Especificación Técnica: Sistema de Gestión de Talleres Automotrices

Versión del Documento: 1.0

Fecha: 16 de Diciembre de 2025

1. Resumen Ejecutivo

El presente documento detalla la arquitectura, diseño y decisiones técnicas fundamentales para el desarrollo del **Sistema de Gestión de Talleres**. Este sistema ha sido concebido bajo estándares de ingeniería de software robustos, priorizando la escalabilidad, la mantenibilidad y la integridad de los datos financieros y operativos.

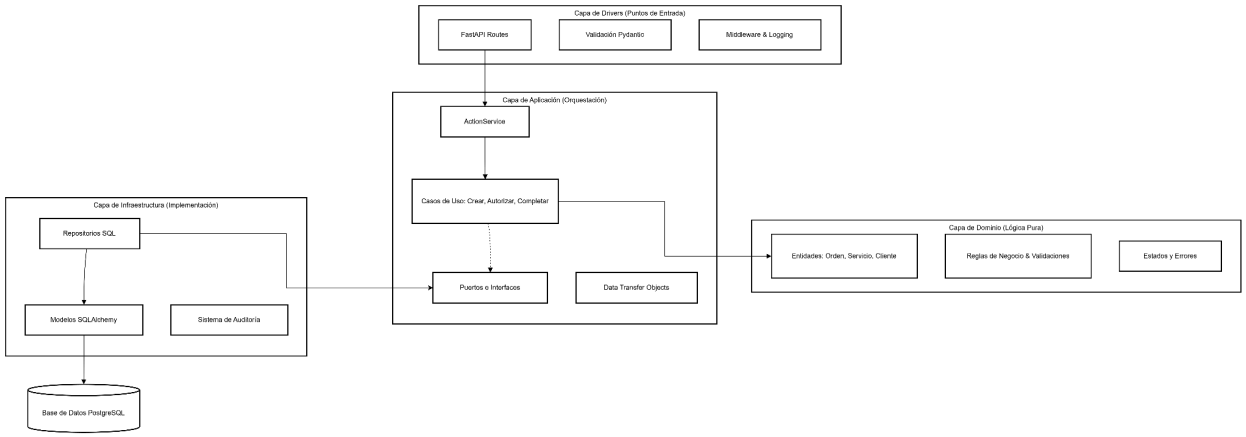
El núcleo del diseño se basa en una **Arquitectura Hexagonal (Ports & Adapters)**, lo que permite aislar la lógica de negocio de las dependencias externas como bases de datos o interfaces de usuario, garantizando que el sistema sea agnóstico a la infraestructura y altamente testeable.

2. Arquitectura del Sistema

Para garantizar el desacoplamiento y la flexibilidad tecnológica, hemos estructurado el sistema en cuatro capas concéntricas, donde las dependencias siempre apuntan hacia el interior (hacia la lógica de negocio).

Diagrama de Capas (Arquitectura Hexagonal)

El siguiente esquema ilustra cómo los "Drivers" (puntos de entrada) se comunican con el "Domain" (núcleo) a través de la capa de "Application".



Principios Rectores

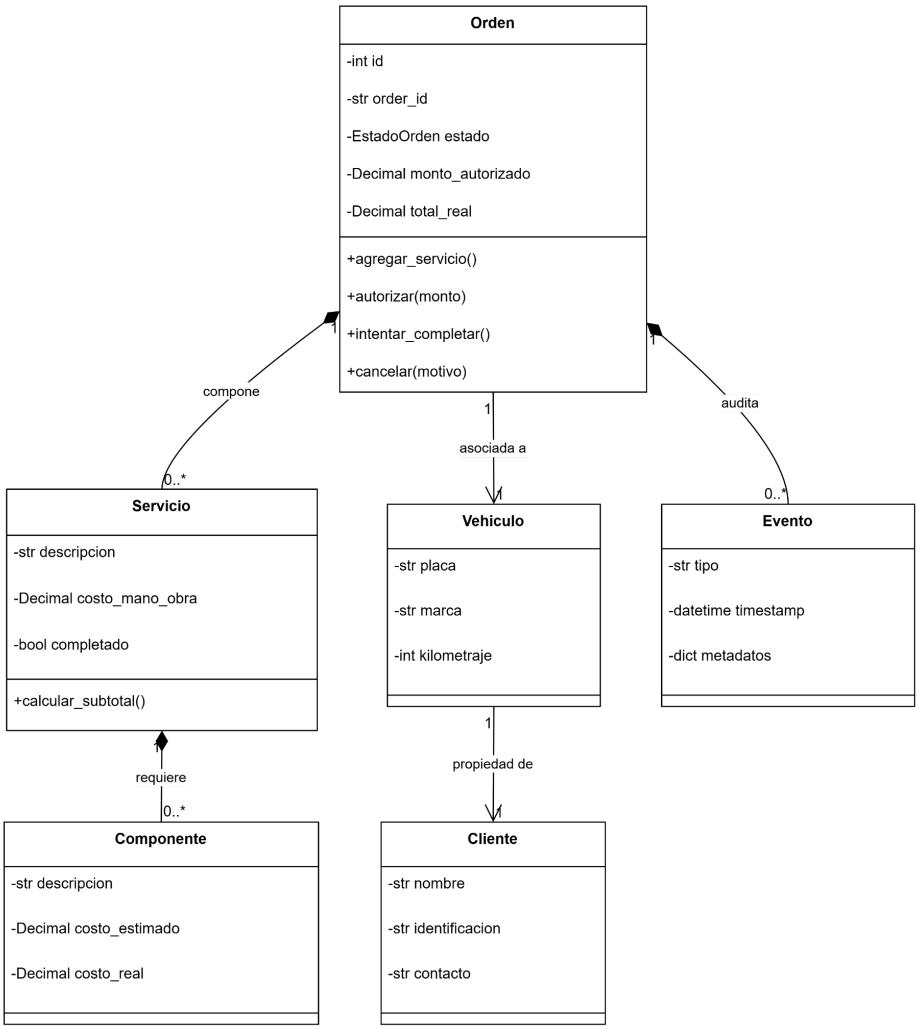
La arquitectura se sustenta en los siguientes pilares de diseño SOLID:

- **Inversión de Dependencias:** El dominio define las interfaces (puertos) que la infraestructura debe implementar, invirtiendo el control tradicional.
- **Responsabilidad Única:** Cada módulo y clase tiene un propósito específico y acotado.
- **Separation of Concerns:** Clara distinción entre la lógica de presentación, orquestación, negocio y persistencia.

3. Modelo del Dominio

El corazón del sistema reside en su modelo de dominio, el cual encapsula todas las reglas de negocio y comportamientos. La entidad **Orden** actúa como el *Agregado Raíz*, garantizando la consistencia de sus entidades internas (Servicios, Componentes y Eventos).

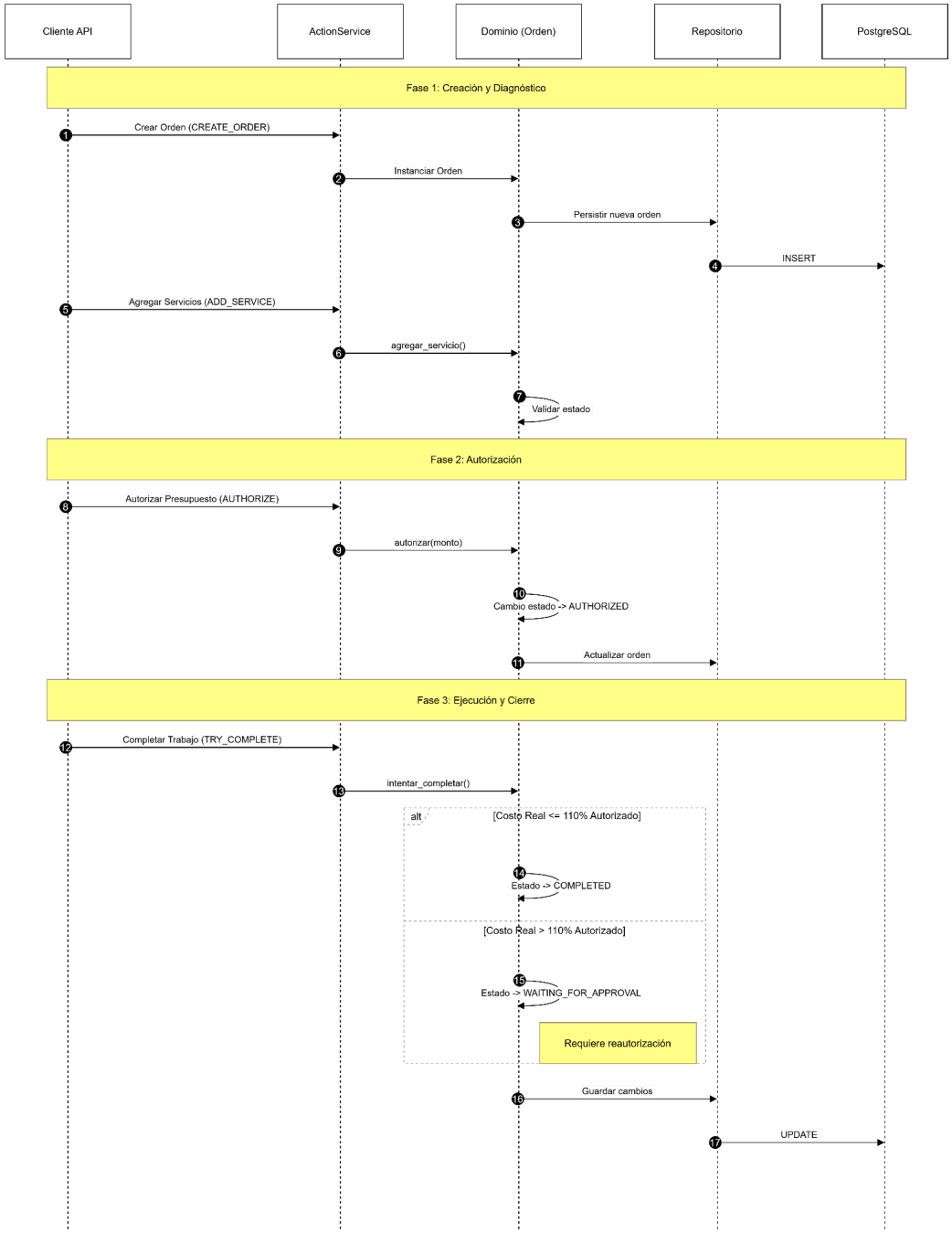
Diagrama de Clases



4. Flujo de Procesos y Secuencia

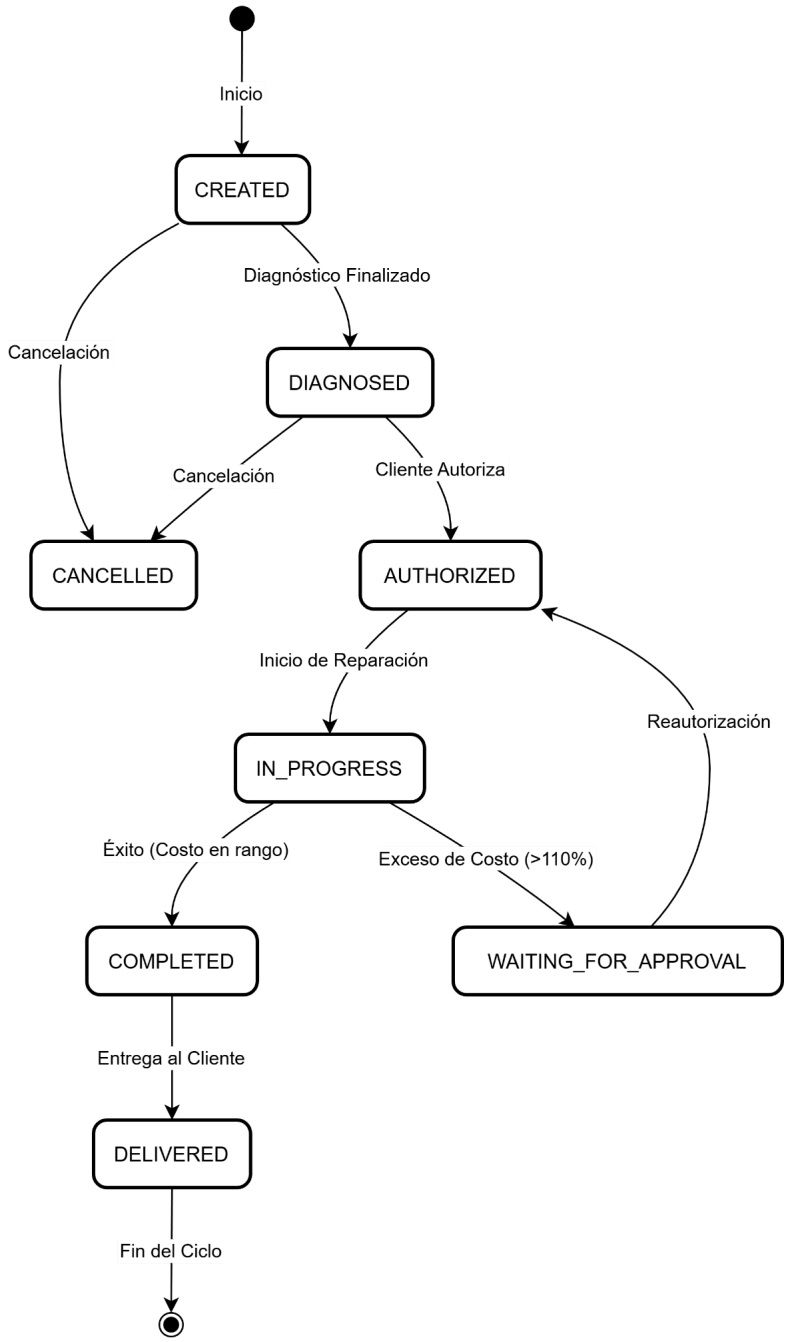
El sistema maneja un flujo de trabajo transaccional. A continuación se detalla el ciclo de vida completo de una orden, desde su creación hasta la entrega, destacando la interacción entre los componentes para asegurar la integridad de los datos.

Secuencia: Ciclo de Vida de la Orden



5. Máquina de Estados (Lógica de Negocio)

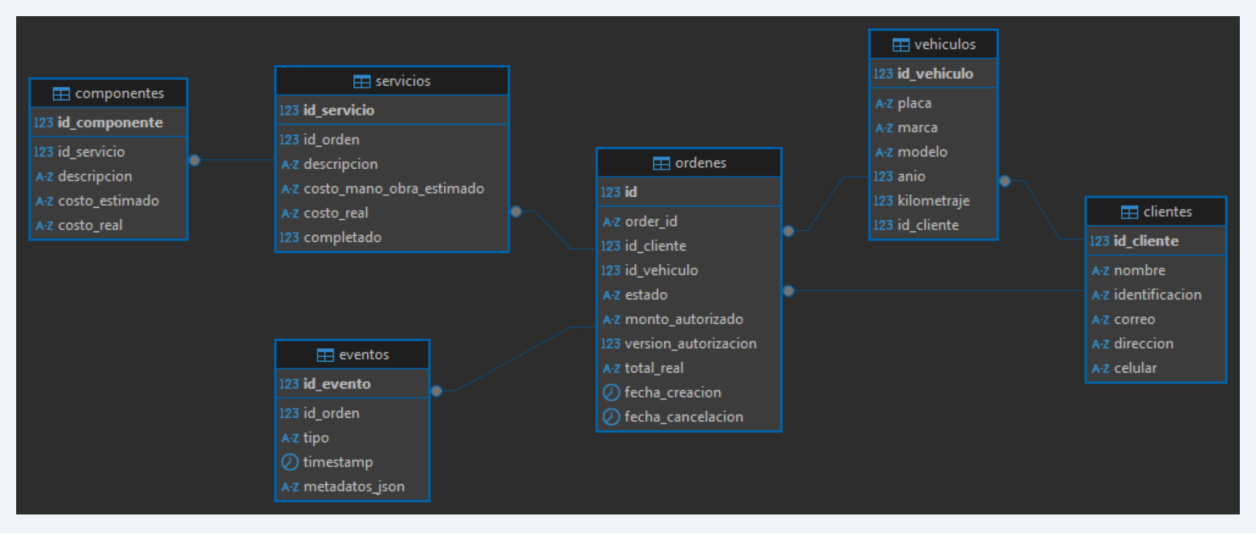
Para garantizar que una orden nunca se encuentre en una situación inconsistente, se ha implementado una máquina de estados finitos estricta. Las transiciones solo son posibles si se cumplen las precondiciones de validación.



6. Modelo de Datos y Persistencia

El esquema de base de datos relacional está diseñado para mantener la integridad referencial y optimizar las consultas transaccionales.

Diagrama Entidad-Relación (ERD)



Decisiones de Modelado:

- **Tipos de Datos:** Uso estricto de tipos DECIMAL para todos los campos monetarios, evitando errores de redondeo de punto flotante.
- **Identificadores:** Uso de order_id (UUID o formato legible) como clave de negocio única, separada de la clave primaria autoincremental (id) para optimización de índices.

7. Patrones de Diseño Aplicados

Para resolver problemas comunes de desarrollo de software, hemos aplicado los siguientes patrones:

1. **Repository Pattern:** Abstrae el acceso a datos. Permite cambiar la base de datos (por ejemplo, de PostgreSQL a otra) sin tocar una sola línea de la lógica de negocio.
2. **Unit of Work:** Gestiona las transacciones atómicas. Asegura que si una operación compleja falla (ej. actualizar inventario y orden), se haga rollback de todo el proceso.
3. **Command Pattern:** Encapsula las solicitudes como objetos (CrearOrden, Autorizar). Esto facilita la creación de colas de tareas, logs de auditoría y operaciones de "deshacer".
4. **Mapper Pattern:** Utilizado para transformar los datos entre las distintas capas (JSON $\leftrightarrow$ DTO $\leftrightarrow$ Entidad $\leftrightarrow$ Modelo SQL), manteniendo las capas limpias.

8. Decisiones Técnicas Clave

A continuación, exponemos la racionalidad detrás de las decisiones técnicas más relevantes:

- Manejo Financiero (Decimal & Banker's Rounding):
  Se decidió utilizar Decimal de Python y el redondeo "Half-Even" (o del banquero). Esto es estándar en aplicaciones financieras para minimizar el sesgo estadístico en sumas acumulativas de dinero.
- Procesamiento por Lotes (Batch Processing):
  El endpoint /commands está diseñado para recibir y procesar múltiples comandos en una sola petición HTTP. Esto reduce drásticamente la latencia de red en operaciones complejas y mejora la experiencia de usuario en interfaces lentas.
- Estrategia "Find-or-Create":
  Para Clientes y Vehículos, el sistema verifica su existencia antes de intentar crearlos. Esto evita duplicidad de datos y simplifica la lógica del cliente (frontend), que no necesita saber los IDs de antemano.
- **Bilingüismo (Código vs API):**
  - **Código Interno:** Español (alineado con el idioma nativo del equipo de desarrollo y los términos del dominio del negocio).
  - **API Externa:** Inglés (estándar de la industria JSON REST), facilitando integraciones con terceros.

9. Calidad y Pruebas

La estabilidad del sistema se valida mediante una estrategia de pruebas piramidal:

- **Pruebas Unitarias (Domain):** Cobertura del ~90%. Se verifica exhaustivamente las reglas de negocio, cálculos de impuestos y transiciones de estado sin depender de bases de datos.
- **Pruebas de Integración:** Verifican que los Repositorios y la Base de Datos interactúan correctamente.
- **Pruebas End-to-End:** Simulan el comportamiento de un usuario real interactuando con la API.

10. Conclusión

El diseño presentado ofrece una solución sólida y profesional para la gestión de talleres. La adopción de la **Arquitectura Hexagonal** protege la inversión a largo plazo, permitiendo que el software evolucione junto con el negocio. Las estrictas validaciones de dominio aseguran que la información financiera y operativa sea siempre confiable, posicionando al sistema como una herramienta crítica y robusta para la operación diaria.