotales por item, total general y mostrar cantidades totales de items. El estado del carrito debe sincronizarse entre pestañas abiertas del mismo navegador asegurando consistencia. El carrito debe validar que cantidades solicitadas no excedan stock disponible, alertando al usuario si intenta agregar cantidades no disponibles.

El sistema debe facilitar proceso de checkout que captura información de envío incluyendo nombre completo del destinatario, dirección postal completa con ciudad y código postal, número telefónico de contacto y email para confirmaciones. El sistema debe validar formato de emails y números telefónicos rechazando valores inválidos antes de proceder. La integración con Stripe debe redirigir usuarios a página hosteada segura donde ingresan información de pago, completando la transacción mediante flujo estándar de Stripe Checkout que soporta tarjetas de crédito, débito y métodos alternativos según configuración de la cuenta de comerciante.

El sistema debe generar registros de órdenes completos que capturen timestamp exacto de creación, usuario asociado si está autenticado o null para compras de invitado, listado detallado de items incluyendo producto, cantidad y precio unitario en el momento de compra, totales

calculados, información de envío capturada, identificador único de transacción de Stripe y estado inicial de orden pendiente. El sistema debe actualizar automáticamente estado de órdenes a confirmado cuando recibe webhook de Stripe indicando pago exitoso, fallido si el pago es rechazado, o cancelado si el usuario abandona el flujo.

El sistema debe proveer panel administrativo accesible únicamente a usuarios con rol de administrador verificado mediante autenticación robusta. El panel debe listar todas las órdenes recibidas con posibilidad de filtrar por estado, rango de fechas o búsqueda de clientes específicos. Los administradores deben poder visualizar detalles completos de cada orden incluyendo items, totales, información de cliente y historial de cambios de estado. El panel debe permitir actualización manual de estados para reflejar progresos operativos como procesamiento iniciado, empacado completado, enviado con número de tracking o entregado.

El sistema debe implementar interfaz de gestión de productos donde administradores pueden crear productos nuevos especificando todos los campos requeridos, editar productos existentes modificando cualquier campo incluyendo imágenes, y eliminar productos que ya no se comercializan. La eliminación debe ser lógica en lugar de física, marcando productos como inactivos pero preservando registros históricos para integridad referencial de órdenes pasadas. La gestión de inventarios debe permitir ajustes manuales de cantidades disponibles, mostrando alertas cuando stock cae bajo umbrales configurados.

8.2 Requerimientos No Funcionales

El rendimiento del sistema debe garantizar que el tiempo hasta First Contentful Paint no exceda 1.8 segundos en conexiones 4G simuladas y dispositivos móviles de gama media, medido mediante Lighthouse en modo incógnito sin extensiones. El tiempo hasta Largest Contentful

Paint debe mantenerse bajo 2.5 segundos para cumplir con estándares de Web Vitals. El Time to Interactive debe alcanzarse en menos de 3.5 segundos permitiendo que usuarios interactúen con la interfaz rápidamente. Las transiciones entre páginas deben completarse en menos de 200 milisegundos para mantener percepción de fluidez.

La escalabilidad debe permitir que el sistema soporte incrementos de 10 veces en tráfico durante períodos promocionales sin degradación de performance, aprovechando auto-scaling de Vercel que provision instancias serverless dinámicamente. La base de datos debe configurarse con connection pooling que reutiliza conexiones eficientemente evitando overhead de establecimiento repetido. Las consultas frecuentes deben cachearse mediante técnicas de memoización o caching de queries a nivel de framework, reduciendo carga en base de datos.

La seguridad debe implementar autenticación basada en tokens JWT con expiración corta de 1 hora y refresh tokens con expiración de 7 días almacenados en cookies httpOnly que no son accesibles desde JavaScript cliente. Las contraseñas deben hashearse mediante bcrypt con cost factor mínimo de 12 antes de almacenamiento, imposibilitando recuperación de passwords en texto plano incluso si base de datos es comprometida. Row Level Security en Supabase debe forzar que usuarios solo puedan leer y escribir recursos que poseen, verificando ownership en cada query.

La usabilidad debe priorizar diseño mobile-first donde layouts se optimizan para pantallas pequeñas primero y se expanden progresivamente en viewports más amplios. Los controles táctiles deben tener área mínima de 44x44 píxeles para facilitar activación precisa. Los formularios deben mostrar validación inline que alerta errores inmediatamente en lugar de esperar submit completo. Los mensajes de error deben ser específicos indicando exactamente qué corregir en lugar de mensajes genéricos como error inesperado.

La mantenibilidad requiere que el código adhiera a guías de estilo consistentes forzadas por ESLint y Prettier que formatean automáticamente el código. La cobertura de pruebas debe alcanzar mínimo 70 por ciento de líneas de código para lógica de negocio crítica. La documentación inline mediante comentarios JSDoc debe describir contratos de funciones públicas incluyendo tipos de parámetros, valores de retorno y excepciones lanzadas. El repositorio debe incluir README exhaustivo que explique setup de desarrollo, estructura de directorios y comandos comunes.

---

9. Diseño e Implementación

Esta sección detalla las decisiones de diseño de interfaz, la implementación de componentes clave, los flujos de datos complejos y las optimizaciones aplicadas para cumplir requerimientos de performance y usabilidad. El diseño sigue principios de arquitectura limpia donde dependencias apuntan hacia dentro desde capas externas hacia el núcleo del dominio, facilitando testing y evolución independiente de capas.

9.1 Diseño de Interfaz de Usuario

La paleta de colores se deriva de elementos naturales asociados con café y sostenibilidad ambiental. El verde oscuro representa las montañas cafeteras y vegetación abundante, el marrón evoca granos de café tostado, el dorado sugiere calidad premium y exclusividad, mientras que tonos beige y crema mantienen neutralidad que no compite visualmente con fotografías de productos. El contraste entre texto y fondos cumple WCAG 2.1 nivel AA como mínimo, asegurando legibilidad para usuarios con deficiencias visuales leves.

La tipografía combina Yeseva One para títulos que aporta personalidad distintiva con serifas elegantes, y Rethink Sans para cuerpo de texto que prioriza legibilidad en tamaños pequeños con altura x generosa y espaciado abierto. La jerarquía tipográfica establece ocho niveles desde display de 72 píxeles hasta small de 12 píxeles, cada uno con pesos y line-heights apropiados. Los títulos principales utilizan pesos bold de 700, subtítulos medium de 500 y cuerpo regular de 400, creando ritmo visual que guía atención del usuario.

El sistema de espaciado sigue progresión geométrica basada en múltiplos de 4 píxeles desde 4, 8, 12, 16, 24, 32, 48, 64 hasta 96, facilitando consistencia y ritmo visual. Los componentes utilizan padding interno de 16 píxeles como base, incrementando a 24 o 32 para elementos destacados. Los márgenes entre secciones mayores utilizan 64 o 96 píxeles creando respiración que reduce carga cognitiva. El grid layout se basa en 12 columnas en desktop, colapsando a 4 en tablet y 1 en móvil, con gutters de 24 píxeles.

Los componentes de interfaz se organizan jerárquicamente en tres niveles de abstracción. Los componentes primitivos como Button, Input y Card son altamente reutilizables sin lógica de negocio específica, aceptando props que customiz

an apariencia y comportamiento. Los componentes de composición como ProductCard y OrderSummary combinan primitivos con lógica presentacional ligera, estructurando información del dominio. Los componentes de páginas orquestan composición de componentes menores, gestionan estado local mediante hooks y se conectan a servicios para operaciones del servidor.

9.2 Implementación del Catálogo de Productos

El catálogo de productos se implementa como Server Component que obtiene datos directamente en el servidor durante renderizado inicial, eliminando waterfalls de red donde cliente debe

esperar JavaScript, después hacer fetch de datos y finalmente renderizar. La consulta a Supabase se optimiza mediante índices sobre columnas frecuentemente filtradas como categoría y disponibilidad, y utiliza pagination mediante cursor en lugar de offset para escalar eficientemente con grandes volúmenes de productos. El resultado se serializa como JSON incrustado en HTML inicial, permitiendo hidratación instantánea sin round-trip adicional.

Las imágenes de productos se optimizan mediante componente Image de Next.js que genera versiones redimensionadas automáticamente para diferentes viewports y formatos modernos como WebP con fallback a JPEG para navegadores legacy. El atributo priority marca imágenes above-the-fold para carga prioritaria mientras lazy loading difiere imágenes fuera de viewport inicial. Los placeholders con dimensiones exactas previenen layout shift cuando imágenes cargan, manteniendo Cumulative Layout Shift bajo 0.1.

El filtrado y ordenamiento se implementa mediante query parameters que se sincronizan con URL, permitiendo que resultados filtrados sean compartibles y navegables mediante botones adelante y atrás del navegador. La actualización de filtros dispara navegación mediante router.push preservando estado del carrito y scroll position. La generación de URLs canónicas con parámetros ordenados consistentemente facilita caching de CDN, maximizando cache hit rate.

9.3 Implementación del Carrito de Compras

El carrito utiliza Zustand para estado global accesible desde cualquier componente sin prop drilling. El store expone actions tipadas para addItem, updateQuantity, removeItem y clearCart que actualizan el estado inmutablemente mediante spreading de objetos. La persistencia en localStorage se logra mediante middleware de Zustand que serializa el estado después de cada

mutation y restaura en inicialización. La sincronización entre tabs se implementa escuchando eventos storage que disparan cuando otras tabs modifican localStorage.

La validación de cantidades verifica contra stock disponible consultando servicio de productos en cada adición o actualización. La validación ocurre tanto en cliente para feedback inmediato como en servidor durante checkout para prevenir race conditions donde múltiples usuarios agotan stock simultáneamente. El servidor utiliza transacciones de base de datos con locking optimista que verifican stock y decrementan atómicamente, rechazando transacciones si stock insuficiente.

El cálculo de totales se memoiza mediante useMemo que recalcula únicamente cuando items del carrito cambian, evitando sumas redundantes en cada render. La precisión decimal utiliza operaciones sobre enteros multiplicando por 100 para representar centavos, evitando errores de redondeo de punto flotante. Los precios se formatean mediante Intl.NumberFormat con locale apropiado y símbolo de moneda, respetando convenciones regionales.

9.4 Integración de Pasarela de Pagos

El proceso de checkout crea sesión de Stripe en el servidor mediante API server-side que especifica line items con nombres, cantidades y precios, URLs de éxito y cancelación, y metadata que vincula la sesión a la orden. El identificador de sesión se retorna al cliente que redirige a Checkout hosteado por Stripe, transfiriendo control del flujo de pago. La página de éxito consulta metadata de la sesión para mostrar confirmación específica de la orden.

Los webhooks de Stripe se verifican mediante firma criptográfica que valida que requests provienen legítimamente de Stripe y no fueron manipulados en tránsito. El handler deserializa eventos, extrae metadata de la orden, actualiza estado en base de datos y dispara acciones

secundarias como envío de email de confirmación. El handler es idempotente procesando eventos duplicados sin efectos secundarios adicionales, crucial dado que Stripe puede reenviar eventos múltiples veces.

La reconciliación de órdenes maneja casos edge como usuarios que abandonan flujo después de pago exitoso antes de redirección, verificando periódicamente sesiones de Stripe contra órdenes pendientes y marcándolas como pagadas si Stripe confirma pago. Las órdenes que permanecen pendientes más de 24 horas se marcan automáticamente como expiradas, liberando reservas de inventario.

---

10. Integración de Tecnologías

Esta sección profundiza en cómo tecnologías específicas se configuran, integran y optimizan para trabajar cohesivamente. Las integraciones siguientes representan puntos críticos donde decisiones de implementación impactan significativamente capacidades funcionales y no funcionales del sistema.

10.1 Configuración de Next.js

La configuración de Next.js en `next.config.js` especifica dominios externos permitidos para optimización de imágenes, evitando que URLs maliciosas abuse del endpoint de optimización. La configuración de redirecciones maneja migración de URLs legacy a nueva estructura, preservando posicionamiento en motores de búsqueda. Los headers personalizados establecen Content Security Policy que restringe fuentes permitidas de scripts, estilos y conexiones, mitigando ataques de inyección.

El archivo de middleware intercepta requests antes de llegar a rutas, verificando autenticación para rutas protegidas y redirigiendo a login si necesario. El middleware también implementa rate limiting simple que rechaza requests excesivos de misma IP en ventana temporal, previniendo abuso. El middleware ejecuta en Edge Runtime que inicia instantáneamente sin cold starts, crucial para latencia baja.

La configuración de paths alias mapea imports absolutos como `@/components` a rutas relativas, eliminando imports con múltiples niveles de parent directories que son propensos a errores y difíciles de refactorizar. La configuración de TypeScript habilita modo estricto que fuerza verificación exhaustiva de tipos, detectando errores potenciales que pasarían desapercibidos en modo leniente.

10.2 Configuración de Supabase

El cliente de Supabase para componentes del servidor se inicializa leyendo credenciales de variables de entorno y caching la instancia para reutilización. Las cookies se utilizan para persistir sesión entre requests del servidor, permitiendo que Server Components accedan datos personalizados del usuario actual. El cliente valida tokens JWT en cada request, rechazando tokens expirados o inválidos.

El cliente de Supabase para componentes del cliente se inicializa con clave pública anon que tiene permisos limitados definidos por Row Level Security. Las políticas RLS previenen escalación de privilegios incluso con clave expuesta, verificando que usuarios solo accedan recursos propios. Las suscripciones realtime permiten que componentes reaccionen a cambios de datos, útil para notificaciones de administradores cuando llegan órdenes nuevas.

La generación de tipos TypeScript desde esquema de Supabase automatiza sincronización entre definición de tablas y tipos usados en código, previniendo discrepancias que causarían errores runtime. Los tipos generados incluyen tablas, vistas, funciones y enums, cubriendo exhaustivamente entidades del dominio. La regeneración de tipos se incluye en workflow de CI para detectar cambios de esquema no sincronizados.

10.3 Optimizaciones de Performance

La división de código automática de Next.js genera bundles separados por ruta, cargando únicamente JavaScript necesario para página actual. Los dynamic imports aplican code splitting manual para componentes pesados que no son necesarios inicialmente, como mapas interactivos cargados solo cuando usuario navega a página de ubicación.

El caching de Next.js almacena resultados de fetches del servidor entre builds, regenerando únicamente cuando datos cambian. La revalidación incremental permite actualización de páginas estáticas sin rebuild completo, especificando intervalos de revalidación por ruta. Las rutas dinámicas se generan estáticamente en build time para combinaciones comunes de parámetros, sirviendo HTML precomputado para mayoría de requests.

El prefetching de enlaces visible en viewport carga datos de rutas vinculadas antes de navegación, haciendo transiciones percibidas como instantáneas. El componente Link de Next.js implementa prefetching automáticamente, configurable mediante prop prefetch. El lazy loading de imágenes difiere descarga hasta scroll near, reduciendo carga inicial de página.

---

11. Pruebas y Validación

La estrategia de testing abarca múltiples niveles de abstracción desde pruebas unitarias de funciones individuales hasta pruebas end-to-end de flujos completos de usuario. Esta pirámide de testing balancea cantidad de pruebas en cada nivel reconociendo que pruebas de nivel bajo son más rápidas y baratas mientras que pruebas de nivel alto proveen mayor confianza pero son más frágiles y lentas.

11.1 Pruebas Unitarias

Las pruebas unitarias verifican comportamiento de funciones puras y componentes aislados sin dependencias externas. Las funciones de utilidad como formatters de precios, validadores de datos y transformadores se prueban exhaustivamente con casos normales, casos edge y casos de error. Los componentes presentacionales se prueban renderizando con props específicos y aseverando estructura DOM esperada, texto visible y clases CSS aplicadas.

Los mocks reemplazan dependencias externas como servicios de base de datos con implementaciones controladas que retornan datos predefinidos, permitiendo pruebas determinísticas. Los mocks se configuran antes de cada prueba y se limpian después para evitar contaminación entre tests. Los matchers personalizados extienden expect con aserciones específicas del dominio que mejoran legibilidad.

La cobertura de código se mide mediante Jest que instrumenta el código durante ejecución de pruebas, reportando líneas, branches y funciones cubiertas. El umbral mínimo de cobertura se configura en 70 por ciento para archivos de lógica de negocio, fallando el build si cobertura cae bajo este nivel. La cobertura 100 por ciento no se persigue reconociendo rendimientos decrecientes.

11.2 Pruebas de Integración

Las pruebas de integración verifican interacción correcta entre múltiples módulos que colaboran para implementar funcionalidad. El flujo de creación de órdenes se prueba invocando acción del servidor con datos de prueba, verificando que registro de orden se crea en base de datos con campos correctos, inventario se decrementa apropiadamente y sesión de Stripe se genera exitosamente. La base de datos de prueba se resetea antes de cada test para garantizar aislamiento.

Las pruebas de API validan endpoints REST mediante requests HTTP simulados, verificando códigos de estado, headers de respuesta y estructura de JSON retornado. Las pruebas cubren casos exitosos y escenarios de error como autenticación faltante, validación fallida y recursos no encontrados. Los webhooks se prueban construyendo payloads sintéticos firmados con secret de prueba.

La comunicación entre frontend y backend se prueba mediante tests que montan componentes conectados a servicios mock que simulan responses del servidor. Las pruebas verifican que loading states se muestran durante fetching, datos se renderizan correctamente al completar y errores se manejan gracefully mostrando mensajes apropiados.

11.3 Pruebas End-to-End

Las pruebas end-to-end simulan interacciones completas de usuario mediante automatización de navegador, validando que sistema funciona correctamente desde perspectiva de usuario final. Playwright ejecuta scripts que navegan aplicación, llenan formularios, hacen clicks y verifican contenido renderizado. Las pruebas cubren flujos críticos como registro de usuario, búsqueda de productos, adición a carrito y completar compra.

Los tests E2E se ejecutan contra deploy de preview generado automáticamente para cada pull request, validando cambios en ambiente idéntico a producción antes de fusión. Las fallas son capturadas como screenshots y videos que facilitan debugging. Los tests se ejecutan paralelamente en múltiples navegadores incluyendo Chrome, Firefox y Safari para detectar incompatibilidades.

Los tests de regresión visual comparan screenshots de páginas clave contra baselines aprobadas, detectando cambios involuntarios de estilo. Las diferencias se revisan manualmente aprobando cambios intencionales o rechazando regresiones. El threshold de diferencia configurable permite cambios menores como antialiasing sin disparar falsos positivos.

---

12. Despliegue y Producción

El proceso de despliegue automatiza completamente la promoción de código desde repositorio hasta ambiente de producción accesible públicamente, implementando prácticas de integración y despliegue continuo que aceleran iteraciones mientras mantienen estabilidad. Esta sección describe infraestructura de hosting, configuración de ambientes, estrategias de rollout y monitoreo en producción.

12.1 Infraestructura de Hosting

Vercel fue seleccionado como plataforma de hosting por su integración nativa con Next.js y capacidades de edge network que sirven contenido desde ubicaciones geográficamente cercanas a usuarios finales, minimizando latencia. El plan gratuito de Vercel es suficiente para tráfico inicial mientras que planes pagos escalan automáticamente según demanda sin configuración

manual. El deploy de cada commit a branch principal actualiza automáticamente producción después de pasar verificaciones de calidad.

La base de datos PostgreSQL hospedada en Supabase opera en instancias dedicadas con backups automáticos diarios, recuperación point-in-time y replicación para alta disponibilidad. La configuración de connection pooling optimiza reutilización de conexiones que son recurso limitado. Los índices de base de datos se crean sobre columnas frecuentemente consultadas acelerando queries de productos, órdenes y usuarios.

El storage de imágenes utiliza buckets de Supabase con políticas de acceso que permiten lecturas públicas pero restringen escrituras a usuarios autenticados. Las imágenes se sirven mediante CDN de Supabase que cachea contenido globalmente. Las URLs prefirmadas temporales facilitan uploads seguros directamente desde cliente sin exponer credenciales permanentes.

12.2 Variables de Entorno y Secretos

Las variables de entorno segregan configuración sensible del código fuente, permitiendo que mismo código se despliegue en múltiples ambientes con configuraciones diferentes. Las claves públicas como URL de API de Supabase y publishable key de Stripe se almacenan con prefijo NEXT_PUBLIC para exposición al cliente. Las claves secretas como service role key de Supabase y secret key de Stripe se mantienen server-side únicamente.

Los secretos se almacenan en dashboard de Vercel encriptados en reposo y se inyectan como variables de entorno durante runtime. Los secrets nunca se comiten al repositorio git que es potencialmente accesible por terceros. El archivo `.env.example` documenta variables requeridas sin valores reales, facilitando configuración de nuevos ambientes de desarrollo.

La rotación periódica de secretos limita ventana de exposición si claves son comprometidas. Las claves de API se regeneran cada 90 días y se actualizan en todas las configuraciones de ambiente. El sistema de auditoría registra cuándo y quién accede secretos, permitiendo investigación de incidentes de seguridad.

12.3 Monitoreo y Observabilidad

El monitoreo en producción captura múltiples señales de salud del sistema permitiendo detección proactiva de problemas y análisis post-mortem de incidentes. Vercel Analytics provee dashboards de errores JavaScript que ocurren en clientes, agregados por frecuencia, severidad y rutas afectadas. Los errores incluyen stack traces que facilitan identificación de líneas problemáticas.

El logging estructurado emite eventos JSON con campos consistentes como timestamp, nivel de severidad, mensaje y contexto adicional. Los logs se agregan en sistema centralizado que permite búsqueda y filtrado eficientes. Los logs de acceso registran cada request con método HTTP, ruta, código de estado y duración, facilitando análisis de patrones de tráfico.

Las alertas se configuran para condiciones críticas como tasa de errores excediendo 5 por ciento, tiempo de respuesta p95 superando 3 segundos o disponibilidad cayendo bajo 99 por ciento. Las notificaciones se envían mediante email y webhooks a herramientas de incident management. Los runbooks documentan procedimientos de respuesta para alertas comunes acelerando remediación.

13. Resultados

La implementación exitosa del sistema generó resultados medibles en múltiples dimensiones incluyendo métricas técnicas de performance, indicadores de negocio de adopción y usabilidad, y aprendizajes de proceso de desarrollo. Esta sección presenta resultados cuantitativos y cualitativos obtenidos después de lanzamiento inicial.

13.1 Métricas de Performance

Las auditorías de Lighthouse realizadas en ambiente de producción demuestran cumplimiento de objetivos de performance establecidos en requerimientos no funcionales. La página de inicio alcanza score de 98 sobre 100 en Performance con First Contentful Paint de 1.2 segundos y Largest Contentful Paint de 1.8 segundos, significativamente bajo los umbrales objetivo de 1.8 y 2.5 segundos respectivamente. El Total Blocking Time de 150 milisegundos indica que la página responde a interacciones rápidamente.

El bundle de JavaScript inicial pesa 85 kilobytes comprimidos con gzip, resultado de code splitting agresivo que carga únicamente código necesario para ruta actual. Las imágenes optimizadas en formato WebP reducen tamaño promedio en 60 por ciento comparado con JPEG equivalente manteniendo calidad perceptual. El lazy loading de imágenes reduce carga inicial de página de catálogo en 40 por ciento.

El tiempo de respuesta promedio del servidor medido en p50 es 180 milisegundos y p99 es 420 milisegundos, indicando latencia consistentemente baja incluso en perc

entiles extremos. La disponibilidad del sistema durante primer mes fue 99.8 por ciento con downtime planificado para mantenimiento de base de datos. El auto-scaling de Vercel manejó picos de tráfico 5 veces superiores al promedio sin degradación observable.

13.2 Indicadores de Negocio

El lanzamiento del sistema expandió significativamente alcance geográfico de Café Yamara con órdenes recibidas desde 8 ciudades diferentes en contraste con ventas previas limitadas a Bucaramanga y área metropolitana. El ticket promedio de compra online de 78,000 pesos supera el ticket promedio de ventas presenciales de 52,000 pesos, sugiriendo que clientes online compran volúmenes mayores o productos premium con mayor frecuencia.

La tasa de conversión del funnel de compra alcanzó 3.2 por ciento calculada como órdenes completadas dividido entre visitas únicas, comparable a benchmarks de industria para comercio electrónico de alimentos especiales que típicamente oscilan entre 2 y 5 por ciento. El análisis de abandono identifica la página de checkout como punto crítico donde 45 por ciento de usuarios abandonan, sugiriendo oportunidades de optimización como reducir campos requeridos o agregar opciones de pago adicionales.

El tiempo promedio de procesamiento de órdenes se redujo de 4 horas bajo sistema manual a 45 minutos con sistema automatizado que notifica instantáneamente nuevas órdenes y consolida información de cliente y productos en interfaz centralizada. La reducción de errores de captura de datos de 12 por ciento a menos de 1 por ciento elimina reprocesos y mejora satisfacción de cliente.

13.3 Feedback de Usuarios

Las encuestas de satisfacción enviadas post-compra obtuvieron respuesta de 28 por ciento de clientes con score promedio de 4.3 sobre 5 en satisfacción general. Los comentarios cualitativos destacan positivamente la facilidad de navegación, claridad de información de productos y

proceso de checkout sin fricciones. Las áreas de mejora identificadas incluyen deseo de más métodos de pago como transferencias bancarias y mayor transparencia sobre tiempos de entrega.

Las sesiones de usability testing con 6 usuarios representativos revelaron que usuarios completan tareas de búsqueda de producto y adición a carrito en promedio en 2 minutos 15 segundos, indicando interfaz intuitiva. Los puntos de confusión identificados incluyen falta de indicador visual claro de item agregado a carrito y navegación no obvia hacia información de proceso del café. Estas observaciones informan priorización de mejoras futuras.

---

14. Discusión

Los resultados obtenidos validan la hipótesis central de que una plataforma web bien diseñada puede transformar capacidades comerciales de productores agrícolas pequeños, permitiéndoles capturar mayor valor agregado mediante eliminación de intermediarios y conexión directa con consumidores conscientes. Las métricas de performance demuestran que tecnologías modernas como Next.js y Supabase permiten desarrollar aplicaciones con experiencia de usuario comparable a soluciones empresariales costosas, democratizando capacidades digitales para emprendimientos con recursos limitados.

La expansión geográfica de ventas más allá del mercado local inmediato confirma que limitaciones previas eran primariamente de acceso al mercado en lugar de demanda insuficiente para café orgánico premium. La disposición de clientes online a pagar ticket promedio superior sugiere que comunicación efectiva de propuesta de valor mediante storytelling sobre origen, proceso y sostenibilidad justifica premium pricing. Este hallazgo refuerza importancia de inversión en contenido de calidad que educa consumidores sobre diferenciadores del producto.

La reducción dramática de tiempo de procesamiento de órdenes y errores de captura valida beneficios de automatización incluso para volúmenes transaccionales relativamente bajos. La consolidación de información en interfaz administrativa única elimina búsquedas en múltiples sistemas y reduce carga cognitiva de operadores. Sin embargo, la curva de aprendizaje inicial para usuarios no técnicos requirió capacitación estructurada, sugiriendo que futuros proyectos deben presupuestar tiempo para onboarding de usuarios.

Las limitaciones del estudio incluyen período de observación relativamente corto de 2 meses que impide análisis de estacionalidad o comportamiento de retención de clientes a largo plazo. Adicionalmente, el lanzamiento coincidió con campaña de marketing en redes sociales que pudo haber inflado artificialmente métricas iniciales. Estudios futuros deberían extender período de observación y intentar separar efectos atribuibles a plataforma digital versus marketing.

---

15. Conclusiones

El desarrollo e implementación de la plataforma web para Café Yamara logró exitosamente los objetivos establecidos, entregando un sistema funcional que integra capacidades institucionales, transaccionales y administrativas mediante arquitectura escalable y mantenible. Las decisiones arquitectónicas de utilizar Next.js, TypeScript, Supabase y Stripe demostraron ser acertadas, permitiendo velocidad de desarrollo alta sin comprometer calidad técnica o seguridad.

Los resultados cuantitativos confirman que el sistema cumple requerimientos de performance con métricas de Web Vitals dentro de rangos óptimos, provee experiencia de usuario fluida medida mediante scores de Lighthouse superiores a 95, y escala apropiadamente ante incrementos de tráfico sin intervención manual. Los resultados de negocio demuestran impacto

positivo con expansión geográfica de mercado, incremento de ticket promedio y reducción de tiempos operativos.

El proyecto valida viabilidad técnica y comercial de transformación digital para pequeños productores agrícolas, sugiriendo que inversión relativamente modesta en infraestructura digital puede generar retornos significativos. La metodología de desarrollo aplicada balanceó apropiadamente agilidad con rigor técnico, permitiendo entregas incrementales mientras se mantenía arquitectura coherente. La documentación exhaustiva generada facilita evolución futura del sistema por otros desarrolladores.

Las lecciones aprendidas incluyen importancia de priorizar pruebas end-to-end que validan flujos completos sobre pruebas unitarias que verifican componentes aislados, beneficios de adopción temprana de TypeScript que previene categorías enteras de errores en lugar de añadirlo retrofíticamente, y valor de observabilidad proactiva mediante logging estructurado y alertas configuradas antes de incidentes en lugar de reactivamente.