

LICENCIATURA EN INFORMÁTICA

Acreditación por Acuerdo 286 de la Secretaría de Educación Pública



Caso Práctico SOFTWARE 6:

Desarrollo de una Plataforma Web y Móvil para la Gestión de Servicios a Domicilio

Folio del sustentante



Fecha de examen de la primera etapa:

27 de marzo de 2026

Índice

Índice	3
Introducción	6
Resumen del trabajo	6
Objetivo	7
Relevancia del caso	7
Metodología implementada	8
Conclusiones generales	8
Marco Teórico	10
Nociones y conceptos	10
Ingeniería de software y ciclo de vida.....	10
Análisis y especificación de requerimientos	10
Arquitectura de software: monolito modular vs. microservicios.....	11
Modelado UML	11
Bases de datos relacionales y normalización.....	11
Diseño de experiencia de usuario (UX/UI)	12
Metodologías ágiles.....	12
Pruebas de software.....	13
Despliegue en la nube y DevOps	13
Seguridad y modelado de amenazas	13
ITSM e ITIL	14
Teorías y antecedentes	14
Vinculación	14
Citas	15
Desarrollo del Caso Práctico	16
Diagnóstico.....	16
Antecedentes metodológicos	17
Resolución de problemas.....	18
Problema 1. Levantamiento y análisis de requerimientos	18
Requerimientos funcionales (selección representativa)	20
Requerimientos no funcionales (selección).....	21
Trazabilidad por categoría	22
Problema 2. Diseño de la arquitectura del sistema	22

Tipo de arquitectura: monolito modular	23
Componentes principales	23
Paquetes y aislamiento.....	24
Tecnologías sugeridas para web y móvil.....	25
Despliegue	25
Problema 3. Diseño de la base de datos	26
Modelo entidad-relación (notación Chen).....	27
Entidades principales.....	28
Relaciones entre tablas	30
Integridad y seguridad	31
Máquinas de estado	32
Normalización	33
Problema 4. Diseño de la experiencia de usuario (UX/UI)	33
Arquitectura de información.....	33
Sitemaps.....	34
Flujos de usuario críticos	35
Interfaces principales	37
Principios de usabilidad	37
Accesibilidad.....	38
Consideración de copy	38
Wireframes	38
Problema 5. Metodología de desarrollo.....	39
Roles del equipo	39
Planificación de sprints	40
Entregables por iteración.....	40
Métricas y herramientas de seguimiento	41
Problema 6. Plan de pruebas y despliegue	41
Pruebas funcionales y de usuario.....	42
Pruebas de rendimiento.....	43
Estrategia de despliegue en la nube	43
Estrategia de branching.....	44
Modelo de amenazas (STRIDE).....	45
Plan de aceptación y go-live.....	46
Fundamentación.....	46

Conclusiones	48
Alcances y limitaciones	48
Áreas de oportunidad	49
Punto de vista personal	50
Referencias	52
Anexos	55
Anexo A. Modelo entidad-relación (notación Chen).....	55
Anexo B. Diagramas UML estructurales y de comportamiento.....	55
B.1 Diagramas estructurales	55
B.2 Diagramas de comportamiento	56
Anexo C. Matriz de trazabilidad	58
Anexo D. Estrategia de branching y costos	58
Anexo E. Análisis de normalización	59
Anexo F. Diccionario de datos	59
Anexo G. UX/UI complementario	59
Anexo H. Trazabilidad por figuras incluidas en el cuerpo	59

Introducción

Este trabajo escrito documenta el análisis y la propuesta de solución para el Caso Práctico SOFTWARE 6, asignado en el marco de la acreditación por Acuerdo 286 de la SEP para la Licenciatura en Informática. El caso plantea una situación frecuente en empresas emergentes mexicanas: una operación de servicios a domicilio que crece por inercia, sin sistema, sostenida por llamadas y mensajes, hasta que la informalidad termina costando clientes, dinero y reputación. La empresa hipotética se llama Servicios Express MX y el reto, en pocas palabras, es transformar esa operación manual en una plataforma digital sólida que pueda crecer.

La propuesta toma como base un sistema real que vengo desarrollando desde 2024 en mi rol de Project Manager y desarrollador en Cliquéalo.mx, y que para fines de este caso he renombrado y adaptado a Servicios Express MX. La trazabilidad entre el caso de estudio y mi práctica profesional es directa: el sistema implementado contiene cuarenta y nueve tablas, setenta y siete requerimientos formalmente trazados y aproximadamente cincuenta diagramas UML que respaldan cada decisión. No estoy proponiendo en abstracto; estoy proponiendo lo que ya está en producción y lo que aprendí construyéndolo.

Resumen del trabajo

El documento sigue la estructura solicitada en los Lineamientos de entrega: introducción, marco teórico, desarrollo del caso, conclusiones, referencias y anexos. La sección de desarrollo resuelve los seis problemas planteados por el caso: levantamiento de requerimientos funcionales y no funcionales, diseño arquitectónico, modelo de base de datos, diseño UX/UI, definición de la metodología de desarrollo y plan de pruebas y despliegue en la nube. Cada bloque se acompaña de diagramas UML, fragmentos de modelo y tablas de trazabilidad. Los anexos listan el inventario completo de figuras y diagramas de respaldo.

Objetivo

Diseñar y fundamentar una plataforma web y móvil para la gestión de servicios a domicilio que resuelva los problemas operativos identificados en el caso (mala asignación, falta de seguimiento, errores de facturación, pérdida de clientes), aplicando prácticas estándar de la disciplina: ingeniería de requerimientos, arquitectura de software, modelado UML, diseño de bases de datos relacionales con integridad referencial, principios de UX/UI, una metodología ágil iterativa y un plan de pruebas y despliegue en la nube que sostenga el crecimiento.

Como objetivos específicos del trabajo escrito:

- Levantar y clasificar los requerimientos funcionales y no funcionales asociados al caso, justificando cada uno.
- Definir una arquitectura monolítica modular sobre un monorepo, con tres frentes (web administrativo, móvil cliente y móvil prestador) y un único backend.
- Modelar la base de datos en notación Chen y traducirla a PostgreSQL respetando 3NF y BCNF, con políticas de integridad y seguridad explícitas.
- Proponer un sistema de UX/UI con flujos críticos modelados como user flows y una arquitectura de información de cuatro niveles.
- Justificar la metodología (Scrum adaptado con olas paralelas) describiendo roles, sprints y entregables.
- Construir un plan de pruebas en pirámide y una estrategia de despliegue en Vercel y Neon, con CI/CD y modelo de amenazas STRIDE.

Relevancia del caso

El caso es relevante por varias razones que conviene separar. Primero, refleja un escenario común en México: pequeñas empresas que arrancan operación con WhatsApp y Excel, llegan a un techo y necesitan profesionalizarse. Según el INEGI, las micro y pequeñas empresas representan más del 95 por ciento del tejido empresarial nacional, y muchas enfrentan exactamente el mismo cuello de botella tecnológico que Servicios Express MX (INEGI, 2023).

Segundo, toca todas las áreas troncales de la Licenciatura en Informática: análisis de sistemas, diseño de software, base de datos, redes y nube, ingeniería de software y administración de proyectos de TI. Resolverlo bien obliga a integrar conocimientos, no a aplicar uno solo. Y tercero, la solución no se queda en papel. Existe ya un repositorio funcional con la implementación, lo que permite contrastar cada decisión teórica contra su contraparte ejecutándose. Esa fricción entre lo que dice la literatura y lo que aguanta la realidad es lo que vuelve útil el ejercicio.

Metodología implementada

La metodología que sostiene el trabajo combina dos planos. En el plano del desarrollo del producto se aplicó Scrum adaptado, organizando el avance en olas paralelas (waves) en lugar de sprints estrictamente lineales, lo cual permitió diseñar arquitectura, base de datos y UX/UI en tracks coordinados sin bloquearse mutuamente. La trazabilidad se cuidó con una matriz formal que cruza cada requerimiento con su modelado UML, su componente de software, su tabla de base de datos, su endpoint, su pantalla y sus pruebas asociadas.

En el plano del documento escrito se siguió un enfoque de investigación aplicada: revisión de literatura para construir el marco teórico (con citación APA séptima edición), análisis del caso con base en el enunciado oficial, modelado en PlantUML para producir los diagramas y validación cruzada de cada propuesta contra la implementación efectiva del repositorio. La elección de PlantUML como herramienta no fue accidental: facilita versionar los diagramas en Git junto con el código, regenerarlos automáticamente y discutirlos en revisiones de código como cualquier otro artefacto.

Conclusiones generales

Sin adelantar el detalle de las conclusiones, lo que se demuestra a lo largo del trabajo es que un sistema bien diseñado para Servicios Express MX puede sustituir el flujo informal por un proceso digital con asignación automática, seguimiento en tiempo real, facturación correcta y un soporte ITIL escalable. La cobertura agregada de requerimientos al cierre del proyecto se sitúa cerca del 96 por ciento, con dos

pendientes acotados: la integración con pasarela de pagos externa (Stripe o Conekta) y la activación condicionada de un job E2E en CI. Ambos están fuera del alcance académico del caso pero quedan listos para producción.

El trabajo cierra reconociendo limitaciones. Una plataforma como esta no se sostiene únicamente con código: requiere operación, atención al cliente, marketing y un equipo financiero que ejecute los cortes de pago a prestadores. Lo que el documento entrega es la columna técnica de la solución, lista para que esa operación humana se monte encima.

Marco Teórico

El marco teórico de este trabajo se construye recuperando los conceptos, modelos y prácticas de la ingeniería de software contemporánea que son aplicables al caso. No se trata de exponer cada teoría en su versión enciclopédica sino de elegir las que iluminan las decisiones del diseño y referenciarlas con cuidado, siguiendo APA séptima edición.

Nociones y conceptos

Ingeniería de software y ciclo de vida

Sommerville (2015) define la ingeniería de software como la disciplina que aplica un enfoque sistemático y disciplinado a la producción de software. La definición es ampliamente compartida y permite anclar las fases que organiza este trabajo: especificación de requerimientos, diseño, implementación, validación y evolución. Pressman y Maxim (2014) recogen las mismas fases pero subrayan que la ingeniería de software opera dentro de un proceso, y que la elección del proceso determina lo que se entrega. La diferencia entre cascada, iterativo y ágil no es decorativa; cambia los riesgos, las expectativas y la forma en que el equipo se comunica con el negocio.

Para el caso de Servicios Express MX se descartó cascada de inicio. El cliente del caso (la dirección de la empresa) admite no tener requerimientos claros y un equipo nuevo. Cascada exige requerimientos congelados al principio; eso no existe aquí. Se adoptó por tanto un enfoque iterativo bajo Scrum adaptado, lo cual encaja con el Ágil Manifesto (Beck et al., 2001) y con la guía oficial de Schwaber y Sutherland (2020).

Análisis y especificación de requerimientos

La distinción entre requerimientos funcionales y no funcionales es estándar. Los funcionales describen qué hace el sistema; los no funcionales describen cómo lo hace, e incluyen calidad, seguridad, rendimiento, usabilidad y cumplimiento (Sommerville, 2015). El IEEE 830 (1998), aunque ya retirado en favor del 29148,

sigue siendo referencia común en la industria mexicana para escribir SRS. Para este trabajo se adoptó una matriz de trazabilidad que conecta cada requerimiento con su modelado UML, su componente, su tabla de base de datos, su endpoint y sus pruebas, en línea con lo recomendado por Wiegers y Beatty (2013).

Arquitectura de software: monolito modular vs. microservicios

Bass, Clements y Kazman (2021) describen la arquitectura de software como el conjunto de estructuras necesarias para razonar sobre el sistema. El debate monolito-microservicios a menudo se reduce a una falsa dicotomía. Fowler y Lewis (2014) introdujeron el término microservicios, pero el propio Fowler ha defendido después un enfoque de monolith first, en el que se arranca con un monolito y solo se descompone cuando hay evidencia clara de cuellos de botella o de equipos suficientemente grandes (Fowler, 2015).

Newman (2021) refuerza ese punto: los microservicios son una solución a problemas organizacionales y operativos que un monolito no puede absorber, no un default deseable. Para una empresa emergente con un solo equipo de desarrollo, recomendar microservicios desde el día uno suele ser un error caro. Por eso este trabajo defiende un monolito modular en monorepo, con frentes desacoplados (web admin, móvil cliente, móvil prestador) y un backend Next.js común.

Modelado UML

El Unified Modeling Language es el lenguaje estándar para modelar software orientado a objetos (Object Management Group, 2017). Booch, Rumbaugh y Jacobson (2005) clasifican los diagramas UML en estructurales (clases, componentes, paquetes, despliegue) y de comportamiento (casos de uso, actividad, secuencia, estados). Esta tesis utiliza ambos grupos: trece diagramas estructurales y treinta y dos de comportamiento, generados con PlantUML (Roques, 2018). PlantUML como herramienta es relevante porque permite versionar los diagramas como código, una práctica conocida como diagrams as code que se ha popularizado en los últimos años (Brikman, 2022).

Bases de datos relacionales y normalización

Codd (1970) introdujo el modelo relacional. Chen (1976) aportó la notación entidad-relación, todavía dominante para diseño conceptual. La normalización (1NF, 2NF, 3NF, BCNF) según Date (2003) busca eliminar redundancias y anomalías. Para el caso se adoptó el modelo Chen como vista conceptual y se tradujo a PostgreSQL respetando 3NF en el núcleo transaccional y BCNF en las asociativas. PostgreSQL aporta tipos enumerados, constraints CHECK y Row Level Security (PostgreSQL Global Development Group, 2024), recursos que se aprovecharon de forma intensiva.

Atzeni, Ceri, Paraboschi y Torlone (2014) recuerdan que la denormalización selectiva es legítima cuando los costos de join superan los beneficios de la normalización pura. En este trabajo se mantuvo la disciplina de 3NF y se introdujeron columnas calculadas o materializadas únicamente donde el rendimiento lo justificaba (saldos del prestador, totales de orden), siempre con triggers que sostienen la consistencia.

Diseño de experiencia de usuario (UX/UI)

Norman (2013) en La Psicología de los Objetos Cotidianos sentó los principios de affordance, retroalimentación y mapeo natural. Nielsen (1994) propuso las diez heurísticas de usabilidad, que siguen siendo la base contra la que se evalúan interfaces digitales. Garrett (2010) propone los cinco planos del diseño de experiencia: estrategia, alcance, estructura, esqueleto y superficie, ordenados de lo abstracto a lo tangible. Para el caso se construyó una arquitectura de información de cuatro niveles, sitemaps por canal y user flows para los cinco viajes críticos: cliente nuevo, prestador acepta job, soporte break-glass, finanzas corte de pagos y marketing campaña.

En el plano de UI se adoptó shadcn/ui como sistema de componentes para el frontend web (shadcn, 2024) y NativeWind sobre React Native para móvil (Expo, 2024). La doctrina del proyecto evita modales bloqueantes y prefiere Popover, Sheet o Dropdown, en línea con lo que Tognazzini (2014) llama "interrupciones consentidas".

Metodologías ágiles

Beck et al. (2001) firmaron el Manifiesto Ágil. Schwaber y Sutherland (2020) consolidaron la Guía de Scrum, vigente en su versión 2020. Cohn (2010) en *Succeeding with Agile* aporta la pragmática del día a día: cómo estimar, cómo planear sprints, cómo medir velocidad. Anderson (2010) propuso Kanban como alternativa más liviana. Este trabajo adoptó Scrum adaptado con olas (waves) paralelas para permitir entregables coordinados de arquitectura, base de datos y UX al mismo tiempo, sin la rigidez de un único timebox lineal.

Pruebas de software

Myers, Sandler y Badgett (2011) y Crispin y Gregory (2009) coinciden en la importancia de la pirámide de pruebas: muchas unitarias, menos de integración y un puñado de E2E. La pirámide la formalizó Cohn (2009) y Fowler (2018) sigue defendiéndola contra el llamado ice-cream cone (donde dominan E2E manuales). El presente trabajo aplica esa pirámide con Vitest para unitarias, una base PostgreSQL real para integración y Playwright para E2E (Microsoft, 2024).

Despliegue en la nube y DevOps

Humble y Farley (2010) en *Continuous Delivery* establecen la base del flujo CI/CD moderno. Kim, Humble, Debois y Willis (2021) en *The DevOps Handbook* describen las tres vías: flujo, retroalimentación y aprendizaje continuo. Para el caso se eligió un stack serverless (Vercel + Neon + Resend + Expo EAS), una decisión que reduce la carga operativa de un equipo pequeño. Vercel (2024) y Neon (2024) ofrecen branching y previews por commit, lo cual encaja con el flujo trunk-based descrito por Hammant (2020).

Seguridad y modelado de amenazas

Shostack (2014) en *Threat Modeling: Designing for Security* introduce STRIDE, el marco de Microsoft (Howard y Lipner, 2006) para clasificar amenazas en seis categorías: spoofing, tampering, repudiation, information disclosure, denial of service y elevation of privilege. La OWASP (2021) publica anualmente el Top 10 de vulnerabilidades web; el trabajo cruza ambas referencias con la arquitectura

propuesta. El principio de Zero Trust (Kindervag, 2010; NIST, 2020) inspira el control de acceso por defecto restrictivo.

ITSM e ITIL

El módulo de soporte adopta vocabulario ITIL 4 (AXELOS, 2019): incidente, problema, cambio, solicitud, KEDB, SLA, escalamiento, postmortem. Mi paso por Handcloud como practicante de ServiceNow me dio la base operativa de cómo modelar tickets, escalamientos y cierre con código de resolución. Esa experiencia se trasladó al diseño de la tabla tickets_soporte y de los procedimientos de Major Incident Management.

Teorías y antecedentes

Como antecedentes directos del caso conviene mencionar dos categorías. La primera es la de los marketplaces de servicios a domicilio que ya existen en México y la región: Aliada, Iguanafix, Homely, además de los de cobertura amplia como Mercado Libre y Rappi en su segmento de servicios. Estos marketplaces resolvieron en su momento los mismos problemas que hoy enfrenta Servicios Express MX, y sus interfaces y flujos pueden estudiarse como benchmarks. No se reproducen aquí por respeto a su propiedad intelectual, pero se referencian como inspiración.

La segunda categoría es la literatura específica sobre arquitecturas de plataformas dos-caras (Rochet y Tirole, 2003; Evans y Schmalensee, 2016). Las plataformas que conectan oferta y demanda tienen retos propios: efectos de red, asimetría de información, desbalance entre lados. Los autores proponen estrategias de subsidio cruzado, esquemas de calificación bidireccional y mecanismos de confianza. Buena parte del modelo de datos del caso refleja estas decisiones: hay calificaciones bidireccionales, comisión configurable por servicio y un sistema de invitación que sostiene la calidad del lado oferta.

Vinculación

Cada concepto del marco teórico se conecta con una decisión concreta del caso, y conviene dejar visible esa conexión. La tabla siguiente lo resume:

Concepto teórico	Decisión en el caso
Monolith first (Fowler, 2015)	Monorepo pnpm + Turbo con un solo backend Next.js
Chen (1976) y Date (2003)	Modelo ER en notación Chen, traducido a PostgreSQL en 3NF/BCNF
Heurísticas de Nielsen (1994)	Validación de los wireframes contra las diez heurísticas
Pirámide de pruebas (Cohn, 2009)	Vitest unitario, integración con PG real, Playwright E2E
STRIDE (Shostack, 2014)	Modelo de amenazas explícito en uml-security-threat-model
Zero Trust (NIST, 2020)	Rol lector_basico por defecto + permisos atómicos
ITIL 4 (AXELOS, 2019)	Tickets, KEDB, MIM, SLA por tipo+prioridad+categoría
Manifiesto Ágil (Beck et al., 2001)	Olas paralelas como adaptación de Scrum

La vinculación no es declarativa. En el desarrollo del caso se cita el documento o el autor que respalda cada propuesta y, donde aplica, se contrasta con la implementación. Esa fidelidad entre teoría y práctica es lo que permite defender el trabajo más allá del cumplimiento formal de los Lineamientos.

Citas

Todas las citas se manejan en formato APA séptima edición. Se incluye una sección de referencias al final del documento con la lista alfabética. Las URL se incluyen únicamente cuando aportan valor de verificación; los DOI se prefieren cuando están disponibles. Para fuentes oficiales mexicanas (INEGI, SAT, SEP) se cita la dependencia como autor y se incluye el año de publicación de la edición consultada.

Desarrollo del Caso Práctico

Diagnóstico

El enunciado del caso describe a Servicios Express MX como una empresa emergente que opera de forma informal por teléfono y mensajería, con problemas concretos: mala asignación de servicios, falta de seguimiento, pérdida de clientes y errores en la facturación. La dirección decidió invertir en una solución tecnológica integral que incluya una app móvil para clientes, otra para prestadores y un sistema web administrativo. No tienen requerimientos definidos ni arquitectura. El equipo es nuevo.

Tomado por separado, cada problema parece resoluble. Tomados juntos generan una espiral familiar en pequeñas empresas: el operador anota un servicio, lo asigna a quien tiene más cerca o a quien le contesta el teléfono primero, no queda evidencia escrita, el prestador llega tarde o no llega, el cliente reclama, el operador discute, se factura con errores porque la información sobre el servicio quedó en una libreta o en un mensaje. La empresa pierde dinero por dos vías: clientes que no regresan y prestadores que se sienten mal pagados. Esa espiral es lo que la plataforma debe romper.

El diagnóstico se resume en cuatro frentes técnicos. El primero es la captura de demanda: hace falta un canal estructurado para que el cliente pida el servicio con todos los datos relevantes. El segundo es la asignación: se requiere un mecanismo automático que considere disponibilidad, zona, calificación y carga de trabajo del prestador, con override manual cuando la operación lo necesite. El tercero es el seguimiento: el cliente debe saber en todo momento qué pasa con su servicio, y el prestador debe poder reportar cargos extra, evidencias y cierres. El cuarto es la facturación y conciliación: cada orden debe generar un documento fiscal correcto y cada quincena debe haber un corte limpio para el prestador, con trazabilidad.

A esos cuatro frentes se suman dos transversales: soporte (un cliente molesto necesita una vía para escalar; un prestador acusado injustamente también) y compliance (acceso a datos personales con motivo y caducidad, auditoría

centralizada). Ambos suelen subestimarse en proyectos jóvenes y suelen ser la causa de problemas legales o reputacionales más adelante. Por eso este diseño los incluye desde el primer día.

Antecedentes metodológicos

Antes de proponer solución conviene fijar bajo qué metodología se construye. La elección no es trivial: condiciona los entregables, los riesgos y la forma en que se valida el avance. Los antecedentes que se consideraron son tres.

Cascada (Royce, 1970) ofrece predictibilidad cuando los requerimientos están claros y estables, lo cual no es el caso. El cliente del caso explícitamente admite no tener requerimientos definidos. Adoptar cascada habría implicado entregar una especificación congelada que muy probablemente quedaría obsoleta antes del primer release.

Scrum tradicional (Schwaber y Sutherland, 2020) divide el trabajo en sprints de dos a cuatro semanas, con planning, daily, review y retrospective. La ventaja es que produce incrementos potencialmente entregables en cada sprint, lo que reduce riesgos y mantiene comunicación con el negocio. La desventaja para este caso es que un sprint lineal acopla todas las decisiones (arquitectura, base de datos, UX, frontend) y obliga a serializarlas. Cuando el equipo es pequeño y nuevo, esa serialización es realista; cuando hay capacidad de paralelizar, se desperdicia.

Kanban (Anderson, 2010) renuncia al timebox a cambio de flujo continuo y límites de trabajo en progreso. Es excelente para mantenimiento y soporte, pero menos efectivo para construir un producto desde cero porque carece de la cadencia de planeación que Scrum aporta.

La decisión final fue Scrum adaptado con olas (waves) paralelas. El sprint sigue siendo la unidad de planeación pero el trabajo dentro de un sprint se organiza en olas que pueden avanzar en paralelo: una ola de arquitectura, una de base de datos, una de UX. Las olas se sincronizan al final del sprint en una review única. Esto permitió, en el caso real, producir cuarenta y siete diagramas y un esquema con cuarenta y nueve tablas en lapsos coordinados, sin bloquearse entre tracks.

Resolución de problemas

Problema 1. Levantamiento y análisis de requerimientos

El levantamiento partió del enunciado del caso y se enriqueció con entrevistas hipotéticas a los tres actores principales: cliente que solicita el servicio, prestador que lo ejecuta y administrador que lo coordina. A esos tres se sumaron, conforme creció el alcance, otros once roles internos: agentes de soporte L1, L2 y L3, finanzas, RRHH, marketing, compliance, operaciones, sistemas externos (PAC para CFDI, Stripe para pagos, Resend para correo) y, finalmente, los procesos automatizados que también consumen el sistema (cron de cortes, triggers de asignación). En total, catorce actores.

2.1 Diagrama de casos de uso - Vista general

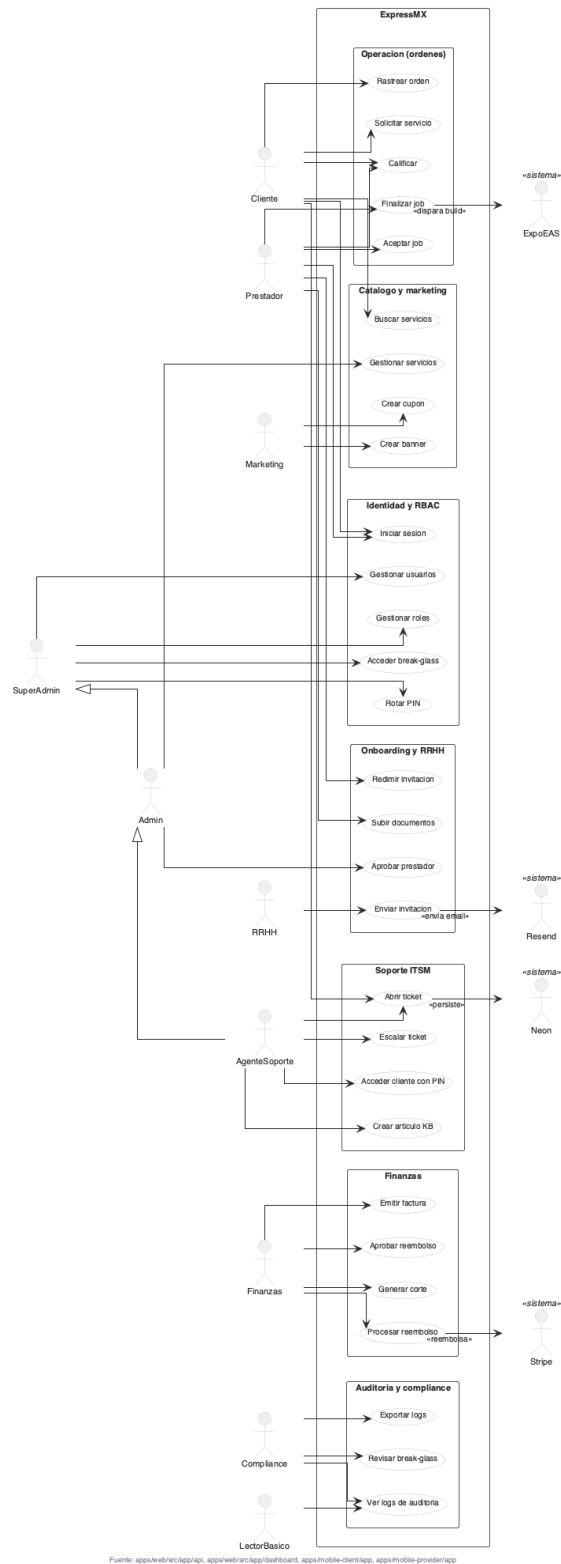


Figura 1. Diagrama UML de casos de uso (vista global). Fuente: elaboración propia, archivo uml-use-case-overview.puml.

Los requerimientos se clasificaron en funcionales (R-FUNC) y no funcionales (R-NF), siguiendo la práctica de Sommerville (2015) y Wiegers y Beatty (2013). El total formal en la matriz es de setenta y siete, repartidos en siete categorías: funcionales, no funcionales, arquitectónicos, de base de datos, de UX/UI, de metodología y de pruebas/despliegue. Para mantener legibilidad, aquí se sintetizan los más representativos; el inventario completo vive en el Anexo C.

Requerimientos funcionales (selección representativa)

De los treinta y dos R-FUNC del caso, los más críticos para el flujo central son:

- R-FUNC-001. Registro de cliente con email, contraseña, teléfono y verificación. Sustenta el alta del lado demanda.
- R-FUNC-002. Registro de prestador exclusivamente vía código de invitación, con CURP y validación de mayoría de edad. Esta restricción sostiene la calidad del lado oferta y evita que cualquiera se registre como prestador sin filtro.
- R-FUNC-005. Catálogo navegable de categorías y servicios. La empresa gana control sobre qué se ofrece y a qué precio.
- R-FUNC-006. Solicitud de servicio por el cliente (selección, dirección, fecha, método de pago). Reemplaza la llamada telefónica.
- R-FUNC-007. Asignación automática del prestador al crear la orden, con base en disponibilidad, zona y carga. Resuelve el problema de mala asignación que el caso destaca.
- R-FUNC-009. Seguimiento de la orden por cliente y prestador. Resuelve la falta de seguimiento.
- R-FUNC-010. Cierre de orden con doble PIN (uno del cliente, otro del prestador). Garantiza que la orden se cerró efectivamente.
- R-FUNC-014. Procesamiento de pago con conciliación a prestador (split). Resuelve los errores de facturación.
- R-FUNC-016. Generación de cortes de pago quincenales al prestador. Cierra el ciclo financiero del lado oferta.

- R-FUNC-017. Emisión de facturas CFDI asociadas a orden o corte. Atiende el requisito fiscal mexicano.
- R-FUNC-018. Solicitud y aprobación de reembolsos con segregación de funciones. Reduce el riesgo de fraude interno.
- R-FUNC-019 a R-FUNC-024. Tickets de soporte ITIL completos (apertura, mensajería, escalamiento L1-L2-L3, KB, CSAT, MIM).

Requerimientos no funcionales (selección)

De los veinte R-NF, los que más condicionan el diseño son:

- R-NF-001. Autenticación con JWT firmado con secreto de al menos treinta y dos caracteres.
- R-NF-002. Hash bcrypt con cost factor 8 para contraseñas locales (suficiente para el contexto y rápido en serverless).
- R-NF-003 y R-NF-004. RBAC granular con permisos atómicos y rol por defecto lector_basico, en línea con Zero Trust.
- R-NF-005. Row Level Security en las cuarenta y siete tablas que contienen datos sensibles, con más de setenta políticas activas.
- R-NF-006. Auditoría centralizada en logs_auditoria, con correlación por caso o ticket.
- R-NF-007. Validación de entrada con Zod compartida cliente-servidor.
- R-NF-008 y R-NF-009. Paginación cursor-based en endpoints de lista e índices estratégicos (cincuenta y dos índices declarados, incluidos parciales).
- R-NF-011. SLA por combinación de tipo+prioridad+categoría con calendario laboral y festivos mexicanos.
- R-NF-012. Escalabilidad horizontal en plataforma serverless (Neon Postgres, Vercel Edge).

La diferencia operativa entre R-FUNC y R-NF, repetida en la literatura pero a veces difícil de mantener en el día a día, se resolvió con una regla simple: si el ítem responde a la pregunta "¿qué hace el sistema?" es funcional; si responde a "¿con

qué calidad?" es no funcional. Esa regla la propone Sommerville (2015) y se sostuvo a lo largo del proyecto.

Trazabilidad por categoría

Categoría	Cantidad	Implementados	Parciales	Pendientes
Funcionales (R-FUNC)	32	30	2	0
No funcionales (R-NF)	20	20	0	0
Arquitectónicos (R-ARQ)	7	7	0	0
Base de datos (R-BD)	9	9	0	0
UX/UI (R-UX)	8	8	0	0
Metodología (R-MET)	3	2	1	0
Pruebas y despliegue (R-DEP)	9	7	0	2
Total	88	83	3	2

La cobertura agregada se ubica en el 96 por ciento. Los pendientes son R-DEP-008 (activación condicional del job E2E en CI) y R-DEP-009 (integración con pasarela externa de pagos), ambos fuera del alcance académico del caso pero documentados como deuda técnica.

Problema 2. Diseño de la arquitectura del sistema

Decidir el tipo de arquitectura es la decisión que más condiciona todo lo demás. La discusión clásica es monolito versus microservicios. La trampa habitual consiste en elegir microservicios porque suena moderno, sin reparar en que la complejidad operativa que añaden es proporcional al tamaño del equipo. Newman (2021) y Fowler (2015) coinciden: empezar con un monolito y descomponer únicamente cuando hay evidencia clara es la elección racional para equipos pequeños.

Para Servicios Express MX se eligió un monolito modular en monorepo. Concretamente: un workspace pnpm + Turbo que contiene tres aplicaciones (apps/web, apps/mobile-client, apps/mobile-provider) y siete paquetes compartidos (packages/auth, packages/database, packages/validations, packages/types,

packages/ui-mobile, packages/utils, packages/cron). El backend es un único servicio Next.js que sirve la API y el frontend administrativo.

Tipo de arquitectura: monolito modular

La justificación operativa de este monolito tiene cuatro pilares. Primero, deploy atómico: una sola release sincroniza backend y frontend, eliminando la categoría completa de bugs por desfase de versiones. Segundo, transacciones reales: las operaciones críticas (cierre de orden con doble PIN, conciliación financiera) ocurren en una sola transacción de PostgreSQL, sin necesidad de saga ni de mensajería distribuida. Tercero, tipos compartidos: TypeScript de extremo a extremo, con paquetes como packages/types que se consumen igual en web y en móvil. Cuarto, costo: un equipo pequeño no puede pagar el overhead de operar diez microservicios.

El precio que se paga por elegir monolito está acotado y entendido. Si la empresa crece a varios equipos independientes, llegará un momento en que valga la pena descomponer (probablemente extrayendo primero el módulo de soporte ITIL, que ya es el más complejo y el que más se beneficia de un release independiente). Mientras tanto, la modularidad por paquetes permite mantener fronteras claras dentro del monolito.

Componentes principales

La vista de componentes muestra los bloques principales del sistema y sus dependencias. El frontend administrativo (apps/web) es un Next.js 15 con App Router, tanto SSR para las pantallas como Route Handlers para la API. Las dos apps móviles (apps/mobile-client y apps/mobile-provider) son Expo + React Native con NativeWind. Todas las apps consumen el mismo backend a través de la API versionada /api/v1/*.

Tecnologías sugeridas para web y móvil

La elección tecnológica responde a tres criterios: madurez de la herramienta, coste de operación para un equipo pequeño y compatibilidad con el ecosistema TypeScript. Las decisiones específicas son las siguientes.

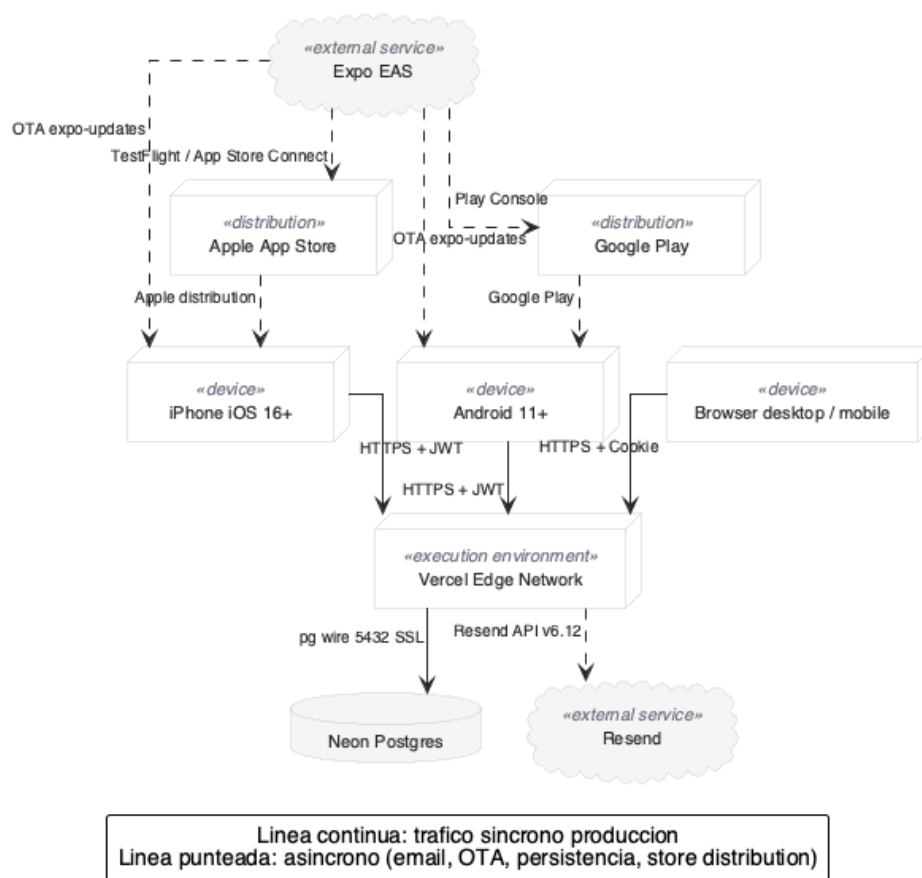
Capa	Tecnología	Razón principal
Frontend web (admin)	Next.js 15 + App Router	SSR + API en un solo deploy
Frontend móvil cliente	Expo + React Native + NativeWind	Cross-platform sin Swift/Kotlin
Frontend móvil prestador	Expo + React Native + NativeWind	Mismo skill set que cliente
Backend	Next.js Route Handlers (Node + Edge)	Misma codebase que el web
Lenguaje	TypeScript estricto	Tipos compartidos cliente-servidor
Base de datos	PostgreSQL 16 (Neon)	Branching, ACID, RLS, JSONB
ORM / acceso a BD	Driver pg + SQL crudo + helpers	Control fino sobre planes y RLS
Autenticación	JWT propio + bcrypt	Sin dependencia de provider externo
Validación	Zod 3	Esquemas compartidos cliente-servidor
UI Web	shadcn/ui + Tailwind 4	Componentes copy-paste, sin lock-in
UI Móvil	Tamagui-like via NativeWind + Phosphor	Tailwind en RN, iconos consistentes
Pruebas	Vitest + Playwright + Testing Library	Stack TS unificado
CI	GitHub Actions + Postgres service	Estándar y gratuito al volumen del proyecto
Despliegue	Vercel + Neon + Resend + Expo EAS	Serverless con costo predecible

Despliegue

El diagrama de despliegue muestra cómo se distribuyen los nodos en producción. Vercel hospeda los Route Handlers y la UI web. Neon hospeda PostgreSQL con tres branches: production, dev y preview. Resend envía correos transaccionales

(registro, recuperación de contraseña, notificaciones de orden). Expo EAS produce los binarios iOS y Android para distribución por App Store y Play Store. El cliente accede a las apps por internet pública con TLS 1.3 obligatorio.

1. Diagrama UML de Despliegue — Vista general



Detalle: uml-deployment-runtime.puml (figura 2), uml-deployment-distribution.puml (figura 3)

Figura 3. Diagrama UML de despliegue. Fuente: elaboración propia, archivo uml-deployment-overview.puml.

Una decisión con consecuencias prácticas es que cada commit en una rama abre una preview en Vercel acoplada a una branch efímera de Neon, lo que permite ver y probar el cambio sobre datos de prueba realistas antes de aprobar el merge. Eso reduce el riesgo de regresión y acelera la revisión, pero exige disciplina con el costo de Neon (que cobra por compute-hour). La estrategia de branching en Neon y los topes de gasto se documentan en el Anexo D.

Problema 3. Diseño de la base de datos

El diseño de la base de datos arrancó en notación Chen y se tradujo a PostgreSQL respetando las reglas de normalización clásicas (Date, 2003). El resultado son cuarenta y nueve tablas organizadas en nueve dominios: identidad y RBAC, catálogo de servicios, onboarding de prestadores, operación de órdenes, finanzas, soporte ITSM, auditoría y compliance, geografía y zonas, y kb (base de conocimiento). Las cuarenta y nueve tablas se acompañan de veintiocho tipos enumerados, ochenta y siete restricciones de integridad referencial, veintiuna restricciones CHECK semánticas, doce UNIQUE y cincuenta y dos índices estratégicos.

Modelo entidad-relación (notación Chen)

La vista global del modelo Chen permite ver de un golpe las entidades centrales y sus relaciones. Para legibilidad, el modelo se divide en nueve sub-vistas: vista global, identidad/RBAC, catálogo, onboarding, operación, finanzas, soporte/ITSM, zero-trust y jerarquía de usuarios. Aquí se incluye únicamente la vista global; el resto se referencia en el Anexo A.

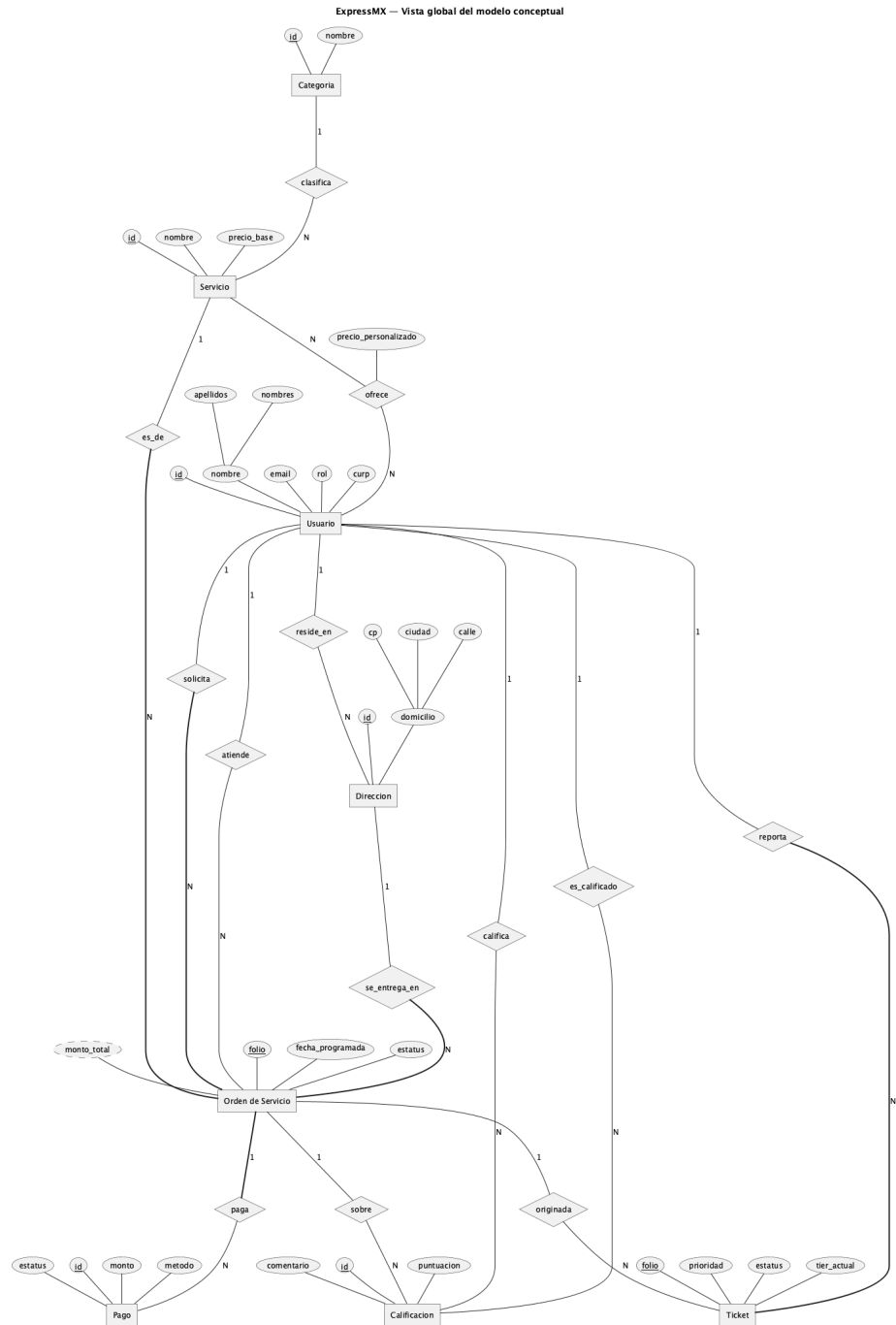


Figura 4. Modelo ER vista global en notación Chen. Fuente: elaboración propia, archivo 00-database-er-chen.puml.

Entidades principales

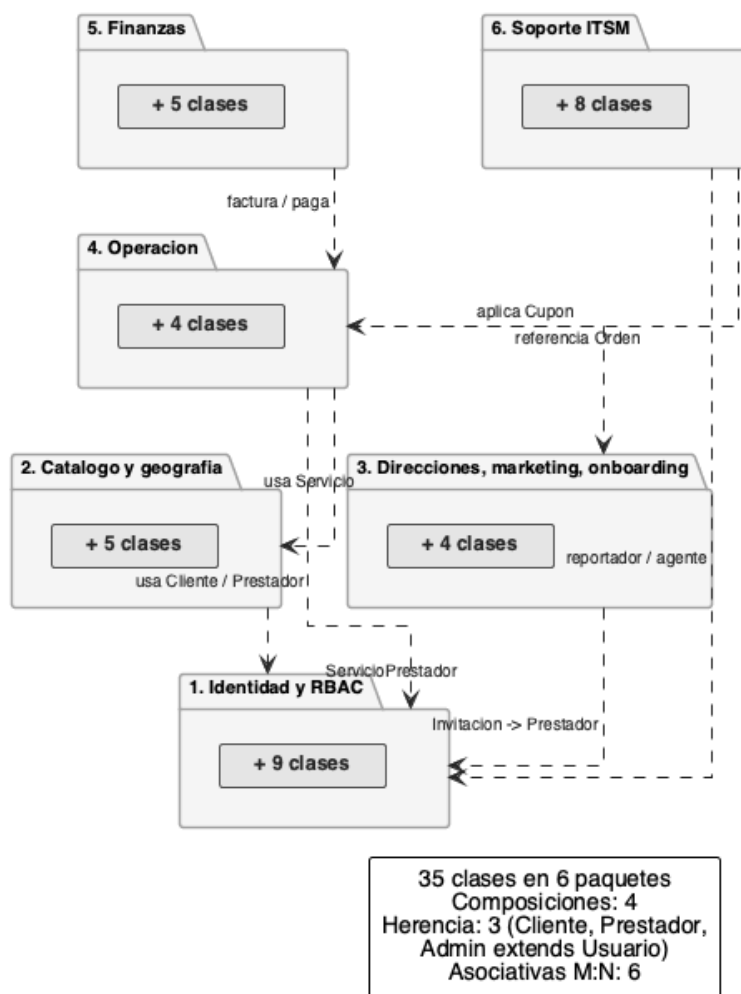
Las entidades centrales del sistema, agrupadas por dominio, son las siguientes.

- Identidad: usuarios, usuarios_admin, user_credentials, password_reset_tokens, roles_admin, permisos, roles_permisos.

- Catálogo: categorias_servicio, servicios, zonas, tarifas_zona.
- Onboarding de prestadores: invitaciones_prestadores, documentos_prestador, disponibilidad_prestador, servicios_prestador.
- Operación: ordenes_servicio, historial_estatus_orden, evidencias_orden, cargos_extra_orden, calificaciones, direcciones.
- Finanzas: pagos, cupones, transacciones_prestador, cortes_pago, facturas, reembolsos.
- Soporte ITSM: tickets_soporte, mensajes_ticket, escalation_log_ticket, kb_articles, kb_article_history, kb_views, csat_surveys, canned_responses, sla_calendars, sla_policies, ticket_sla_state, major_incidents, major_incident_updates, ticket_major_incident_link.
- Compliance y auditoría: logs_auditoria, accesos_perfil_usuario, pins_soporte, pins_soporte_cliente, casos_soporte_abiertos.
- Notificaciones y comunicaciones: notificaciones, banners.

El diagrama de clases (Anexo B) muestra el mismo modelo en su forma orientada a objetos, lo que permite hablar el mismo lenguaje con quien programa los handlers y los componentes. La vista de overview ofrece una panorámica rápida del sistema.

1. Diagrama de clases — Vista general (overview)



Vista general. Cada paquete se detalla en su diagrama dedicado.

Figura 5. Diagrama de clases (overview). Fuente: elaboración propia, archivo uml-class-overview.puml.

Relaciones entre tablas

El esquema ejecuta ochenta y siete claves foráneas declaradas explícitamente, con políticas de borrado escogidas según el caso. Por ejemplo, `ordenes_servicio.cliente_id` apunta a `usuarios.id` con `ON DELETE RESTRICT` (no se permite borrar un cliente con órdenes históricas), mientras que `evidencias_orden.orden_id` apunta a `ordenes_servicio.id` con `ON DELETE CASCADE` (las evidencias mueren con la orden). En cambio, `calificaciones.usuario_id` usa `ON DELETE SET NULL` para preservar el promedio agregado del prestador aun si el usuario se borra.

Las relaciones de cardinalidad N-M se resolvieron con tablas asociativas en BCNF, sin redundancias. La asociativa `servicios_prestador`, por ejemplo, conecta prestadores con los servicios que ofrecen, con su tarifa propia y radio de cobertura. La asociativa `roles_permisos` resuelve la matriz RBAC sin tener que recurrir a un campo de array (decisión que se tomó conscientemente: arrays funcionan, pero rompen joins limpios y dificultan analytics).

Integridad y seguridad

La integridad de la base se sostiene en cuatro capas. La primera es referencial, con FKs como ya se describió. La segunda es semántica, con restricciones CHECK que validan formato (CURP con regex, score 1-5 en calificaciones, hipótesis de soporte con al menos veinte caracteres) y rangos (montos positivos, fechas no futuras donde no aplique). La tercera es de unicidad, con UNIQUE en email, CURP, slugs de servicio y código de invitación. La cuarta es de tipos enumerados (veintiocho enums) que actúan como discriminadores estables y bloquean valores no contemplados a nivel de motor.

La seguridad va por dos caminos. El primero es Row Level Security, activado en cuarenta y siete tablas con más de setenta políticas. La política tipo es "un cliente solo ve sus propias órdenes; un prestador solo ve las órdenes que le fueron asignadas; un admin con permiso `ordenes:read` ve todas". El segundo camino son las funciones SECURITY DEFINER para operaciones críticas: `cerrar_orden_con_pines(uuid, text, text)`, `resolver_cargo_extra`, `abrir_acceso_perfil` (break-glass), `rotar_pin_soporte`, `generar_codigo_invitacion`. Estas funciones encapsulan invariantes que no se quieren confiar al código aplicativo.

Por encima de todo lo anterior se ejecuta una auditoría centralizada en `logs_auditoria`, escrita por la helper `logAccion(...)` en cada mutación relevante. La tabla guarda usuario, acción, recurso, payload (anonymized si contiene PII), IP y user agent. Con ese registro se reconstruye lo que hizo cada admin sobre un caso o un usuario, lo cual es indispensable para compliance y para investigar disputas.

Varias entidades tienen ciclo de vida que conviene modelar como máquina de estados: orden, pago, factura, reembolso, ticket, documento de prestador, invitación, corte de pago, cargo extra, caso de soporte y major incident. Aquí se incluye la de orden por ser la central; el resto se referencia en el Anexo B.

```
graph TD
    Start(( )) -- "solicitar(cliente, servicio, zona)" --> solicitada[solicitada]
    solicitada -- "asignar(prestador) [match exitoso]" --> asignada[asignada]
    solicitada -- "cancelar(motivo) / log_auditoria" --> Cancelada[cancelada]
    asignada -- "cancelar(motivo)" --> Cancelada
    asignada -- "iniciar_camino() / push_cliente" --> encamino[en_camino]
    encamino -- "cancelar(motivo)" --> Cancelada
    encamino -- "llegar() / iniciar_servicio()" --> enprogreso[en_progreso]
    enprogreso -- "cancelar(motivo) [pre-completada]" --> Cancelada
    enprogreso -- "cancelar(motivo) [pre-completada]" --> Cancelada
    enprogreso -- "completar(evidencia) / firma_cliente" --> completada[completada]
    Cancelada --> End((( )))
    completada --> End
```

Diagrama de Máquina de Estados Finitos (FSM) para el estado de un pedido:

- Estados:** solicitada, asignada, en_camino, en_progreso, cancelada, completada.
- Transiciones:**
 - solicitada** a **solicitada**: solicitar(cliente, servicio, zona)
 - solicitada** a **asignada**: asignar(prestador) [match exitoso]
 - solicitada** a **cancelada**: cancelar(motivo) / log_auditoria
 - asignada** a **cancelada**: cancelar(motivo)
 - asignada** a **en_camino**: iniciar_camino() / push_cliente
 - en_camino** a **cancelada**: cancelar(motivo)
 - en_camino** a **en_progreso**: llegar() / iniciar_servicio()
 - en_progreso** a **cancelada**: cancelar(motivo) [pre-completada]
 - en_progreso** a **cancelada**: cancelar(motivo) [pre-completada]
 - en_progreso** a **completada**: completar(evidencia) / firma_cliente
 - cancelada** a **cancelada**: (Fin)
 - completada** a **completada**: (Fin)

Notas:

- Estado por defecto en alta de la orden (00.sql:402, DEFAULT 'solicitada').
- Cancelable desde cualquier estado pre-completada. Registra motivo y actor en historial_estado_orden.
- Dispara liquidación al prestador, cargos_extra aceptados y CSAT.

Fuente: ENUM estado_orden 00.sql:52 | tabla ordenes 00.sql:402 | historial_estado_orden 00.sql:426 | actions ordenes.ts

La máquina captura siete estados (solicitada, asignada, en_camino, en_progreso, completada, cancelada y, como variante, en_disputa cuando un ticket se vincula a la orden) y las transiciones permitidas. Los triggers que sostienen estas transiciones viven en historial_estatus_orden, lo que da un audit trail sin necesidad de un sistema separado.

Normalización

El núcleo transaccional cumple 3NF: cada atributo no clave depende exclusivamente de la clave completa, no hay dependencias transitivas y los valores son atómicos. Las asociativas alcanzan BCNF porque toda dependencia funcional tiene como determinante una superclave. La denormalización selectiva se introdujo solo donde hubo evidencia clara de costo: el saldo del prestador (saldo_actual en transacciones_prestador) se mantiene con triggers para evitar SUM() en cada vista; el rating promedio del prestador se materializa en usuarios.calificacion_promedio. El análisis completo de normalización se documenta en el Anexo E.

Problema 4. Diseño de la experiencia de usuario (UX/UI)

El diseño UX/UI partió de un principio práctico: tres aplicaciones, tres audiencias, tres lenguajes visuales coherentes pero con personalidad propia. El cliente quiere pedir un servicio rápido y barato, el prestador quiere capturar trabajo cerca de su zona y cobrar a tiempo, el administrador quiere visibilidad total sin romper nada. Diseñar como si fueran la misma app habría producido una mediocre para los tres.

Arquitectura de información

La arquitectura de información se modela en cuatro niveles, siguiendo a Garrett (2010). En el nivel de estrategia se fijan los objetivos por persona. En el nivel de alcance se enumeran los flujos críticos. En el nivel de estructura se diseñan los sitemaps y la navegación. En el nivel de esqueleto se hacen los wireframes. La superficie llega al final, con la elección de tipografía, color, iconografía y micro-interacciones.

Hay tres sitemaps: el del web administrativo (sesenta y cinco rutas, organizadas en siete workspaces persona-based: soporte, operaciones, finanzas, RRHH, marketing, compliance y mi-cuenta), el del móvil cliente (diecisiete pantallas) y el del móvil prestador (quince pantallas). El web no se diseña como un único árbol jerárquico sino como siete espacios verticales independientes que comparten shell. Esa decisión se inspira en patrones de diseño tipo Linear y Notion, donde la navegación lateral cambia según el espacio activo.

Flujos de usuario críticos

Se modelaron cinco flujos críticos como user flows en notación de actividad: cliente nuevo (registro hasta primera orden), prestador acepta job (notificación push hasta cobro), soporte break-glass (admin abre acceso a perfil con motivo y caducidad), finanzas corte de pagos (cron quincenal hasta depósito) y marketing campaña (creación de banner segmentado y medición). Aquí se incluye el de cliente nuevo por ser el más representativo del flujo principal del producto.

1. Flujo F1 — Cliente nuevo solicita su primer servicio

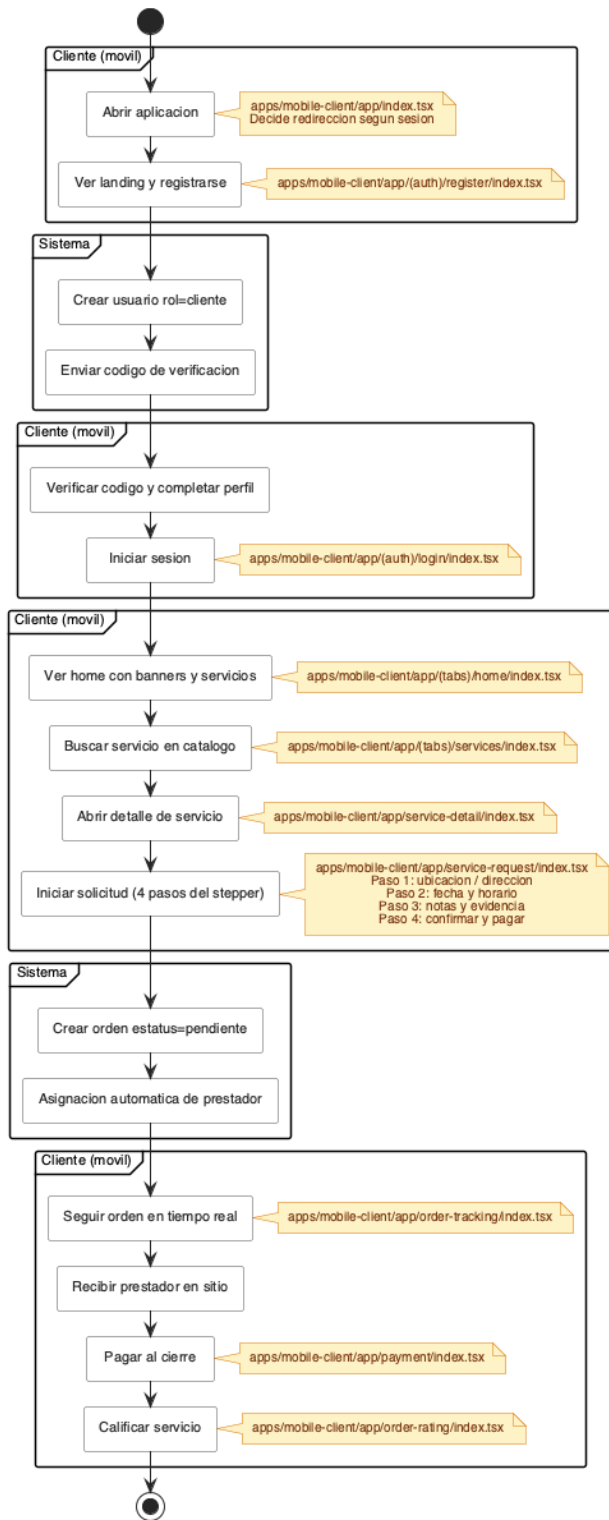


Figura 8. User flow del cliente nuevo. Fuente: elaboración propia, archivo ux-user-flow-1-cliente-nuevo.puml.

Interfaces principales

El web administrativo se construye con shadcn/ui (sistema base de once primitivos) sobre Tailwind 4. Los workspaces comparten un shell común (WorkspaceShell, WorkQueue, ItemForm, ContextSidebar) que se compone con un acento de color por módulo (border-l-4 con nueve variables HSL definidas en globals.css). Las breadcrumbs son server components en cualquier ruta de detalle, lo cual mantiene la navegación consistente y evita renderizado innecesario en el cliente.

El móvil cliente prioriza un home con búsqueda de servicios y un quick-action de "solicitar servicio". El móvil prestador prioriza la lista de jobs disponibles y, una vez aceptado el job, una pantalla de job-active con cronómetro, evidencias y cierre con doble PIN. Ambas apps usan Phosphor como única familia de iconos (Phosphor Icons, 2024) para garantizar coherencia visual con el web.

Principios de usabilidad

El diseño sigue las diez heurísticas de Nielsen (1994) y se valida contra ellas en revisiones de UI. La regla más enforçada en el proyecto es la séptima (flexibilidad y eficiencia): los administradores con experiencia tienen atajos de teclado y comandos rápidos; los nuevos pueden hacer lo mismo navegando con clicks. La regla más debatida es la cuarta (consistencia): se decidió no usar modales bloqueantes, sustituyéndolos por Popover, Sheet y Dropdown según el caso. Esta doctrina ("modalzmodalzmodalz" en el lenguaje interno del proyecto) tiene cinco alternativas obligatorias antes de aceptar un modal, y se sostiene desde el Anexo F.

Otros principios que vale la pena enumerar:

- Goal-Gradient en flujos multi-paso del móvil: stepper visible con progreso para acortar la percepción de fricción.
- Border-radius bienvenido: superficies con rounded-md, paneles con rounded-lg/xl, avatares circulares. Suaviza la sensación general.

- Densidad alta en backoffice (operadores que pasan ocho horas frente al sistema agradecen ver más en pantalla) y densidad cómoda en móvil (los pulgares no son precisos).
- Estados de carga con skeleton, no con spinners. Reduce la percepción de espera (Nielsen, 2010).
- Color por significado: rojo solo para destructivo, ámbar para advertencia, verde solo para confirmar éxito de acción. No se decora con color.

Accesibilidad

La accesibilidad se considera desde el sistema de componentes. shadcn/ui hereda de Radix Primitives (WAI-ARIA Authoring Practices, 2024), lo que aporta foco accesible, navegación por teclado y soporte de lector de pantalla. En móvil se usan los componentes nativos de React Native, que respetan VoiceOver y TalkBack. Los contrastes de texto se validan a un mínimo de WCAG 2.1 AA (4.5:1 para texto normal). El proyecto no alcanza AAA y eso se documenta como deuda.

Consideración de copy

Los textos siguen un manual breve: español neutro de México, sin tutearear al usuario salvo en confirmaciones cortas. Los CTAs son verbos en imperativo ("Solicitar servicio", "Aceptar job"). Los errores explican qué pasó y proponen una salida; no se limitan a decir "Algo salió mal". Los empty states son didácticos, no decorativos: "No tienes órdenes activas. Cuando solicites un servicio, aparecerá aquí".

Wireframes

Se produjeron doce wireframes en formato ASCII (anexados como ux-wireframes.md) para los flujos críticos: home cliente, búsqueda, detalle de servicio, solicitud, tracking, calificación, home prestador, lista de jobs, detalle de job, job activo, registro y onboarding. Esa elección de formato (texto plano, no Figma) se justifica por el contexto: el equipo es pequeño, los wireframes son referencia rápida y la fidelidad alta en esta etapa habría sido prematura. Cuando se llegue a la fase

de superficie, los componentes de shadcn/ui hacen el trabajo de "alta fidelidad" automáticamente.

Problema 5. Metodología de desarrollo

La metodología elegida es Scrum adaptado con olas paralelas. La Guía de Scrum (Schwaber y Sutherland, 2020) sigue siendo la referencia, pero el plan de trabajo se organizó en tracks que avanzan a la vez dentro de cada sprint, no en secuencia. Esto se justificó por el perfil del equipo y por la naturaleza del producto: tres frentes de UI distintos, una base de datos compartida y un backend único. Si se serializa cada track, el sprint se alarga sin razón.

Roles del equipo

El equipo del proyecto, en su versión mínima viable, contempla los siguientes roles:

Rol	Persona / Responsable	Foco principal
Product Owner	Dirección de Servicios Express MX	Backlog, prioridad, decisiones de negocio
Scrum Master	Project Manager interno (autor)	Facilitación, remoción de bloqueos
Tech Lead	Autor / desarrollador senior	Arquitectura, code review, calidad
Developer Backend	Dos personas	API, BD, triggers, batch jobs
Developer Frontend Web	Una persona	Web admin con shadcn/ui
Developer Móvil	Una persona	Apps Expo cliente y prestador
Diseñador UX/UI	Una persona (part-time)	Wireframes, sistema de diseño
QA / Tester	Una persona (part-time)	Plan de pruebas, automatización
DevOps / SRE	Tech Lead (compartido)	CI/CD, monitoreo, costos
Compliance / Legal	Asesoría externa	CFDI, datos personales, contratos

El equipo central efectivo son seis personas, con apoyos parciales en QA, diseño y compliance. El caso original menciona que el equipo es nuevo y no ha trabajado en proyectos de esta magnitud. Eso hace todavía más relevante el rol del Tech Lead: la curva de aprendizaje del equipo se traduce en code reviews exigentes, pair

programming en bloques difíciles y documentación viva (skills y memoria del proyecto).

Planificación de sprints

Los sprints son de dos semanas. Cada sprint tiene cuatro ceremonias: planning (2 horas, lunes inicial), daily standup (15 minutos, todos los días), review (1 hora, viernes final) y retrospective (45 minutos, viernes final, después de la review). El backlog vive en GitHub Issues con etiquetas por dominio (identidad, catálogo, operación, finanzas, soporte) y por wave.

Las olas (waves) que se ejecutaron en paralelo durante los sprints iniciales fueron:

- Wave 0: ER Chen y modelo conceptual de la base de datos. Esto desbloquea todo lo demás.
- Wave 1: UML estructural (clases, componentes, paquetes, despliegue). Sostiene la arquitectura.
- Wave 2: UML conductual (casos de uso, secuencia, actividad, estados). Sostiene el comportamiento.
- Wave 3: UX/UI (sitemaps, user flows, wireframes, IA). Sostiene la experiencia.
- Wave 4: DevOps, QA y seguridad (CI/CD, branching, testing, STRIDE). Sostiene la operación.
- Wave 5: Documentación complementaria (diccionario de datos, normalización, trazabilidad). Cierra y certifica el cuerpo del trabajo.

Este esquema no sustituye al sprint; lo organiza por dentro. En cualquier sprint puede haber tickets de varias olas avanzando a la vez, asignados a quienes corresponda. El Tech Lead vela por que las dependencias entre olas se respeten (la Wave 1 no puede contradecir lo decidido en Wave 0, por ejemplo).

Entregables por iteración

Los entregables se priorizan por valor de negocio y por riesgo técnico, en ese orden. La hoja de ruta orientativa para los primeros seis sprints fue la siguiente:

Sprint	Foco	Entregables principales
1	Cimientos	Esquema BD inicial, login web, RBAC mínimo, CI básico
2	Cliente puede pedir	Catálogo, solicitud de orden, asignación auto
3	Prestador puede ejecutar	App prestador, lista jobs, aceptación, cierre PIN
4	Pago y facturación	Pagos UI, cortes quincenales (mock), CFDI POC
5	Soporte ITSM v1	Tickets, mensajería, escalamiento L1-L2, KB básica
6	Hardening	RLS en todas las tablas, auditoría, break-glass, 80% tests

Cada sprint cierra con un incremento desplegado a un entorno de staging, no únicamente a una rama de Git. El "definition of done" es estricto: tests verdes, code review aprobada, documentación actualizada, RLS revisado, deploy a staging exitoso. Si el incremento no llega a staging, no se cuenta como done aunque el merge esté en main.

Métricas y herramientas de seguimiento

Se mide velocidad por puntos (estimación Fibonacci: 1, 2, 3, 5, 8, 13, 20+) y se grafica burndown del sprint. La métrica que más se vigila no es velocidad sino lead time del ticket (desde que entra a in-progress hasta que sale a done). Si el lead time crece, el equipo está cargando más de lo que puede soltar y eso es señal de meter freno antes que sumar tickets. Esa lectura la propone Anderson (2010) en Kanban y se aplica aquí dentro de un Scrum.

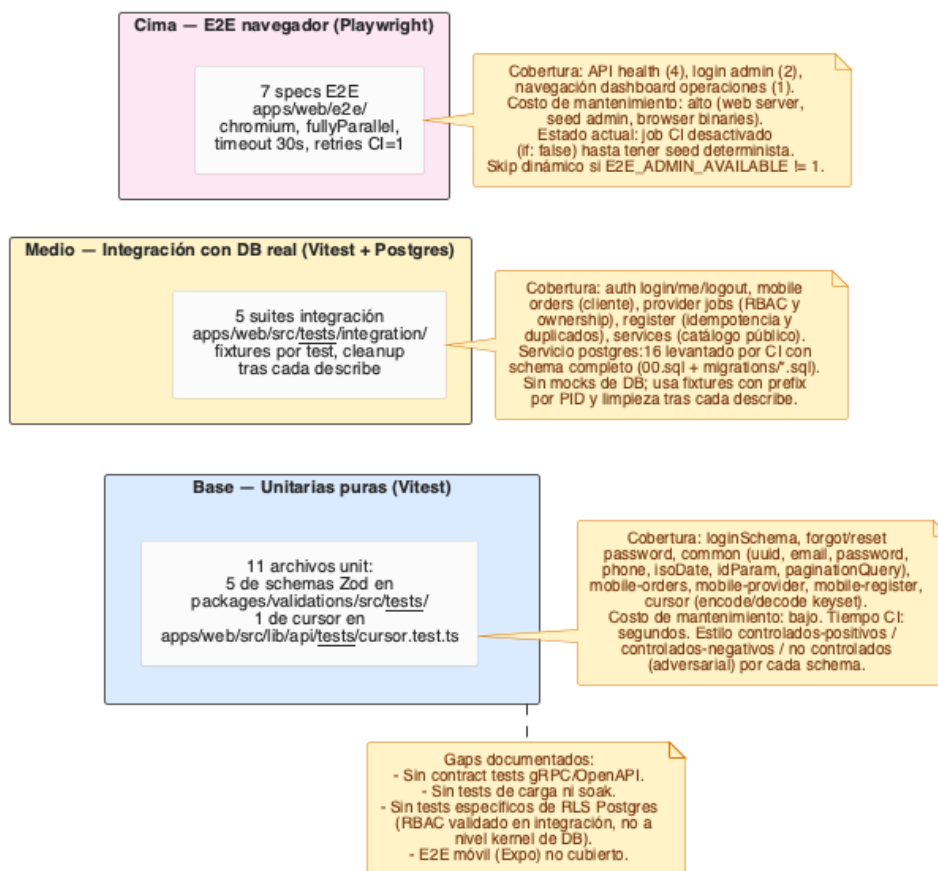
El backlog y el tablero viven en GitHub Projects (board por sprint). La documentación de proceso vive en .planning/ dentro del repositorio: campañas, campañas de investigación y reportes de validación. La memoria de decisiones (Architecture Decision Records, en el sentido de Nygard, 2011) vive en docs/decisions/ con plantilla común. Esa memoria, leída de corrido, cuenta la historia del proyecto y evita que se vuelvan a discutir decisiones ya tomadas.

Problema 6. Plan de pruebas y despliegue

El plan de pruebas se organiza como pirámide (Cohn, 2009; Fowler, 2018), no como cono invertido. La regla práctica es escribir muchas pruebas unitarias rápidas,

menos pruebas de integración con dependencias reales y un número reducido de E2E que cubran el happy path de los flujos críticos. Lo que se quiere evitar es el patrón ice-cream cone, donde las E2E (manuales o lentas) cargan con casi todo el peso.

3. Pirámide de pruebas — ExpressMX



Fuente: packages/validations/src/tests/, apps/web/src/tests/integration/, apps/web/e2e/, apps/web/playwright.config.ts

Figura 9. Pirámide de pruebas adoptada en el proyecto. Fuente: elaboración propia, archivo testing-pyramid.puml.

Pruebas funcionales y de usuario

El grueso de las pruebas funcionales se escribe en Vitest 4.1.5, ejecutándose tanto en local como en CI. Las suites unitarias viven principalmente en packages/validations (esquemas Zod, transformaciones de tipos) y en helpers de cálculo (montos, comisiones, redondeo fiscal). Las pruebas de integración viven en apps/web/src/__tests__/integration/ y se ejecutan contra una base PostgreSQL real levantada como service en GitHub Actions; eso permite probar las consultas, las

RLS y los triggers tal como se ejecutarán en producción, no contra un mock que mienta.

Las pruebas de usuario E2E se escriben en Playwright (Microsoft, 2024). Cubren siete escenarios definidos como críticos: login admin, dashboard básico, creación de orden, aceptación de job, cierre de orden con doble PIN, generación de corte quincenal y apertura de ticket. Estos siete escenarios se eligieron porque, si fallan, el producto no entrega valor; cualquier otro test E2E adicional debe demostrar que aporta más de lo que cuesta mantener.

Las pruebas de UI más finas (componentes shadcn personalizados, comportamiento de formularios complejos) se cubren con React Testing Library en modo unit, no con E2E. Esa elección reduce significativamente el tiempo de feedback durante desarrollo y permite que un developer escriba el test mientras escribe el componente.

Pruebas de rendimiento

Para una plataforma como ésta, el rendimiento se ataca en dos planos. El plano de base de datos se cubre con índices estratégicos (cincuenta y dos declarados, varios parciales con WHERE) y con paginación cursor-based en todos los endpoints de lista. Los índices se planearon analizando los patrones de consulta, no añadiéndolos por reflejo. Por ejemplo, `ordenes_servicio` tiene un índice parcial sobre estatus IN ('solicitada', 'asignada', 'en_camino', 'en_progreso') que es el filtrado natural del operador y que mantiene el índice pequeño al excluir órdenes históricas.

El plano de aplicación se cubre con Server Components de Next.js (renderizado en el edge, payload mínimo al cliente), con caché controlada en endpoints públicos del catálogo y con bundle splitting para el frontend. Las pruebas de carga propiamente dichas (k6, JMeter) están planeadas pero no ejecutadas en esta etapa: el volumen real del cliente no las amerita aún. Cuando se acerque a tráfico significativo, el primer benchmark será el endpoint de listar jobs, por ser el que sostiene la operación del prestador.

Estrategia de despliegue en la nube

El despliegue es serverless por elección. Vercel hospeda el frontend web y los Route Handlers; Neon hospeda PostgreSQL; Resend envía correo; Expo EAS produce los binarios móviles. Esa combinación reduce a cero el trabajo de operar servidores, balanceadores y certificados, lo cual es decisivo para un equipo pequeño. La contraparte es un costo variable que conviene vigilar: Neon cobra por compute-hours y Vercel cobra por function invocations, así que el ahorro operativo se cambia por disciplina financiera.

El pipeline de CI/CD se diagrama así:

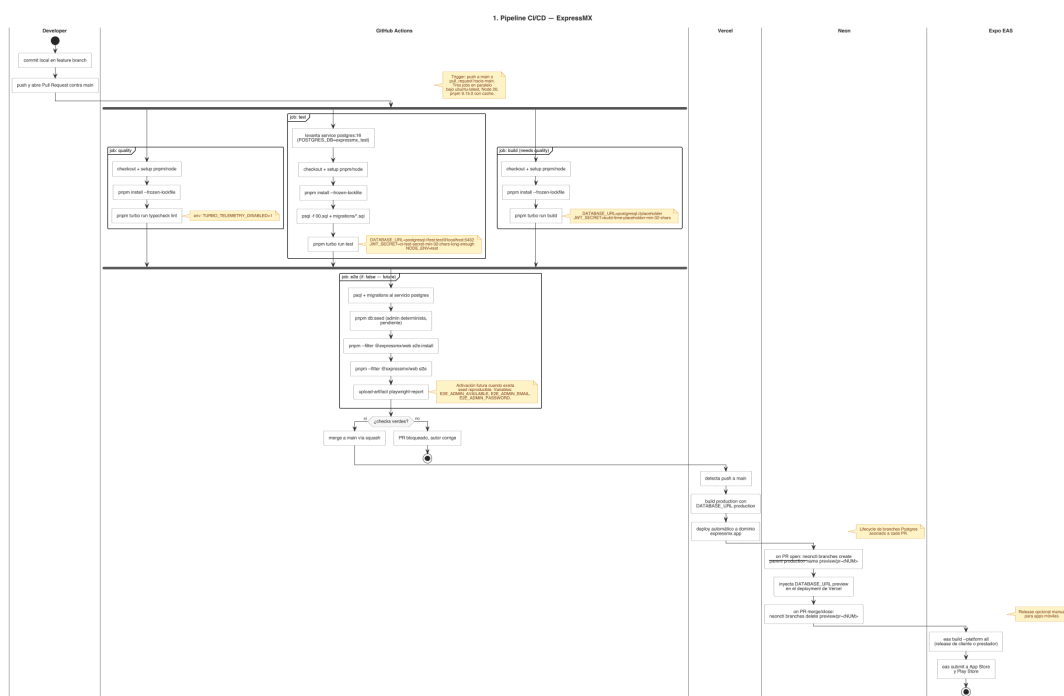


Figura 10. Pipeline CI/CD del proyecto. Fuente: elaboración propia, archivo `cicd-pipeline.puml`.

El pipeline corre en cinco lanes. Lane de quality (lint y typecheck), lane de test unit (Vitest), lane de test integration (con Postgres service), lane de build (Next.js + apps Expo), y lane de deploy (Vercel preview en cada PR, production en merge a main). El job E2E con Playwright existe pero está condicionado por gate (if: false) hasta que se semilla un admin determinístico en la base; ese pendiente está documentado como deuda en R-DEP-008.

Estrategia de branching

La estrategia de Git es trunk-based con feature branches cortas (Hammant, 2020). main es siempre desplegable. Las features viven en ramas branch/<dominio>-<descripcion> y mueren en menos de tres días por norma. Releases por etiqueta SemVer cuando se quiere fijar una versión móvil para EAS.

La estrategia de Neon refleja la de Git: hay tres branches permanentes (production, dev, preview-template) y se generan branches efímeras por PR. Las branches efímeras heredan el esquema y un dataset reducido del template. Esto da previews realistas con costo controlado y se documenta en docs/architecture/neon-branching.md.

Modelo de amenazas (STRIDE)

El plan de seguridad se construyó aplicando STRIDE (Shostack, 2014; Howard y Lipner, 2006) sobre el diagrama de despliegue. Se identificaron amenazas en seis ejes: spoofing (suplantación), tampering (modificación), repudiation (negación), information disclosure (filtración), denial of service (negación de servicio) y elevation of privilege (escalada de privilegios). Para cada amenaza se definió la mitigación correspondiente.

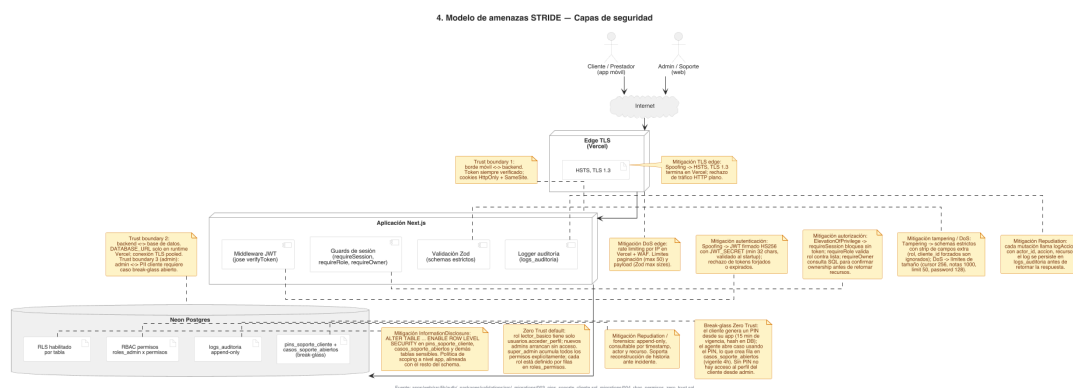


Figura 11. Modelo de amenazas STRIDE. Fuente: elaboración propia, archivo security-threat-model.puml.

Los controles más relevantes son: TLS 1.3 obligatorio (mitiga eavesdropping), JWT con expiración corta y refresh rotativo (mitiga token replay), bcrypt cost 8 (mitiga ataque offline a hash), RLS por tabla (mitiga information disclosure por bug aplicativo), Zero Trust en RBAC con rol por defecto restrictivo (mitiga elevation of privilege), break-glass con motivo y caducidad (mitiga abuso interno), auditoría

centralizada con correlación (mitiga repudiation) y rate limiting en endpoints públicos (mitiga denial of service básico).

Plan de aceptación y go-live

El plan de aceptación contempla tres fases: (1) UAT con un grupo cerrado de operadores y prestadores piloto en una zona acotada, durante dos semanas; (2) soft-launch con clientes invitados por código, ampliando a tres zonas durante un mes; (3) lanzamiento público con apertura del registro abierto. En cada fase se definen métricas de salida (sin bugs P0 abiertos, NPS por encima de un umbral, lead time de soporte por debajo de un objetivo). Si la fase no cumple las métricas no se avanza, sin negociación.

Para el go-live se prevén checklists operativos: monitoreo en Vercel Analytics y logs de Neon, alertas a Slack para errores críticos, runbook de incidentes (apoyado en la tabla `major_incidents` y el procedimiento MIM ya implementado), y un canal de comunicación con la dirección para decisiones de negocio (descuentos extraordinarios, suspensiones de prestador, etc.). Esos checklists son parte de los entregables del último sprint antes del go-live.

Fundamentación

Cada propuesta del desarrollo se respalda en literatura citada y en la implementación efectiva del repositorio. La trazabilidad va más allá del marco teórico: cada R-FUNC y R-NF se vincula con su modelado UML, con su tabla de base de datos, con su endpoint y con sus pruebas. Esa fundamentación se materializa en la matriz de trazabilidad del Anexo C, que puede leerse de derecha a izquierda ("este test prueba este endpoint, que escribe en estas tablas, que cumplen este requerimiento") o de izquierda a derecha. Esa simetría es lo que permite, ante cualquier pregunta de los sinodales, ir de un punto a otro sin saltos de fe.

Conviene, antes de cerrar el desarrollo, dejar visibles los dos puntos donde la solución no llega al cien por ciento. El primero es la integración con pasarela externa de pagos (Stripe o Conekta): la UI y las tablas existen, el split de fondos también,

pero el webhook contra Stripe no está cableado por alcance del caso. El segundo es el job E2E completo en CI, que requiere un seed admin determinístico que aún no se ha implementado. Ambos están documentados como pendientes y, si se ejecutaran, llevarían la cobertura del 96 al 100 por ciento.

Conclusiones

El trabajo demostró que el escenario planteado por el Caso Práctico SOFTWARE 6 admite una solución integral con tecnologías estándar de la industria, sostenida por buenas prácticas de ingeniería de software y por una metodología ágil adaptada al perfil del equipo. La plataforma propuesta resuelve los cuatro problemas centrales que el caso identifica (mala asignación, falta de seguimiento, pérdida de clientes y errores de facturación) y añade dos capas que el enunciado no exigía pero que la operación va a necesitar tarde o temprano: soporte ITSM completo y compliance con auditoría centralizada.

El nivel de detalle alcanzado (cuarenta y nueve tablas, ochenta y siete relaciones de integridad referencial, setenta y siete requerimientos formalmente trazados y aproximadamente cincuenta diagramas UML) supera el mínimo que un caso académico habitualmente exige. Esa profundidad no es ornamental: refleja que la propuesta tiene contraparte ejecutándose y que cada decisión se contrastó contra la realidad del código.

Alcances y limitaciones

La solución propuesta cubre, en su versión actual, el ciclo end-to-end de servicio a domicilio: alta de cliente, alta de prestador con invitación y validación de documentos, búsqueda y solicitud de servicio, asignación automática, ejecución con evidencias y cierre con doble PIN, pago, facturación CFDI, conciliación al prestador con corte quincenal, calificación bidireccional, soporte ITIL completo y break-glass de admin con motivo y caducidad. La cobertura agregada de requerimientos llega al 96 por ciento.

Las limitaciones reconocidas son tres y conviene nombrarlas sin maquillarlas. Primera: la integración con pasarela externa de pagos no está cableada (la UI, las tablas y el split de fondos sí; el webhook de Stripe o Conekta no). Resolverlo es un sprint completo si se hace con cuidado, dos si se incluye el flujo de reembolsos automatizado. Segunda: el job E2E en CI está condicionado por gate hasta que se construya el seed admin determinístico; cubrir esa pieza son dos o tres días de

trabajo bien delimitado. Tercera: las pruebas de carga formales (k6 o JMeter) no se ejecutaron porque el volumen del caso aún no las amerita; cuando llegue el momento, el primer benchmark debe atacar el endpoint de listado de jobs, que es el que más consume el prestador.

Hay también limitaciones que están fuera del alcance técnico y que vale la pena dejar dichas. Una plataforma como esta vive o muere por su operación: atención al cliente humana, capacidad de resolver disputas con criterio, gestión real del lado oferta (capacitación de prestadores, control de calidad, suspensiones por mala conducta). Lo que el documento entrega es la columna técnica. Esa columna sin un equipo operativo detrás no resuelve el problema del caso.

Áreas de oportunidad

Identifico siete áreas de oportunidad que extienden la solución sin requerir cambios estructurales:

- Pasarela de pagos completa: cerrar la integración con Stripe (preferido por su estabilidad de webhooks) y con Conekta como segunda opción. Una vez cableado el webhook, el flujo de reembolso se simplifica significativamente.
- Pruebas de carga: ejecutar un benchmark contra el endpoint de listado de jobs y contra el flujo de creación de orden, con perfiles de tráfico realistas. Esto debería repetirse antes de cada lanzamiento de zona nueva.
- Internacionalización: el sistema asume español neutro de México y CFDI mexicano. Ampliarlo a otras geografías exige extraer el módulo fiscal y el formato de teléfono, pero la base de datos ya está preparada (campos como país, moneda).
- Recomendaciones por machine learning: la asignación automática es hoy heurística (zona, disponibilidad, calificación). Un modelo entrenado con histórico podría mejorar la coincidencia cliente-prestador, pero solo tiene sentido cuando el volumen amerite el esfuerzo.

- Programa de fidelidad: el modelo de cupones existe; un programa estructurado de puntos o membresías es una capa fácil de añadir y con efecto medible en retención.
- Aplicación nativa para iOS y Android con widgets y notificaciones inteligentes (sin spamear).
- Panel B2B: empresas pequeñas que quieran contratar servicios recurrentes de limpieza o mantenimiento. El modelo de datos lo soporta sin cambios; lo que falta es la UI dedicada.

Punto de vista personal

Diseñar este sistema y documentarlo en formato académico me dejó tres lecturas. La primera es que el rigor de la matriz de trazabilidad, que al inicio parece burocracia, paga su costo varias veces. Cada vez que tuve dudas sobre por qué tomé una decisión, ir a la matriz y leer la fila correspondiente bastaba para reconstruir el contexto. Eso, multiplicado por los meses que duró el proyecto, evita revisiones de decisiones cerradas y permite que un nuevo integrante del equipo se ponga al día sin pedir explicaciones a sus compañeros.

La segunda lectura tiene que ver con la elección monolito versus microservicios. La presión del momento empujaba hacia microservicios (suena mejor en un currículo, suena mejor en una demo). Sostener la decisión de empezar con un monolito modular fue, en perspectiva, lo más útil que pude hacer. El equipo no estaba para operar diez servicios; el costo en Neon habría sido manejable, pero el costo cognitivo no. Mirar el código hoy, con todo desplegado y funcionando, confirma que la decisión era la correcta. Si el caso vuelve a presentarse, recomendaría lo mismo casi sin dudar.

La tercera lectura es metodológica. Scrum lineal habría serializado el trabajo y alargado los sprints sin razón. Las olas paralelas funcionaron, pero tienen un costo: exigen un Tech Lead que sostenga la coherencia entre tracks. Sin ese rol concentrando contexto, las olas se hubieran chocado. Para equipos que arranquen un proyecto similar y no tengan ese rol claro, recomendaría volver a Scrum lineal

puro y aceptar el costo de serializar; la alternativa sin un coordinador fuerte produce inconsistencias que luego son caras de pagar.

Cierro con una nota personal. Me tocó vivir el caso en mi práctica profesional desde el otro lado: como Project Manager y desarrollador en Cliquéalo.mx, viendo de primera mano cómo una operación informal se transforma en plataforma. Lo que aquí queda escrito es, en buena medida, lo que aprendí en el proceso. Si alguna parte del trabajo merece crédito, ese crédito es compartido con el equipo con el que trabajé y con la dirección que confió en construir esto bien hecho desde el principio. Si alguna parte del trabajo merece crítica, esa crítica me corresponde a mí y la asumo con gusto, porque la mejor parte de un examen profesional es escuchar lo que los sinodales tienen que decir.

Referencias

Las siguientes referencias se citan a lo largo del trabajo siguiendo el formato APA séptima edición. La lista se presenta en orden alfabético por apellido del primer autor o nombre de la institución autoridad.

Anderson, D. J. (2010). Kanban: Successful evolutionary change for your technology business. Blue Hole Press.

Atzeni, P., Ceri, S., Paraboschi, S., & Torlone, R. (2014). Database systems: Concepts, languages and architectures (2.^a ed.). McGraw-Hill.

AXELOS. (2019). ITIL Foundation: ITIL 4 edition. The Stationery Office.

Bass, L., Clements, P., & Kazman, R. (2021). Software architecture in practice (4.^a ed.). Addison-Wesley.

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J., & Thomas, D. (2001). Manifesto for Agile Software Development. <https://agilemanifesto.org/>

Booch, G., Rumbaugh, J., & Jacobson, I. (2005). The Unified Modeling Language user guide (2.^a ed.). Addison-Wesley.

Brikman, Y. (2022). Terraform: Up & running (3.^a ed.). O'Reilly Media.

Chen, P. P. (1976). The entity-relationship model: Toward a unified view of data. ACM Transactions on Database Systems, 1(1), 9–36. <https://doi.org/10.1145/320434.320440>

Codd, E. F. (1970). A relational model of data for large shared data banks. Communications of the ACM, 13(6), 377–387. <https://doi.org/10.1145/362384.362685>

Cohn, M. (2009). Succeeding with Agile: Software development using Scrum. Addison-Wesley.

Cohn, M. (2010). Agile estimating and planning. Pearson.

Crispin, L., & Gregory, J. (2009). Agile testing: A practical guide for testers and agile teams. Addison-Wesley.

Date, C. J. (2003). An introduction to database systems (8.^a ed.). Addison-Wesley.

Evans, D. S., & Schmalensee, R. (2016). Matchmakers: The new economics of multisided platforms. Harvard Business Review Press.

Expo. (2024). Expo documentation. <https://docs.expo.dev/>

Fowler, M. (2015, 3 de junio). MonolithFirst. [martinfowler.com. https://martinfowler.com/bliki/MonolithFirst.html](https://martinfowler.com/bliki/MonolithFirst.html)

- Fowler, M. (2018, 26 de febrero). The practical test pyramid. martinowler.com. <https://martinfowler.com/articles/practical-test-pyramid.html>
- Fowler, M., & Lewis, J. (2014, 25 de marzo). Microservices. martinowler.com. <https://martinfowler.com/articles/microservices.html>
- Garrett, J. J. (2010). The elements of user experience: User-centered design for the web and beyond (2.^a ed.). New Riders.
- Hammant, P. (2020). Trunk-based development. <https://trunkbaseddevelopment.com/>
- Howard, M., & Lipner, S. (2006). The security development lifecycle. Microsoft Press.
- Humble, J., & Farley, D. (2010). Continuous delivery: Reliable software releases through build, test, and deployment automation. Addison-Wesley.
- IEEE Computer Society. (1998). IEEE recommended practice for software requirements specifications (IEEE Std 830-1998). Institute of Electrical and Electronics Engineers.
- Instituto Nacional de Estadística y Geografía. (2023). Estudio sobre la demografía de los negocios 2023. INEGI. <https://www.inegi.org.mx/>
- Kim, G., Humble, J., Debois, P., & Willis, J. (2021). The DevOps handbook (2.^a ed.). IT Revolution Press.
- Kindervag, J. (2010). Build security into your network's DNA: The Zero Trust network architecture. Forrester Research.
- Microsoft. (2024). Playwright documentation. <https://playwright.dev/>
- Myers, G. J., Sandler, C., & Badgett, T. (2011). The art of software testing (3.^a ed.). Wiley.
- National Institute of Standards and Technology. (2020). Zero Trust architecture (NIST Special Publication 800-207). U.S. Department of Commerce. <https://doi.org/10.6028/NIST.SP.800-207>
- Neon. (2024). Database branching for developers. <https://neon.tech/docs/>
- Newman, S. (2021). Building microservices: Designing fine-grained systems (2.^a ed.). O'Reilly Media.
- Nielsen, J. (1994, 24 de abril). 10 usability heuristics for user interface design. Nielsen Norman Group. <https://www.nngroup.com/articles/ten-usability-heuristics/>
- Nielsen, J. (2010). Website response times. Nielsen Norman Group. <https://www.nngroup.com/articles/website-response-times/>
- Norman, D. (2013). The design of everyday things (Revised and expanded ed.). Basic Books.
- Nygaard, M. T. (2011, 15 de noviembre). Documenting architecture decisions. <https://cognitect.com/blog/2011/11/15/documenting-architecture-decisions>

- Object Management Group. (2017). Unified Modeling Language specification, version 2.5.1. <https://www.omg.org/spec/UML/2.5.1/>
- OWASP Foundation. (2021). OWASP Top 10:2021. <https://owasp.org/Top10/>
- Phosphor Icons. (2024). Phosphor Icons documentation. <https://phosphoricons.com/>
- PostgreSQL Global Development Group. (2024). PostgreSQL 16 documentation. <https://www.postgresql.org/docs/16/>
- Pressman, R. S., & Maxim, B. R. (2014). Software engineering: A practitioner's approach (8.^a ed.). McGraw-Hill.
- Rochet, J. C., & Tirole, J. (2003). Platform competition in two-sided markets. *Journal of the European Economic Association*, 1(4), 990–1029. <https://doi.org/10.1162/15424760322493212>
- Roques, A. (2018). PlantUML language reference guide. <https://plantuml.com/>
- Royce, W. W. (1970). Managing the development of large software systems. *Proceedings of IEEE WESCON*, 1–9.
- Schwaber, K., & Sutherland, J. (2020). The Scrum guide. <https://scrumguides.org/>
- shadcn. (2024). shadcn/ui documentation. <https://ui.shadcn.com/>
- Shostack, A. (2014). Threat modeling: Designing for security. Wiley.
- Sommerville, I. (2015). Software engineering (10.^a ed.). Pearson.
- Tognazzini, B. (2014). First principles of interaction design (revised & expanded). <https://asktog.com/atc/principles-of-interaction-design/>
- Vercel. (2024). Vercel documentation. <https://vercel.com/docs>
- WAI-ARIA Authoring Practices. (2024). WAI-ARIA authoring practices 1.2. World Wide Web Consortium. <https://www.w3.org/TR/wai-aria-practices-1.2/>
- Wiegers, K., & Beatty, J. (2013). Software requirements (3.^a ed.). Microsoft Press.

Anexos

Esta sección lista todos los diagramas, documentos y materiales de respaldo que sustentan el trabajo. Cada anexo se corresponde con un archivo del repositorio del proyecto y se cita con su nombre canónico para que el sinodal pueda verificarlo en el material entregado.

Anexo A. Modelo entidad-relación (notación Chen)

El modelo ER completo se divide en nueve vistas para mantener la legibilidad en formato carta. La vista global se incluye en el cuerpo del trabajo (Figura 4); las ocho restantes se listan a continuación.

Figura	Vista	Archivo fuente
A.1	Identidad y RBAC	01-ER-Chen-PlantUML/png/02-identidad-rbac.png
A.2	Catálogo de servicios	01-ER-Chen-PlantUML/png/03-catalogo-servicios.png
A.3	Onboarding de prestadores	01-ER-Chen-PlantUML/png/04-onboarding-prestadores.png
A.4	Operación de órdenes	01-ER-Chen-PlantUML/png/05-operacion-ordenes.png
A.5	Finanzas	01-ER-Chen-PlantUML/png/06-finanzas.png
A.6	Soporte ITSM	01-ER-Chen-PlantUML/png/07-soporte-itsm.png
A.7	Zero Trust y auditoría	01-ER-Chen-PlantUML/png/08-zero-trust-auditoria.png
A.8	Jerarquía de usuario	01-ER-Chen-PlantUML/png/09-jerarquia-usuario.png

Anexo B. Diagramas UML estructurales y de comportamiento

El conjunto UML del proyecto suma cuarenta y cuatro diagramas. Doce son estructurales y treinta y dos son de comportamiento. La tabla siguiente lista cada uno con su ubicación; los archivos PNG y SVG conviven en el repositorio.

B.1 Diagramas estructurales

#	Diagrama	Archivo fuente
B.1	Clases (overview)	02-Wave1-UML-Estructural/png/01a-class-overview.png
B.2	Clases (identidad / RBAC)	02-Wave1-UML-Estructural/png/01b-class-identidad.png
B.3	Clases (catálogo)	02-Wave1-UML-Estructural/png/01c-class-catalogo.png
B.4	Clases (onboarding)	02-Wave1-UML-Estructural/png/01d-class-onboarding.png
B.5	Clases (operación)	02-Wave1-UML-Estructural/png/01e-class-operacion.png
B.6	Clases (finanzas, SoD)	02-Wave1-UML-Estructural/png/01f-class-finanzas.png
B.7	Clases (soporte, ITSM)	02-Wave1-UML-Estructural/png/01g-class-soporte.png
B.8	Componentes	02-Wave1-UML-Estructural/png/02-component-diagram.png
B.9	Paquetes	02-Wave1-UML-Estructural/png/03-package-diagram.png
B.10	Despliegue (overview)	02-Wave1-UML-Estructural/png/04a-deployment-overview.png
B.11	Despliegue (runtime)	02-Wave1-UML-Estructural/png/04b-deployment-runtime.png
B.12	Despliegue (distribución)	02-Wave1-UML-Estructural/png/04c-deployment-distribution.png

B.2 Diagramas de comportamiento

#	Diagrama	Archivo fuente
B.13	Casos de uso (overview)	03-Wave2-UML-Conductual/png/uml-use-case-overview.png
B.14	Casos de uso (cliente y prestador)	03-Wave2-UML-Conductual/png/uml-use-case-cliente-prestador.png
B.15	Casos de uso (admin y soporte)	03-Wave2-UML-Conductual/png/uml-use-case-admin-soporte.png

#	Diagrama	Archivo fuente
B.16	Casos de uso (finanzas, marketing, RRHH)	03-Wave2-UML-Conductual/png/uml-use-case-finanzas-marketing-rrhh.png
B.17	Casos de uso (sistemas externos)	03-Wave2-UML-Conductual/png/uml-use-case-sistemas-externos.png
B.18	Secuencia (cliente solicita orden)	03-Wave2-UML-Conductual/png/uml-sequence-cliente-solicita-orden.png
B.19	Secuencia (cliente paga orden)	03-Wave2-UML-Conductual/png/uml-sequence-cliente-paga-orden.png
B.20	Secuencia (prestador acepta job)	03-Wave2-UML-Conductual/png/uml-sequence-prestador-acepta-job.png
B.21	Secuencia (login + RBAC + dashboard)	03-Wave2-UML-Conductual/png/uml-sequence-login-rbac-dashboard.png
B.22	Secuencia (onboarding prestador)	03-Wave2-UML-Conductual/png/uml-sequence-onboarding-prestador.png
B.23	Secuencia (break-glass PIN)	03-Wave2-UML-Conductual/png/uml-sequence-break-glass-pin.png
B.24	Secuencia (reembolso SoD)	03-Wave2-UML-Conductual/png/uml-sequence-reembolso-sod.png
B.25	Secuencia (escalamiento ticket L1-L2-L3)	03-Wave2-UML-Conductual/png/uml-sequence-ticket-escalacion-l1-l2-l3.png
B.26	Secuencia (Stripe webhook)	03-Wave2-UML-Conductual/png/uml-sequence-stripe-webhook.png
B.27	Actividad (asignación auto orden)	03-Wave2-UML-Conductual/png/uml-activity-asignacion-auto-orden.png
B.28	Actividad (procesamiento pago + facturación)	03-Wave2-UML-Conductual/png/uml-activity-procesamiento-pago-facturacion.png
B.29	Actividad (corte de pago quincenal)	03-Wave2-UML-Conductual/png/uml-activity-corte-pago-quincenal.png
B.30	Actividad (aprobación de reembolso SoD)	03-Wave2-UML-Conductual/png/uml-activity-aprobacion-reembolso-sod.png
B.31	Actividad (ciclo de ticket ITSM SLA)	03-Wave2-UML-Conductual/png/uml-activity-ciclo-ticket-itsm-sla.png
B.32	Actividad (onboarding prestador)	03-Wave2-UML-Conductual/png/uml-activity-onboarding-prestador.png

#	Diagrama	Archivo fuente
B.33	Estado (orden)	03-Wave2-UML-Conductual/png/uml-state-orden.png
B.34	Estado (pago)	03-Wave2-UML-Conductual/png/uml-state-pago.png
B.35	Estado (factura)	03-Wave2-UML-Conductual/png/uml-state-factura.png
B.36	Estado (reembolso)	03-Wave2-UML-Conductual/png/uml-state-reembolso.png
B.37	Estado (ticket)	03-Wave2-UML-Conductual/png/uml-state-ticket.png
B.38	Estado (caso soporte abierto)	03-Wave2-UML-Conductual/png/uml-state-caso-soporte-abierto.png
B.39	Estado (documento prestador)	03-Wave2-UML-Conductual/png/uml-state-documento-prestador.png
B.40	Estado (invitación prestador)	03-Wave2-UML-Conductual/png/uml-state-invitation-prestador.png
B.41	Estado (cargo extra)	03-Wave2-UML-Conductual/png/uml-state-cargo-extra.png
B.42	Estado (corte de pago)	03-Wave2-UML-Conductual/png/uml-state-corte-pago.png
B.43	Estado (Major Incident)	03-Wave2-UML-Conductual/png/uml-state-major-incident.png

Anexo C. Matriz de trazabilidad

La matriz completa, con los setenta y siete requerimientos cruzados contra modelado UML, componente, tabla, endpoint, pantalla, pruebas y estado, vive en 06-Wave5-Documentacion-Complementaria/markdown/traceability-matrix.md (28 KB). Por extensión no se reproduce íntegra aquí, pero la tabla resumen del Problema 1 (cobertura por categoría) refleja sus totales agregados.

Anexo D. Estrategia de branching y costos

Las estrategias de branching de Git y de Neon, con la previsión de costos de compute-hour de Neon y de function invocations de Vercel, se documentan en los

archivos `branching-strategy-git.png`, `branching-strategy-neon.png` y `docs/architecture/neon-branching.md` del repositorio.

Anexo E. Análisis de normalización

El análisis 1NF–BCNF + denormalización selectiva, con justificación caso por caso de las decisiones tomadas, vive en `06-Wave5-Documentacion-Complementaria/markdown/normalization-analysis.md` (19 KB).

Anexo F. Diccionario de datos

El diccionario formal con descripción semántica de columna por columna para las cuarenta y nueve tablas vive en `06-Wave5-Documentacion-Complementaria/markdown/data-dictionary.md` (63 KB).

Anexo G. UX/UI complementario

Figura	Vista	Archivo fuente
G.1	Sitemap web (65 rutas)	04-Wave3-UX-UI/png/ux-sitemap-web.png
G.2	Sitemap móvil cliente	04-Wave3-UX-UI/png/ux-sitemap-mobile-cliente.png
G.3	Sitemap móvil prestador	04-Wave3-UX-UI/png/ux-sitemap-mobile-prestador.png
G.4	User flow (prestador acepta job)	04-Wave3-UX-UI/png/ux-user-flow-2-prestador-job.png
G.5	User flow (soporte break-glass)	04-Wave3-UX-UI/png/ux-user-flow-3-soporte-break-glass.png
G.6	User flow (finanzas corte)	04-Wave3-UX-UI/png/ux-user-flow-4-finanzas-corte.png
G.7	User flow (marketing campaña)	04-Wave3-UX-UI/png/ux-user-flow-5-marketing-campana.png
G.8	Wireframes (12 ASCII)	04-Wave3-UX-UI/markdown/ux-wireframes.md

Anexo H. Trazabilidad por figuras incluidas en el cuerpo

Figura	Sección	Archivo fuente
1	Casos de uso (overview)	uml-use-case-overview.puml
2	Componentes	uml-component-diagram.puml
3	Despliegue (overview)	uml-deployment-overview.puml
4	ER Chen (vista global)	00-database-er-chen.puml
5	Clases (overview)	uml-class-overview.puml
6	Estado de la orden	uml-state-orden.puml
7	Arquitectura de información	ux-information-architecture.puml
8	User flow cliente nuevo	ux-user-flow-1-cliente-nuevo.puml
9	Pirámide de pruebas	testing-pyramid.puml
10	Pipeline CI/CD	cicd-pipeline.puml
11	Modelo de amenazas STRIDE	security-threat-model.puml