Software Development Design (SDD)

Proyecto: ReparaYa — MVP Web (Zona Metropolitana de Guadalajara)

Versión: v1.0
Fecha: 10/09/2025
Propietario del SDD: Mateo Garcia Lopez
Sponsor: David Emmanuel Ramírez
Clasificación: Uso interno y Sponsor

0. Control de cambios

Versión	Fecha	Autor
1.0	11/09/2025	Equipo

1. Introducción

1.1 Propósito

El propósito de este documento es definir el diseño y arquitectura del sistema **ReparaYa**. Se detallarán las funciones y arquitectura del proyecto así como la forma en la que se conectan los distintos servicios, sus interfaces de comunicación y las dependencias externas. Este diseño proporciona la guía técnica para el desarrollo, integración y mantenimiento del sistema, garantizando el cumplimiento de los requerimientos desarrollados en el documento **SRS** (Software Requirements Specification).

1.2 Alcance

ReparaYa está orientada a ofrecer una **solución digital integral** que facilite la **búsqueda, reserva, calificación y pago de contratistas** dedicados a diversos servicios locales (por ejemplo, mantenimiento, reparaciones o instalaciones).

El sistema permitirá a los usuarios **explorar y seleccionar servicios** dentro de una amplia variedad de categorías, así como **registrarse para ofrecer sus propios servicios** dentro de la misma plataforma, promoviendo un **ecosistema colaborativo** entre clientes y prestadores.

Asimismo, la plataforma integrará un **sistema de calificación y reseñas** que permitirá a los usuarios **evaluar la calidad del servicio recibido**, generando una retroalimentación constante que contribuya a la mejora continua de los contratistas y fomente la confianza y transparencia dentro del sistema.

Durante su fase inicial, el alcance del proyecto se limitará a la **zona metropolitana de Guadalajara**, priorizando la **escalabilidad, estabilidad y modularidad** de la arquitectura. A futuro, se proyecta una expansión progresiva hacia otras regiones de México, manteniendo los mismos estándares de calidad, seguridad y rendimiento.

Finalmente, la propuesta actual contempla únicamente el **canal web responsive** como medio principal de interacción. Las **aplicaciones móviles nativas** quedan fuera del alcance del presente **MVP (Minimum Viable Product)**.

1.3 Definiciones, acrónimos y abreviaciones

Este apartado contiene las definiciones de todos los términos, acrónimos y abreviaturas utilizados en este documento.

Término / Acrónimo	Definición
API	*Application Programming Interface* — Conjunto de métodos y endpoints que permiten la comunicación entre componentes del sistema.
AWS	*Amazon Web Services* — Proveedor de servicios en la nube utilizado para geocodificación (Amazon Location Service) y correo transaccional (SES).
DB	*Database* — Base de datos relacional PostgreSQL empleada para el almacenamiento persistente de información.
MVP	*Minimum Viable Product* — Versión mínima funcional del sistema, centrada en validar las funcionalidades principales.
REST	*Representational State Transfer* — Estilo arquitectónico para la comunicación entre cliente y servidor mediante HTTP.
S3	*Simple Storage Service* — Servicio de almacenamiento de objetos utilizado para guardar archivos multimedia.

SDD	*Software Design Description — Documento que arquitectura y las interacciones del sistema*
SRS	*Software Requirements Specification — Documento que define los requerimientos funcionales y no funcionales del sistema.*
Stripe Connect	*Pasarela de pago utilizada para procesar anticipos, liquidaciones y comisiones.*
CDN	*Content Delivery Network — Red de distribución de contenido usada para optimizar la entrega de recursos estáticos (imágenes, scripts, estilos).*

1.4 Referencias

1. **SRS — ReparaYa v1.0 (2025)** — Especificación de Requerimientos de Software asociada a este SDD.
2. IEEE SDDTemplate (https://github.com/jam01/SDD-Template/blob/master/template.md)
3. **W3C Web Content Accessibility Guidelines (WCAG) 2.1 AA** — Accessibility standard for web interfaces.
4. **Stripe API v2024-08** — Stripe Checkout and Connect Express documentation.
5. **Amazon Location Service Developer Guide** — Geocoding and map services documentation.
6. ***Amazon Simple Email Service (SES)*** — *Transactional email delivery and configuration guide.*

1.5 Descripción general del documento

Este documento está estructurado conforme a la norma **IEEE 1016**, con el propósito de ofrecer una descripción técnica clara, modular y trazable del sistema **ReparaYa — MVP Web**.

Las secciones que lo componen son las siguientes:

- **Sección 1 – Introducción:** Presenta el propósito, alcance, definiciones, referencias y visión general del documento.

- **Sección 2 – Descripción general del diseño:** Describe la arquitectura global del sistema, los patrones de diseño aplicados, las capas principales y los componentes de software.

- **Sección 3 – Diseño detallado:** Define la estructura de datos, diagramas de clases, modelos de entidad-relación, y flujos lógicos de procesos clave (autenticación, reserva, pago, mensajería, etc.).

- **Sección 4 – Interfaces externas:** Detalla las interacciones con sistemas de terceros, incluyendo Stripe, AWS y servicios de correo electrónico.

- **Sección 5 – Requisitos de rendimiento y seguridad:** Expone los lineamientos de calidad, escalabilidad, encriptación, autenticación y control de errores.

- **Sección 6 – Trazabilidad y futuras extensiones:** Establece la relación entre los componentes de diseño y los requisitos del SRS, así como las proyecciones de evolución del sistema.

2. Diseño

2.1 Perspectiva del sistema

El sistema **ReparaYa** es una plataforma web tipo *marketplace* que conecta a **clientes** que requieren servicios locales con **contratistas** que los ofrecen (por ejemplo, mantenimiento, plomería, electricidad o reparaciones domésticas).

La aplicación se implementa bajo una **arquitectura cliente–servidor de tres capas**, asegurando separación de responsabilidades, escalabilidad y mantenibilidad.

Capas principales del sistema:

1. **Capa de presentación (Frontend):**
   Implementada en **React** con **TailwindCSS**, responsable de la interfaz web responsive. Maneja la interacción con el usuario final, validación de formularios y consumo de API REST.

2. **Capa de negocio (Backend):**
   Desarrollada en **Node.js**, donde reside la lógica de negocio. Gestiona autenticación, reservas, pagos, calificaciones, mensajería y operaciones administrativas.

3. **Capa de datos (Persistencia):**
   Basada en **PostgreSQL** y gestionada mediante **Prisma ORM**, responsable del almacenamiento persistente y la integridad de la información.

Servicios externos integrados:

- **Stripe (Checkout + Connect Express):** Procesa anticipos, liquidaciones y comisiones; recibe y maneja webhooks de pago.

- **Amazon Location Service:** Permite geocodificación y cálculo de distancias mediante la fórmula de Haversine.

- **AWS SES / SMTP:** Envía correos electrónicos transaccionales y notificaciones al usuario.

El sistema está diseñado para operar en la **nube (AWS región México Central)**, utilizando prácticas de *DevOps* para despliegue continuo y monitoreo básico.

2.2 Objetivos del diseño

El diseño del sistema ReparaYa persigue los siguientes objetivos técnicos:

- **Escalabilidad:** El sistema debe soportar crecimiento progresivo en número de usuarios, servicios y transacciones sin comprometer el rendimiento.

- **Mantenibilidad:** facilitar la comprensión y modificación del código a través de una estructura modular y desacoplada..

- **Seguridad:** Implementación de autenticación basada en JWT, cifrado de contraseñas (bcrypt), uso de HTTPS y conformidad con la LFPDPPP para protección de datos personales.

- **Disponibilidad**: El sistema deberá mantener una operación continua y estable, con manejo de errores controlado en la capa backend.

- **Extensibilidad:** Facilitar la incorporación futura de módulos como apps móviles, programas de fidelización y pagos internacionales.

2.3 Patrones arquitectónicos

En el siguiente apartado se mostrarán los patrones de diseño utilizados en el sistema.

Tipo	Patrón	Descripción	Justificación
Arquitectónico	N-Capas	Divide la aplicación en presentación, negocio y datos.	Aumenta la mantenibilidad y escalabilidad.
Diseño	MVC (Model-View-Controller)	Organiza la lógica del backend Express separando rutas, controladores y modelos.	Simplifica el desarrollo y pruebas.
Diseño	Repository Pattern	Aísla la lógica de acceso a datos dentro de repositorios conectados a Prisma.	Facilita la sustitución o actualización del ORM.
Diseño	Inyección de dependencias	Permite inyectar servicios en controladores.	Mejora testabilidad y reduce acoplamiento.

Comportamiento	Eventos	Emite y escucha eventos del sistema (pagos, reservas, notificaciones).	Sincroniza procesos sin acoplar módulos.
Estructural	DTO (Data Transfer Object)	Define estructuras para enviar y recibir datos por la API.	Garantiza consistencia y validación de entrada.

2.4 Componentes principales del sistema

El sistema se compone de los siguientes módulos funcionales implementados en la API de Node.js y Express.

Módulo	Descripción
AuthController / AuthService	Gestiona registro, inicio de sesión, verificación de identidad y emisión de tokens JWT.
UserController / UserService	Administra perfiles, roles, ubicación y validación de documentos.

ServiceController / ServiceService	Permite crear, editar y eliminar servicios ofrecidos por contratistas, además de consultar categorías y disponibilidad.
BookingController / BookingService	Controla la creación, actualización y cancelación de reservas, así como sus estados.
PaymentController / PaymentService **ChatController / ChatService**	Integra la API de Stripe para gestionar anticipos y liquidaciones, incluyendo el manejo de webhooks. Gestiona la mensajería entre cliente y contratista dentro de una reserva.
RatingController / RatingService	Administra reseñas, calificaciones y promedio de contratistas.
AdminController / AdminService	Permite revisar reportes, disputas y moderar usuarios y categorías.
	Enviar correos electrónicos de notificaciones o confirmación mediante

Notification Service	AWS SES o SMTP.

2.5 Diagrama general del sistema

El modelo de clases del sistema **ReparaYa** (Figura 3.1) representa las entidades del dominio que intervienen en los procesos de publicación de servicios, reservas, pagos, reseñas y comunicación. El diseño sigue los principios de orientación a objetos y garantiza consistencia con el modelo relacional de la base de datos.

Relaciones principales:

- **Usuario** es clase base de cliente y contratista.
- **Contratista** ofrece múltiples servicios (1-N).
- **Cliente** genera varias Reservas (1-N).
- **Reserva** íntegra **Pago**, **Chat**, **Reseña** (1-1).
- **Servicio** se asocia a una o varias Reservas (1-N).

Clase	Descripción	Métodos principales
Usuario	Clase base que contiene los datos comunes de todos los usuarios.	`iniciarSesion()`, `cerrarSesion()`, `actualizarPerfil()`, `cambiarContraseña()`
Cliente	Subclase de Usuario que representa al consumidor del servicio.	`reservarServicio()`, `pagarReserva()`, `dejarReseña()`, `obtenerHistorial()`
Contratista	Subclase de Usuario que publica servicios y	`publicarServicio()`, `editarServicio()`, `eliminarServicio()`, `calcularRatingPromedi`

	recibe calificaciones, también puede contratar otros servicios.	o(), listarServicios()
Servicios	Entidad que representa los servicios publicados en la plataforma.	calcularCosto(), actualizarDisponibilidad(), cambiarTarifa(), obtenerDetalles()
Reserva	Controla el flujo de contratación entre cliente y contratista.	confirmarReserva(), cancelarReserva(), completarReserva(), calcularAnticipo(), actualizarEstado(
Pago	Gestiona los pagos y liquidaciones asociados a las reservas.	procesarPago(), reembolsarPago(), verificarEstado(), generarRecibo()
Chat	Canal de comunicación entre cliente y contratista asociado a una reserva	enviarMensaje(), obtenerMensajes(), cerrarChat(), marcarLeido()
Reseña	Permite calificar y dejar comentarios sobre los servicios recibidos.	guardarReseña(), eliminarReseña()

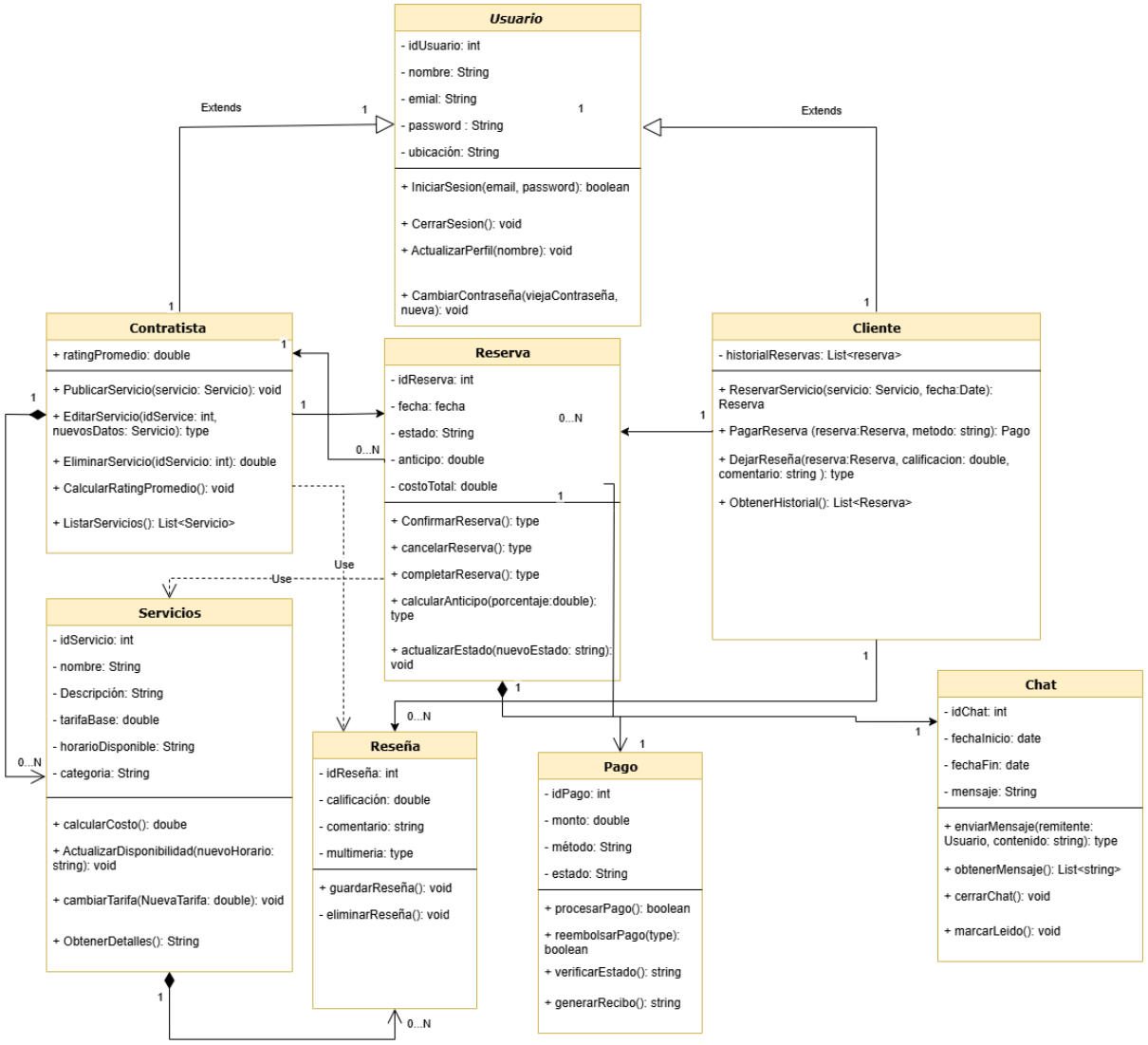

Figura 3.1 - Diagrama de clases UML.

2.6 Interfaces externas

Stripe (Checkout + Connect Express): sesiones separadas para anticipo y liquidación (MXN); onboarding de contratistas (Express); comisión desde el **markup 15%**; transferencia al contratista **≤24 h** tras liquidación; webhooks: checkout.session.completed, payment_intent.succeeded, charge.refunded, transfer.created, payout.paid, charge.dispute.created; **solo** comprobante interno (no recibos Stripe).

Amazon Location Service: Place Index para geocodificar; cálculo de distancia **Haversine**; mapa base estándar.

Email transaccional: SES o SMTP gratuito; remitente verificado (placeholder en MVP); fallas **solo registradas** para monitorizar.

2.7 Modelo de datos (MVP)

2.7.1 Diagrama ER — representación “normal” (mermaid)

```
erDiagram

    USER ||--o{ ADDRESS : tiene

    USER ||--|| CONTRACTOR_PROFILE : "si es contratista"

    USER ||--o{ RESERVATION : "como cliente"

    USER ||--o{ RESERVATION : "como contratista"

    USER ||--o{ MESSAGE : envia

    USER ||--o{ REVIEW : emite

    USER ||--o{ ADMIN_AUDIT_LOG : "acciones admin"


    CONTRACTOR_PROFILE ||--o{ CONTRACTOR_COVERAGE : cubre

    CONTRACTOR_COVERAGE }o--|| MUNICIPALITY : municipio


    CONTRACTOR_PROFILE ||--o{ SERVICE : publica

    SERVICE }o--|| CATEGORY : categoria

    SERVICE ||--o{ SERVICE_SCHEDULE : horario

    SERVICE ||--o{ SERVICE_PHOTO : fotos
```

RESERVATION }o--|| SERVICE : sobre

RESERVATION }o--|| ADDRESS : en

RESERVATION ||--o{ PAYMENT : pagos

RESERVATION ||--o{ MESSAGE : chat

RESERVATION ||--o{ REVIEW : reseñas

RESERVATION ||--o{ DISPUTE : disputa

2.7.2 (bis) Diagrama ER — versión explicada (matriz + narración)

Matriz de relaciones (cardinalidades + FK):

Origen	Relación	Destino	Card.	FK
USER	tiene	ADDRESS	1→*	ADDRESS.user_id → USER.id
USER	(si contratista) tiene	CONTRACTOR_PROFILE	1→1	CONTRACTOR_PROFILE.user_id → USER.id
CONTRACTOR_PROFILE	cubre	CONTRACTOR_COVERAGE	1→*	CONTRACTOR_COVERAGE.contractor_user_id → USER.id
CONTRACTOR_COVERAGE	pertenece a	MUNICIPALITY	*→1	CONTRACTOR_COVERAGE.municipality_id → MUNICIPALITY.id
CONTRACTOR_PROFILE	publica	SERVICE	1→*	SERVICE.contractor_user_id → USER.id

SERVICE	es de	CATEGORY	*→1	SERVICE.category_id → CATEGORY.id
SERVICE	tiene	SERVICE_SCHEDULE	1→*	SERVICE_SCHEDULE.service_id → SERVICE.id
SERVICE	tiene	SERVICE_PHOTO	1→*	SERVICE_PHOTO.service_id → SERVICE.id
USER (Cliente)	crea	RESERVATION	1→*	RESERVATION.client_user_id → USER.id
USER (Contratista)	atiende	RESERVATION	1→*	RESERVATION.contractor_user_id → USER.id
RESERVATION	sobre	SERVICE	*→1	RESERVATION.service_id → SERVICE.id
RESERVATION	en	ADDRESS	*→1	RESERVATION.address_id → ADDRESS.id
RESERVATION	genera	PAYMENT	1→*	PAYMENT.reservation_id → RESERVATION.id
RESERVATION	tiene chat	MESSAGE	1→*	MESSAGE.reservation_id → RESERVATION.id
USER	envía	MESSAGE	1→*	MESSAGE.sender_user_id → USER.id
RESERVATION	recibe	REVIEW	1→*	REVIEW.reservation_id → RESERVATION.id
USER	reseña a	USER	1→*	REVIEW.reviewer_user_id / reviewee_user_id → USER.id
RESERVATION	puede abrir	DISPUTE	1→*	DISPUTE.reservation_id → RESERVATION.id

USER (Admin)	registra	ADMIN_AUDIT_LOG	1→*	ADMIN_AUDIT_LOG.admin_user_id → USER.id

Narración (flujo): USER es origen; CONTRACTOR_PROFILE extiende a contratistas y define política fuera de municipio; CONTRACTOR_COVERAGE lista MUNICIPALITY; cada contratista publica SERVICE (CATEGORY, SCHEDULE, PHOTO). El cliente aporta ADDRESS. RESERVATION enlaza cliente, contratista, servicio y dirección; de ella cuelgan PAYMENT, MESSAGE, REVIEW y DISPUTE. Acciones de Admin se registran en ADMIN_AUDIT_LOG.

2.7.3 Entidades/Enums/Reglas/Índices

2.7.3.1 Esquema de datos completo (tablas, campos, tipos, constraints)

> Tipos pensados para **PostgreSQL**. `numeric(12,2)` para montos MXN. `timestamp with time zone` donde aplique. Reglas de negocio replicadas en capa aplicación.

Tabla `user`

- **id** `uuid` **PK**
- **email** `citext` **UNIQUE NOT NULL**
- **password_hash** `text` **NOT NULL**
- **name** `varchar(120)` **NOT NULL**
- **phone** `varchar(20)` NULL
- **role** `enum('ADMIN','CONTRACTOR','CLIENT')` **NOT NULL**
- **email_verified_at** `timestamptz` NULL
- **blocked** `boolean` **DEFAULT false**
- **created_at**, **updated_at** `timestamptz`

Índices: `idx_user_email_unique`, `idx_user_role`, `idx_user_blocked`
Checks: `chk_user_role_valid`

Tabla `contractor_profile`

- **user_id** `uuid` **PK FK→user.id**
- **accepts_outside** `boolean` **DEFAULT false**
- **outside_markup_percent** `smallint` **NOT NULL DEFAULT 0** (valores permitidos **0/10/20**)
- **about** `varchar(300)` NULL

Checks: `chk_outside_markup IN (0,10,20)`
Índices: `idx_contractor_profile_outside`

Tabla `municipality` (catálogo ZMG)

- **id** `serial` **PK**
- **name** `varchar(80)` **UNIQUE NOT NULL**

Tabla `contractor_coverage`

- **id** `uuid` **PK**
- **contractor_user_id** `uuid` **FK→user.id NOT NULL**
- **municipality_id** `int` **FK→municipality.id NOT NULL**
- **is_primary** `boolean` **DEFAULT false**

Únicos: `uq_coverage_contractor_muni (contractor_user_id, municipality_id)`
Índices: `idx_coverage_contractor`, `idx_coverage_municipality`

Tabla `address`

- **id** `uuid` **PK**
- **user_id** `uuid` **FK→user.id NOT NULL**
- **street** `varchar(140)` **NOT NULL**
- **ext_number** `varchar(20)` **NOT NULL**
- **int_number** `varchar(20)` NULL
- **neighborhood** `varchar(140)` NULL
- **municipality_id** `int` **FK→municipality.id NOT NULL**
- **postal_code** `varchar(10)` **NOT NULL**
- **lat** `decimal(10,7)` NULL
- **lng** `decimal(10,7)` NULL
- **is_default** `boolean` **DEFAULT false**

Índices: `idx_address_user`, `idx_address_municipality`

Tabla `category`

- **id** `serial` **PK**
- **name** `varchar(100)` **UNIQUE NOT NULL**

Tabla `service`

- **id** `uuid` **PK**
- **contractor_user_id** `uuid` **FK→user.id NOT NULL**
- **category_id** `int` **FK→category.id NOT NULL**
- **title** `varchar(100)` **NOT NULL**

- **description** `varchar(1000)` **NOT NULL**
- **hours_estimated** `numeric(3,1)` **NOT NULL** (rango recomendado **0.5–12.0**)
- **price_base_B** `numeric(12,2)` **NOT NULL CHECK (price_base_B > 0)**
- **active** `boolean` **DEFAULT true**
- **created_at**, **updated_at** `timestamptz`

Índices: `idx_service_contractor`, `idx_service_category`, `idx_service_active`

Tabla `service_schedule`

- **id** `uuid` **PK**
- **service_id** `uuid` **FK→service.id NOT NULL**
- **day_of_week** `smallint` **NOT NULL** CHECK (0 ≤ day_of_week ≤ 6) // 0=Dom … 6=Sáb
- **start_time** `time` **NOT NULL**
- **end_time** `time` **NOT NULL**

Checks: `chk_schedule_start_before_end (start_time < end_time)`
Índices: `idx_schedule_service_day`

Tabla `service_photo`

- **id** `uuid` **PK**
- **service_id** `uuid` **FK→service.id NOT NULL**
- **url** `text` **NOT NULL**
- **size_mb** `numeric(4,2)` **NOT NULL CHECK (size_mb <= 5.00)**

- **created_at** `timestamptz`

Únicos: `uq_service_photo_url (service_id, url)`
Regla UI: máximo **5** fotos por servicio.

Tabla `reservation`

- **id** `uuid` **PK**
- **client_user_id** `uuid` **FK→user.id NOT NULL**
- **contractor_user_id** `uuid` **FK→user.id NOT NULL**
- **service_id** `uuid` **FK→service.id NOT NULL**
- **address_id** `uuid` **FK→address.id NOT NULL**
- **start_at** `timestamptz` **NOT NULL**
- **hours_estimated** `numeric(3,1)` **NOT NULL** (copia del servicio)
- **status** `enum('CREATED','ON_ROUTE','ON_SITE','IN_PROGRESS','COMPLETED','NO_SHOW','DISPUTED','CANCELED')` **NOT NULL**
- **recargo_percent** `smallint` **NOT NULL DEFAULT 0** CHECK (recargo_percent IN (0,10,20))
- **markup_percent** `smallint` **NOT NULL DEFAULT 15**
- **price_base_B** `numeric(12,2)` **NOT NULL**
- **price_base_adjusted_Bp** `numeric(12,2)` **NOT NULL** // B′ = B × (1+recargo)
- **price_public_P** `numeric(12,2)` **NOT NULL** // P = 1.15 × B′
- **anticipo_amount** `numeric(12,2)` **NOT NULL** // 0.20 × P
- **liquidacion_amount** `numeric(12,2)` **NOT NULL** // 0.80 × P
- **currency** `char(3)` **NOT NULL DEFAULT 'MXN'**

- **created_at**, **updated_at** `timestamptz`

Índices: `idx_reservation_client_start`, `idx_reservation_contractor_start`, `idx_reservation_status`

Tabla `payment`

- **id** `uuid` **PK**
- **reservation_id** `uuid` **FK→reservation.id NOT NULL**
- **type** `enum('ANTICIPO','LIQUIDACION','REFUND')` **NOT NULL**
- **status** `enum('INITIATED','SUCCEEDED','FAILED')` **NOT NULL**
- **amount** `numeric(12,2)` **NOT NULL**
- **stripe_checkout_session_id** `varchar(120)` NULL
- **stripe_payment_intent_id** `varchar(120)` NULL
- **stripe_charge_id** `varchar(120)` NULL
- **stripe_transfer_id** `varchar(120)` NULL
- **created_at**, **updated_at** `timestamptz`

Índices: `idx_payment_reservation`, `idx_payment_type_status`

Tabla `message`

- **id** `uuid` **PK**
- **reservation_id** `uuid` **FK→reservation.id NOT NULL**
- **sender_user_id** `uuid` **FK→user.id NOT NULL**
- **content** `text` **NOT NULL**
- **created_at** `timestamptz` **NOT NULL**

Retención: eliminación automática de registros **> 7 días**.

Tabla `review`

- **id** `uuid` **PK**
- **reservation_id** `uuid` **FK→reservation.id NOT NULL**
- **reviewer_user_id** `uuid` **FK→user.id NOT NULL**
- **reviewee_user_id** `uuid` **FK→user.id NOT NULL**
- **stars** `smallint` **NOT NULL CHECK (stars BETWEEN 1 AND 5)`
- **text** `varchar(500)` NULL
- **status** `enum('ACTIVE','REMOVED_BY_ADMIN')` **NOT NULL DEFAULT 'ACTIVE'`
- **removed_reason** `varchar(280)` NULL
- **created_at** `timestamptz` **NOT NULL**

Únicos: `uq_review_unique_per_party (reservation_id, reviewer_user_id)`
Índices: `idx_review_reviewee_created`

Tabla `dispute`

- **id** `uuid` **PK**
- **reservation_id** `uuid` **FK→reservation.id NOT NULL**
- **initiator_user_id** `uuid` **FK→user.id NOT NULL**
- **status** `enum('OPEN','RESOLVED_UPHOLD_CLIENT','RESOLVED_UPHOLD_CONTRACTOR')` **NOT NULL DEFAULT 'OPEN'`
- **resolution_notes** `varchar(1000)` NULL

- **created_at** `timestamptz` **NOT NULL**
- **resolved_at** `timestamptz` NULL

Tabla `admin_audit_log` (lite)

- **id** `uuid` **PK**
- **admin_user_id** `uuid` **FK→user.id NOT NULL**
- **action_type** `enum('DELETE_REVIEW','BLOCK_ACCOUNT','PASSWORD_RESET','RESOLVE_DISPUTE')` **NOT NULL**
- **target_type** `varchar(40)` **NOT NULL** // 'REVIEW' | 'USER' | 'DISPUTE' | 'RESERVATION'
- **target_id** `uuid` **NOT NULL**
- **reason** `varchar(280)` NULL
- **created_at** `timestamptz` **NOT NULL**

Índices: `idx_audit_admin_created`, `idx_audit_target`

Notas transversales (3.4.2)

- **Moneda:** `MXN` fija en MVP.
- **Redondeo:** todos los totales a **2 decimales**.
- **Idempotencia de pagos:** basada en `stripe_payment_intent_id` y `event.id` de webhooks.
- **Integridad de estados:** transiciones válidas validadas en aplicación (BR-003) + constraints de negocio en endpoints.
- **Borrado lógico:** reseñas moderadas (`status='REMOVED_BY_ADMIN'`) y log en `admin_audit_log`.