

---

# **Especificación de requisitos de software**

**Para**

## **Public Transit Agency**

**Versión 1.0 por aprobar**

**Preparado por:**

**Alejandro Pedro Steinman Cuesta, Mario Julio Wilches, Andrés  
Rubiano Marrugo**

**Universidad Tecnológica de Bolívar**

**5 de junio de 2025**

## **Tabla de contenido**

<b>1. Introducción</b>	<b>1</b>
Beneficios Esperados	1
Objetivos Principales	1
Meta General	2
<b>2. Descripción general</b>	<b>2</b>
2.1 Características del producto	2
2.2 Clases de usuarios y características	2
2.3 Limitaciones de diseño y aplicación	4
Limitaciones Técnicas y de Negocio	4
Definición del Alcance	5
Funcionalidades Excluidas (Fuera del Alcance)	6
Restricciones del Alcance	6
Definición de Métricas de Éxito	7
Métricas Técnicas	7
Métricas de Adopción y Uso	7
Métricas de Impacto en la Operación	8
2.4 Supuestos y dependencias	8
<b>3. Características del sistema</b>	<b>9</b>
3.1 Gestión de Flotas y Rutas	10
3.1.1 Descripción y prioridad	10
3.1.2 Secuencias de estímulo/respuesta	10
3.1.3 Requisitos Funcionales	10
3.2. Administración de Conductores y Operarios	11
3.2.1 Descripción y Prioridad	11
3.2.2 Secuencias de Estímulo/Respuesta	11
3.2.3 Requisitos Funcionales	11
3.3 Portal de Información para Pasajeros	11
3.3.1 Descripción y Prioridad	11
3.3.2 Secuencias de Estímulo/Respuesta	11
3.3.3 Requisitos Funcionales	12
3.4 Monitoreo en Tiempo Real de Vehículos	12
3.4.1 Descripción y Prioridad	12
3.4.2 Secuencias de Estímulo/Respuesta	12
3.4.3 Requisitos Funcionales	12
3.5 Gestión de Rutas y Horarios	12
3.5.1 Descripción y Prioridad	12
3.5.2 Secuencias de Estímulo/Respuesta	12

3.5.3 Requisitos Funcionales	13
3.6 Reportes y Analítica	13
3.6.1 Descripción y Prioridad	13
3.6.2 Secuencias de Estímulo/Respuesta	13
3.6.3 Requisitos Funcionales	13
3.7 Documentación de Microservicios	13
3.7.1. Microservicio: Pagos	13
3.7.2. Microservicio: TransportUnit	17
3.7.3. Microservicio: EstadoMantenimiento	22
3.7.4. Microservicio: Incidencia	26
3.7.5. Microservicio: Ticket	31
3.7.6. Microservicio: Ruta	35
3.7.7. Microservicio: Horario	39
3.7.8. Microservicio: Turno	43
3.7.9. Microservicio: Tarjeta	48
3.7.10. Microservicio: Parada	52
3.7.11. Microservicio: Tipo-Tarjeta	56
3.7.12. Microservicio: Mantenimiento	60
3.7.13. Microservicio: Movimiento	64
3.7.14. Microservicio: TipoMovimiento	69
3.7.15. Microservicio: Precio	74
3.7.16. Microservicio: TipoTransporte	79
3.7.17. Microservicio: RolUsuario	85
3.7.18. Microservicio: Usuario	91
3.7.19. Microservicio: PQR	97
3.7.20. Microservicio: Rendimiento	104
3.7.21. Microservicio: Asistencia	108
3.8 Prueba de Integración	114
3.8.1 Parámetros de Éxito	114
3.8.2 Imágenes del paso a paso	114
3.9 Prueba de Seguridad	116
3.9.1. Introducción	116
3.9.2. Datos Generales del Escaneo	116
3.9.3. Resumen Ejecutivo	117
3.9.4. Detalle de Hallazgos	117
3.9.4.2. Alertas Informativas	119
3.9.5. Evidencia Técnica	120
3.9.6. Recomendaciones Técnicas Generales	121

3.9.7. Referencias	122
3.9.8. Conclusión	122
3.10 Prueba de Estrés	122
<b>4. Otros requisitos No funcionales</b>	<b>124</b>
4.1 Requerimientos de desempeño	124
4.2 Requisitos de seguridad	124
4.3 Atributos de calidad del software	125
<b>5. Requerimientos de Sistemas</b>	<b>125</b>
5.1 Arquitectura orientada a Servicios	126
5.2 Lenguajes de Programación	126
5.3 Base de Datos	126
5.4 Repositorio	126
<b>6. Sprints</b>	<b>127</b>
6.1. Sprint #1	127
6.1.1. Backlog	127
6.1.2. Reporte Semanal	128
6.2. Sprint #2	130
6.2.1. Backlog	130
6.2.2. Reporte Semanal	131
6.3. Sprint #3	134
6.3.1. Backlog	134
6.3.2. Reporte Semanal	135
6.4. Sprint #4	140
6.4.1. Backlog	140
6.4.2. Reporte Semanal	143
6.5. Sprint #5	147
6.5.1. Backlog	147
6.5.2. Reporte Semanal	152
6.6. Sprint #6	156
6.6.1. Backlog	156
6.6.2 Weekly Report	160
6.7. Sprint #7	163
6.7.1. Backlog	163
6.7.1. Weekly Report	165
<b>Apéndice A: Glosario</b>	<b>169</b>
<b>Apéndice B: Modelos de análisis</b>	<b>170</b>
<b>Apéndice C: Lista de problemas</b>	<b>178</b>

## **Historial de revisiones**

<b>Nombre</b>	<b>Fecha</b>	<b>Motivo de los cambios</b>	<b>Versión</b>
Actualización	24/03/2025	Finalización de Sprint #1, presentación de documentación para el cliente	0.2
Actualización	31/03/2025	Finalización de Sprint #2, presentación de documentación para el cliente	0.3
Actualización	21/04//2025	Finalización de Sprint #3, presentación de documentación para el cliente	0.4
Actualización	28/04//2025	Finalización de Sprint #4, presentación de documentación para el cliente	0.5
Actualización	4/05//2025	Finalización de Sprint #5, presentación de documentación para el cliente	0.6
Añadido y Actualización	16/05/2025	Finalización de Sprint #6, corrección de formato, documentación para el cliente	0.7
Actualización: Pruebas de integración, seguridad, estrés	29/05/2025	Finalización de Sprint #7, corrección de formato, documentación para el cliente	1.0
Adición: Prueba de Integración	5/06/2025	Se agrega y corrige contenido en la prueba de integración	1.1

# 1. Introducción

*El presente documento tiene como objetivo proporcionar una descripción detallada del proyecto de la Agencia de Transporte Público. En él se expondrán el propósito fundamental del proyecto, las características esenciales del sistema, sus interfaces, las funcionalidades que el sistema llevará a cabo, los parámetros bajo los cuales deberá operar y su capacidad de respuesta ante estímulos externos.*

*El proyecto se centra en el desarrollo de una arquitectura de software moderna para una agencia de transporte público que actualmente opera con sistemas heredados y aislados. El objetivo principal es superar las limitaciones de la infraestructura actual, que impide el compartir y el analizar eficientemente los datos entre los distintos departamentos. Esto se logrará mediante la implementación de una arquitectura basada en microservicios y un almacén de datos en la nube.*

*El software a desarrollar consiste en un conjunto de microservicios diseñados para encapsular funciones críticas de la agencia, como la programación, el despacho y el mantenimiento. Estos microservicios expondrán APIs estandarizadas, protegidas por una pasarela, que permitirán el acceso seguro a los datos operativos. Además, se implementará un almacén de datos en la nube para consolidar y analizar la información recopilada a través de estas APIs.*

## Beneficios Esperados

- **Mejora de la eficiencia operativa:** La capacidad de compartir y analizar datos en tiempo real permitirá optimizar rutas, horarios y mantenimiento, reduciendo retrasos y mejorando la utilización de los recursos.
- **Mejor planificación y previsión:** El análisis de datos consolidados permitirá prever la demanda, ajustar los servicios y planificar futuras expansiones de manera más precisa.
- **Incremento de la calidad del servicio:** La reducción de la congestión y los retrasos mejorará la experiencia del usuario y aumentará la satisfacción del cliente.
- **Flexibilidad y escalabilidad:** La arquitectura de microservicios permitirá una mayor flexibilidad para adaptarse a cambios futuros y escalar la infraestructura según sea necesario.

## Objetivos Principales

- Desarrollar microservicios para las funciones principales de la agencia.
- Exponer datos relevantes a través de APIs seguras.
- Implementar un almacén de datos en la nube para consolidar la información.
- Aplicar análisis avanzados para optimizar las operaciones de transporte.

## Meta General

*La meta general es transformar la infraestructura de software de la agencia, pasando de sistemas monolíticos a una arquitectura moderna y ágil que permita una gestión de datos eficiente y una mejora continua del servicio.*

## 2. Descripción general

*El producto que se va a desarrollar surge de la idea de crear un software generalizado de un sistema de transporte público lo que quiere decir que contendrá todas las características posibles para que pueda ser aplicado en cualquier ciudad que lo necesite.*

*El producto pretende ser un sustituto de los softwares de transporte público que se usan actualmente en la ciudad de Cartagena, y ofrecerá mejoras en el manejo y la personalización de las tarjetas, una mejor administración del mantenimiento y muchas otras más mejoras que serán tocadas en los siguientes apartados de esta documentación.*

### 2.1 Características del producto

**2.1.1.** *Tiene diferentes roles de acceso (Cliente y administrador) así como diferentes opciones de uso para cada uno.*

**2.1.2.** *Permite al usuario personalizar su tarjeta, esto conlleva a que también pueda ver su historial de viajes y transacciones.*

**2.1.3.** *Permite la elección del destino, lo que facilita al software mostrarle todas las rutas, y la más efectiva.*

**2.1.4.** *Permite conocer la hora de llegada con un porcentaje de error mínimo del vehículo en cuestión.*

**2.1.5.** *Facilita un horario estipulado por parámetros definidos por el administrador para registrar y notificar la fecha de los mantenimientos vehiculares.*

**2.1.6.** *Notificaciones si existe alguna eventualidad con el vehículo esperado.*

### 2.2 Clases de usuarios y características

#### 2.2.1.

- *Nombre : Cliente casual/Peatón*
- *Descripción: Será el mayor beneficiado de este nuevo sistema de transporte público, siendo que además de que puede hacer rastreo de su saldo, puede enterarse de toda*

*la información que necesite por medio de esta app, como ejemplos puntuales tendríamos; si ocurre un contratiempo le llegaría una notificación de retraso y porque sucedió (accidente, tráfico vehicular, etc.), la elección de la mejor ruta posible y junto a esta el tiempo de llegada, entre otros casos de uso.*

- *Frecuencia de uso: Se espera que el usuario tenga una frecuencia de uso **alta** de la aplicación, siendo que desde esta puede planear donde esperar su vehículo y en cuanto tiempo puede llegar.*
- *Conocimientos técnicos: Para el cliente casual, no se necesitan conocimientos técnicos previos sobre el manejo de la aplicación, ya que estos no tendrán un nivel de autoridad avanzado que necesite del mismo.*
- *Niveles de seguridad y privilegios: El nivel de seguridad para el cliente será **alto**, ya que se manejan cosas como los datos personales de la persona, y su saldo. Por otra parte sus privilegios serán **bajos**, los suficientes para manejar la aplicación base para el peatón.*
- *Experiencia: La experiencia necesaria será **baja**, ya que la aplicación para los clientes casuales es de naturaleza intuitiva.*

#### 2.2.2.

- *Nombre : Administrador*
- *Descripción: Será el principal usuario de la aplicación, siendo el responsable de dar las alertas, actualizar el estado de los vehículos, estipular el horario de mantenimiento, registrar nuevos los vehículos, el poder agendar mantenimientos de emergencia si lo ve necesario, además de todo esto, la persona que tenga el rol de administrador podrá acceder a las funciones base de la aplicación para peatones.*
- *Frecuencia de uso: Se espera que el administrador tenga una frecuencia de uso **muy alta** de la aplicación, siendo que desde esta puede básicamente hacer todo su trabajo.*
- *Conocimientos técnicos: El administrador debe tener un **alto** conocimiento en el manejo de la aplicación y de todas las capacidades que posee su rol en la aplicación.*
- *Niveles de seguridad y privilegios: El nivel de seguridad para el cliente será **muy alto**, se manejan los datos de la empresa en el uso de esta aplicación.*
- *Experiencia: A pesar de que la aplicación de es naturaleza intuitiva, cosas como agregar un nuevo vehículo o agendar mantenimientos son cosas un poco más complejas de las cuales se deben tener conocimientos previos en el manejo del modo administrador de otras aplicaciones, lo que quiere decir que el nivel de experiencia necesario es **muy alto**.*

## 2.3 Limitaciones de diseño y aplicación

### Limitaciones Técnicas y de Negocio

#### *Limitaciones Técnicas*

##### **a. Infraestructura y Recursos:**

- **Capacidad de los servidores:** La infraestructura en la nube o en servidores locales debe ser evaluada según costos y necesidades.

##### **b. Seguridad y Protección de Datos:**

- **Autenticación y permisos:** Diferentes roles de usuario (pasajero, administrador, supervisor) deben contar con controles de acceso bien definidos para evitar filtraciones de información.
- **Normativas de privacidad:** Cumplimiento con regulaciones como **GDPR** (en caso de expansión a Europa) o leyes locales sobre protección de datos personales en Latinoamérica.

#### *Limitaciones de Negocio*

##### **a. Regulaciones y Normativas**

- **Requisitos gubernamentales:** Dependiendo de la ciudad, puede ser obligatorio cumplir con normativas específicas de transporte, seguridad y accesibilidad.
- **Restricciones en el uso de datos:** En algunas ciudades, la recopilación de información sobre rutas, pagos y hábitos de los usuarios puede estar regulada.

##### **b. Presupuesto y Costos Operativos**

- **Inversión inicial:** Desarrollo, pruebas e implementación pueden requerir una inversión significativa, y el retorno de inversión puede no ser inmediato.

- **Mantenimiento del sistema:** Actualizaciones, servidores y soporte técnico representan costos recurrentes que deben ser considerados en el modelo de negocio

#### c. Limitaciones en la Expansión

- **Infraestructura de transporte variable:** Cada ciudad tiene diferentes necesidades de transporte, lo que puede dificultar la adaptación del software sin modificaciones importantes.
- **Saturación de usuarios:** En eventos especiales o temporadas altas, la alta demanda de transporte puede sobrecargar el sistema.

### Definición del Alcance

El desarrollo de un software de gestión para el transporte público que integre funcionalidades avanzadas para mejorar la experiencia del usuario y optimizar la administración de flotas. El sistema estará diseñado para ser adaptable a distintas ciudades, ofreciendo un control eficiente sobre rutas, horarios, mantenimiento y acceso a información en tiempo real.

### *Funcionalidades Incluidas*

#### Gestión de Flotas y Rutas

- Administración de vehículos y asignación de rutas en tiempo real.
- Monitoreo para conocer ubicación y estado de los vehículos.
- Creación y ajuste dinámico de la integridad de la base de datos, monitorio de los registros

#### Administración de Conductores y Operarios

- Registro de conductores y asignación de turnos.
- Control de asistencia y desempeño del personal.

- Generación de reportes sobre cumplimiento de horarios y eficiencia.

### Portal de Información para Pasajeros

- Consulta de horarios y disponibilidad de autobuses.
- Planificación de rutas con recomendaciones personalizadas.

### Gestión de Rutas y Horarios

- Creación, edición y eliminación de rutas.
- Asociación de rutas con horarios y vehículos.
- Acceso a información de rutas a través de la app.

### Reportes y Analítica

- Generación de reportes sobre operación, asistencia y desempeño.
- Análisis de tiempos de llegada y uso del servicio.

### Funcionalidades Excluidas (Fuera del Alcance)

- Pago directo dentro de la aplicación (se maneja con tarjetas preexistentes).
- Implementación de hardware adicional en vehículos (sensores avanzados, cámaras).
- Soporte para múltiples idiomas (inicialmente solo en español).

### Restricciones del Alcance

- **Disponibilidad geográfica:** El sistema se diseñará inicialmente para su implementación en Cartagena, pero con capacidad de expansión a otras ciudades con adaptaciones mínimas.
- **Regulaciones gubernamentales:** Se ajustará a las normativas vigentes para la gestión del transporte público.
- **Infraestructura tecnológica:** Dependerá de la conectividad a internet, servidores en la nube y GPS para su correcto funcionamiento.
- **Modelo de negocio:** Enfocado en la colaboración con empresas de transporte público y entidades gubernamentales.

## Definición de Métricas de Éxito

### Métricas Técnicas

#### Tiempo de Respuesta del Sistema

- Meta: Responder en menos de **2 segundos** a consultas de rutas y disponibilidad.

#### Disponibilidad del Sistema

- Meta: Mantener una disponibilidad del **99.5%** o superior durante el primer año de operación.

#### Precisión en los Tiempos de Llegada

- Meta: Reducir el margen de error en las estimaciones de llegada a **menos del 10%**.

#### Tiempo de Carga de Mapas y Datos en Tiempo Real

- Meta: No superar **3 segundos** en la carga de información de vehículos y rutas.

#### Seguridad y Protección de Datos

- Meta: Cumplir con los estándares de seguridad **ISO 27001** y encriptación de datos sensibles.

### Métricas de Adopción y Uso

#### Usuarios Registrados

- Meta: Alcanzar al menos **10,000 usuarios activos** en los primeros **6 meses**.

#### Frecuencia de Uso

- Meta: Que al menos el **70% de los usuarios activos** utilicen la app **más de 3 veces por semana**.

## Satisfacción del Usuario

- Meta: Obtener una calificación promedio de **4.5/5** en encuestas de satisfacción de pasajeros.

## Tiempo de Capacitación para Administradores y Conductores

- Meta: Que el **80% del personal** aprenda a usar el sistema en menos de **5 horas** de formación.

## Métricas de Impacto en la Operación

### Optimización de Rutas

- Meta: Reducir en al menos **15% los tiempos de espera** para los pasajeros.

### Reducción de Incidencias Operativas

- Meta: Disminuir los problemas de asignación de rutas en al menos **30%** en el primer año.

### Eficiencia del Mantenimiento Vehicular

- Meta: Asegurar que el **95% de los mantenimientos** se realicen en el plazo estipulado.

### Generación de Reportes

- Meta: Que los reportes operativos se generen en **menos de 10 segundos**.

## 2.4 Supuestos y dependencias

< Enumere los factores supuestos (en lugar de los hechos conocidos) que podrían afectar a los requisitos establecidos en la SRS. Pueden ser componentes comerciales o de terceros que se vayan a utilizar, problemas relacionados con el desarrollo o el entorno operativo, o limitaciones. El proyecto podría verse afectado si estos supuestos son incorrectos, no se comparten o cambian. Identifique también cualquier dependencia que tenga el proyecto de factores externos, como componentes de software que pretenda reutilizar de otro proyecto, a menos que ya estén documentados en otra parte (por ejemplo, en el documento de visión y alcance o en el plan del proyecto).>



### 3. Características del sistema

#### 3.1 Gestión de Flotas y Rutas

##### 3.1.1 Descripción y prioridad

*Esta funcionalidad permite monitorear y administrar en tiempo real la ubicación, disponibilidad y estado de los vehículos de transporte público. Optimiza rutas, reduce tiempos de espera y mejora la planificación operativa.*

**Prioridad:** Alta

##### 3.1.2 Secuencias de estímulo/respuesta

- a. *El administrador asigna rutas a los vehículos.*

**Entrada:** Configuración de nuevas rutas en el sistema.

**Respuesta:** Los conductores reciben sus asignaciones en la aplicación móvil.

- b. *El supervisor monitorea la ubicación de las unidades en tiempo real.*

**Entrada:** Accede al mapa del sistema.

**Respuesta:** Se muestra la ubicación actual y el estado de cada unidad.

- c. *Un usuario consulta la disponibilidad de autobuses cercanos.*

**Entrada:** Accede a la app de pasajeros y selecciona una ruta.

**Respuesta:** Se muestran los tiempos estimados de llegada y ubicación de los vehículos.

##### 3.1.3 Requisitos Funcionales

- **REQ-1:** El sistema debe permitir la gestión y asignación de rutas a los vehículos.
- **REQ-2:** Los supervisores deben poder monitorear la ubicación en tiempo real.
- **REQ-3:** Los pasajeros deben recibir información actualizada de horarios y rutas.
- **REQ-4:** Se deben generar reportes de desempeño y cumplimiento de rutas.

### 3.2. Administración de Conductores y Operarios

#### 3.2.1 Descripción y Prioridad

*Facilita la asignación de turnos, monitoreo de desempeño y gestión de asistencia del personal operativo, asegurando un servicio eficiente.*

- **Prioridad:** Alta

#### 3.2.2 Secuencias de Estímulo/Respuesta

- a. *El supervisor asigna turnos a los conductores.*

**Entrada:** Configuración de horarios en la plataforma.

**Respuesta:** Los conductores reciben su itinerario en la app.

- b. *Un conductor inicia sesión para registrar su turno.*

**Entrada:** Marca su inicio de jornada en la app.

**Respuesta:** El sistema registra la asistencia y asigna su ruta.

- c. *El sistema genera reportes de asistencia y desempeño.*

**Entrada:** Consulta de estadísticas por parte del supervisor.

**Respuesta:** Se muestra el historial de turnos y desempeño de los operarios.

#### 3.2.3 Requisitos Funcionales

- **REQ-5:** *El sistema debe permitir la asignación de turnos a los conductores.*
- **REQ-6:** *Debe registrar la asistencia y el cumplimiento de horarios.*
- **REQ-7:** *Los supervisores deben acceder a reportes de desempeño.*

### 3.3 Portal de Información para Pasajeros

#### 3.3.1 Descripción y Prioridad

*Proporciona a los pasajeros información en tiempo real sobre horarios, rutas, disponibilidad y tiempos de espera, mejorando la experiencia del usuario.*

- **Prioridad:** Media

#### 3.3.2 Secuencias de Estímulo/Respuesta

- a. *Un usuario consulta los horarios de los autobuses.*

**Entrada:** Accede a la app y selecciona una ruta.

**Respuesta:** Se muestran los horarios de llegada y disponibilidad de unidades.

- b.** El sistema envía alertas de retrasos o cambios en la ruta.

**Entrada:** Cambio en la disponibilidad de los vehículos.

**Respuesta:** Notificación push al usuario con la actualización.

- c.** Un usuario planifica su ruta con recomendaciones personalizadas.

**Entrada:** Introduce su punto de origen y destino.

**Respuesta:** El sistema sugiere la mejor ruta y combinación de transportes.

### 3.3.3 Requisitos Funcionales

- **REQ-8:** Debe mostrar información en tiempo real sobre disponibilidad de transporte.
- **REQ-9:** Debe enviar alertas a los usuarios sobre cambios en el servicio.
- **REQ-10:** Debe permitir a los usuarios planificar rutas personalizadas.

## 3.4 Monitoreo en Tiempo Real de Vehículos

### 3.4.1 Descripción y Prioridad

Muestra la ubicación y estado de los vehículos en tiempo real a través de GPS.

- **Prioridad:** Alta.

### 3.4.2 Secuencias de Estímulo/Respuesta

- a.** Los supervisores pueden visualizar información de cada unidad.

### 3.4.3 Requisitos Funcionales

- **REQ-13:** Alertas en caso de desvíos o detenciones prolongadas.

## 3.5 Gestión de Rutas y Horarios

### 3.5.1 Descripción y Prioridad

Permite definir y gestionar rutas, paradas y horarios de los vehículos.

- **Prioridad:** Alta.

### 3.5.2 Secuencias de Estímulo/Respuesta

- a.** Un administrador registra una nueva ruta con sus paradas y horarios.
- b.** El sistema almacena la información y la muestra a los operarios.
- c.** Los usuarios pueden consultar las rutas y horarios en la plataforma.

### 3.5.3 Requisitos Funcionales

- **REQ-14:** Creación, edición y eliminación de rutas.
- **REQ-15:** Asociación de rutas con horarios y vehículos.
- **REQ-16:** Visualización de rutas y horarios en la plataforma.

## 3.6 Reportes y Analítica

### 3.6.1 Descripción y Prioridad

Genera reportes detallados sobre operaciones, asistencia y desempeño del transporte.

- **Prioridad:** Media.

### 3.6.2 Secuencias de Estímulo/Respuesta

- a. Un administrador solicita un reporte de operaciones.
- b. El sistema genera un archivo con métricas clave.

### 3.6.3 Requisitos Funcionales

- **REQ-17:** Creación de reportes automáticos y personalizados.
- **REQ-18:** Visualización de métricas clave (tiempo de llegada, uso del servicio, etc.).
- **REQ-19:** Exportación de reportes en diferentes formatos.

## 3.7 Documentación de Microservicios

### 3.7.1. Microservicio: Pagos

#### FASE 1: Análisis y Diseño Funcional del Microservicio

- **Microservicio:** payment\_service
- **Tareas puntuales:**
  - **Identificar el bounded context:**
    - El dominio "Pago" incluye información del usuario, cantidad, método, tipo de vehículo, ID de tarjeta y estado del pago.
    - Se separa payment\_service de otros servicios relacionados (e.g., BillingService, NotificationService).

- **Definir la entidad principal:**
  - Entidad: Payment
  - Atributos: id, user\_id, payment\_quantity, payment\_method, vehicle\_type, card\_id, status, created\_at, transaction\_id
- **Delimitar funcionalidad del microservicio:**
  - CRUD completo para pagos.
  - Filtrar pagos por usuario, método de pago, tipo de vehículo, estado y rango de fechas.
  - Obtener el historial de pagos de un usuario.
  - Obtener detalles de un pago específico.
  - Crear un nuevo pago.
  - Actualizar el estado de un pago.
- **Diagramar caso de uso:**
  - Usuario realiza pago -> Pide datos -> Recibe confirmación.
  - Usuario consulta detalle de pago -> Pide ID de pago -> Recibe detalles.
  - Usuario consulta historial de pagos -> Pide ID de usuario -> Recibe lista de pagos.
  - Sistema actualiza estado de pago -> Pide ID de pago y nuevo estado -> Recibe confirmación.
  - Sistema lista pagos -> Recibe lista de pagos.
- **Diseñar endpoint inicial (borrador):**
  - GET /payments/{id}: Obtener detalle del pago.
  - POST /payments: Crear un nuevo pago.

- PUT /payments/{id}: Actualizar un pago.
  - DELETE /payments/{id}: Eliminar un pago.
  - GET /payments: Listar y filtrar pagos.
  - GET /payments/user/{user\_id}: Obtener el historial de pagos de un usuario.
  - PATCH /payments/{id}/status: Actualizar el estado de un pago.
- **Validar con stakeholders:** \* Revisar con el líder de producto para definir los estados de pago válidos, métodos de pago, tipos de vehículo, etc.

## FASE 2: Diseño Técnico y de Interfaces

- **Tareas puntuales:**
  - **Diseñar arquitectura interna en 3 capas:**
    - API Layer (FastAPI controllers)
    - Service Layer (PaymentService)
    - Repository Layer (SQL Database)

## FASE 3: Implementación del Servicio

- Se han creado los servicios asociados a operaciones CRUD mediante endpoints
  - Mostrar formulario de creación.
  - Mostrar formulario de actualización.
  - Mostrar formulario de eliminación.
  - Crear nuevo pago.
  - Actualizar pago existente.
  - Eliminar pago.
  - Listar pagos.

- Obtener detalles del pago.
  - Obtener historial de pagos por usuario.
  - Actualizar estado del pago
- Implementar lógica en PaymentController y PaymentService para manejar operaciones sobre pagos
    - Validar la existencia del pago y del usuario antes de realizar operaciones.
    - Manejar la lógica de negocio para crear, actualizar y eliminar pagos.
    - Implementar la lógica para obtener el historial de pagos de un usuario.
    - Implementar la lógica para actualizar el estado de un pago.
- Uso del patrón de inyección de dependencias
    - Integración con FastAPI Security para validar los scopes de acceso por rol (Security(get\_current\_user)).
- Se han escrito pruebas unitarias para las operaciones clave
    - Verifica la creación válida de un pago.
    - Comprueba la actualización exitosa de un pago existente.
    - Verifica el error esperado al intentar actualizar un pago inexistente.
    - Valida la eliminación correcta de un pago.
    - Comprueba la respuesta 404 cuando se intenta eliminar un pago no registrado.
    - Verifica la obtención del historial de pagos de un usuario.

#### **FASE 4: Seguridad, Configuración y Middleware**

- Agregar middleware de autenticación con JWT.
  - Se aplicó el siguiente requerimiento a cada ruta: Security(get\_current\_user, scopes=["system", "administrador", "supervisor", "cajero"])

- Separar configuración sensible en .env
  - Se separó la configuración mediante getenv().
- Roles y scopes
  - "system": "Acceso completo al sistema"
  - "administrador": "Permiso para gestionar usuarios y pagos"
  - "supervisor": "Permiso para supervisar pagos"
  - "cajero": "Permiso para crear y actualizar pagos"
- Validación de scopes:
  - La función get\_current\_user() se encarga de verificar y validar los scopes.

## Diagrama Estructural

```
© src.backend.app.api.routes.payment_cud_service.PaymentService
❶ controller
❶ router
❶ templates
❷ __init__(self)
❷ _setup_routes(self)
❸ listar_pago(self, request: Request, current_user: dict = Security(get_current_user, scopes=["system", "administrador", "supervisor"]))
❸ detalle_pago(self, id: int, request: Request, current_user: dict = Security(get_current_user, scopes=["system", "administrador", "supervisor"]))
❸ crear_pago_form(self, request: Request, current_user: dict = Security(get_current_user, scopes=["system", "administrador", "supervisor"]))
❸ crear_pago(self, id: int = Form(...), user: str = Form(...), payment_quantity: float = Form(...), payment_method: bool = Form(...), vehicle_type: int = Form(...))
❸ actualizar_pago_form(self, request: Request, current_user: dict = Security(get_current_user, scopes=["system", "administrador", "supervisor"]))
❸ actualizar_pago(self, id: int = Form(...), user: str = Form(...), payment_quantity: float = Form(...), payment_method: bool = Form(...), vehicle_type: int = Form(...))
❸ eliminar_pago_form(self, request: Request, current_user: dict = Security(get_current_user, scopes=["system", "administrador", "supervisor"]))
❸ eliminar_pago(self, id: int = Form(...), current_user: dict = Security(get_current_user, scopes=["system", "administrador", "supervisor"]))
```

### 3.7.2. Microservicio: TransportUnit

#### FASE 1: Análisis y Diseño Funcional del Microservicio

- **Microservicio:** transport\_service
- **Tareas puntuales:**
  - Identificar el bounded context:

- El dominio "Transporte" incluye tipo de unidad, estado, ubicación, capacidad, ruta asignada, conductor asignado y fecha de mantenimiento.
- Se separa transport\_service de otros servicios relacionados (e.g., RouteService, DriverService).
- **Definir la entidad principal:**
  - Entidad: Transport
  - Atributos: id, type, status, location, capacity, assigned\_route, assigned\_driver, last\_maintenance\_date, created\_at, updated\_at
- **Delimitar funcionalidad del microservicio:**
  - CRUD completo para unidades de transporte.
  - Filtrar unidades por tipo, estado, ubicación y ruta asignada.
  - Obtener detalles de una unidad de transporte específica.
  - Listar todas las unidades de transporte.
  - Crear una nueva unidad de transporte.
  - Actualizar una unidad de transporte existente.
  - Asignar/desasignar una unidad a una ruta.
  - Asignar/desasignar un conductor a una unidad.
- **Diagramar caso de uso:**
  - Usuario crea unidad de transporte -> Pide datos -> Recibe confirmación.
  - Usuario consulta detalle de unidad de transporte -> Pide ID de unidad -> Recibe detalles.
  - Usuario actualiza unidad de transporte -> Pide ID de unidad y datos -> Recibe confirmación.

- Usuario elimina unidad de transporte -> Pide ID de unidad -> Recibe confirmación.
  - Usuario lista unidades de transporte -> Recibe lista de unidades.
  - Usuario asigna unidad a ruta -> Pide ID de unidad e ID de ruta -> Recibe confirmación.
  - Usuario asigna conductor a unidad -> Pide ID de unidad e ID de conductor -> Recibe confirmación.
- **Diseñar endpoint inicial (borrador):**
    - GET /transports/{id}: Obtener detalle de la unidad de transporte.
    - POST /transports: Crear una nueva unidad de transporte.
    - PUT /transports/{id}: Actualizar una unidad de transporte.
    - DELETE /transports/{id}: Eliminar una unidad de transporte.
    - GET /transports: Listar y filtrar unidades de transporte.
    - POST /transports/{id}/route: Asignar unidad a una ruta.
    - DELETE /transports/{id}/route/{route\_id}: Desasignar unidad de una ruta.
    - POST /transports/{id}/driver: Asignar conductor a una unidad.
    - DELETE /transports/{id}/driver/{driver\_id}: Desasignar conductor de una unidad.
  - **Validar con stakeholders:** \* Revisar con el líder de producto para definir los tipos de unidad válidos, estados, reglas de asignación, etc.

## FASE 2: Diseño Técnico y de Interfaces

- **Tareas puntuales:**
  - **Diseñar arquitectura interna en 3 capas:**

- API Layer (FastAPI controllers)
- Service Layer (TransportService)
- Repository Layer (SQL Database)

### FASE 3: Implementación del Servicio

- Se han creado los servicios asociados a operaciones CRUD mediante endpoints
  - Mostrar formulario de creación.
  - Mostrar formulario de actualización.
  - Mostrar formulario de eliminación.
  - Crear una nueva unidad de transporte.
  - Actualizar una unidad de transporte existente.
  - Eliminar una unidad de transporte.
  - Listar las unidades de transporte.
  - Obtener detalles de una unidad de transporte específica.
- Implementar lógica en TransportController y TransportService para manejar operaciones sobre las unidades de transporte
  - Validar la existencia de la unidad de transporte antes de realizar operaciones.
  - Manejar la lógica de negocio para crear, actualizar y eliminar unidades de transporte.
  - Implementar la lógica para obtener detalles de una unidad de transporte.
  - Implementar la lógica para listar las unidades de transporte.
  - Implementar la lógica para asignar y desasignar unidades a rutas y conductores.
- Uso del patrón de inyección de dependencias
  - Integración con FastAPI Security para validar los scopes de acceso por rol (Security(get\_current\_user)).

- Se han escrito pruebas unitarias para las operaciones clave
  - Verifica la creación válida de una unidad de transporte.
  - Comprueba la actualización exitosa de una unidad de transporte existente.
  - Verifica el error esperado al intentar actualizar una unidad de transporte inexistente.
  - Valida la eliminación correcta de una unidad de transporte.
  - Comprueba la respuesta 404 cuando se intenta eliminar una unidad de transporte no registrada.

#### FASE 4: Seguridad, Configuración y Middleware

- Agregar middleware de autenticación con JWT.
  - Se aplicó el siguiente requerimiento a cada ruta: Security(get\_current\_user, scopes=["system", "administrador", "supervisor", "tecnico"])
- Separar configuración sensible en .env
  - Se separó la configuración mediante getenv().
- Roles y scopes
  - "system": "Acceso completo al sistema"
  - "administrador": "Permiso para gestionar usuarios y unidades de transporte"
  - "supervisor": "Permiso para supervisar las unidades de transporte"
  - "tecnico": "Permiso para crear y modificar unidades de transporte"
- Validación de scopes:
  - La función get\_current\_user() se encarga de verificar y validar los scopes.

#### Diagrama Estructural

```
© src.backend.app.api.routes.transport_unit_cud_service.TransportQueryService

❶ controller
❷ router
❸ templates
❹ __init__(self)
❺ _setup_routes(self)
❻ listar_unidades(self, request: Request, current_user: dict = Security(get_current_user, scopes=["system", "administrador", "supervisor"]))
❼ detalle_unidad(self, id: int, request: Request, current_user: dict = Security(get_current_user, scopes=["system", "administrador", "supervisor"]))
```

```
© src.backend.app.api.routes.transport_unit_cud_service.TransportCUDService

❶ controller
❷ router
❸ templates
❹ __init__(self)
❺ _setup_routes(self)
❻ crear_unidad_form(self, request: Request, current_user: dict = Security(get_current_user, scopes=["system", "administrador", "supervisor"]))
❼ crear_unidad(self, id: int = Form(...), type: str = Form(...), status: str = Form(...), ubicacion: str = Form(...), capacity: int = Form(...), current_user: dict = Security(get_current_user, scopes=["system", "administrador", "supervisor"]))
❽ actualizar_unidad_form(self, request: Request, current_user: dict = Security(get_current_user, scopes=["system", "administrador", "supervisor"]))
❾ actualizar_unidad(self, id: int = Form(...), type: str = Form(...), status: str = Form(...), ubicacion: str = Form(...), capacity: int = Form(...), current_user: dict = Security(get_current_user, scopes=["system", "administrador", "supervisor"]))
❿ eliminar_unidad_form(self, request: Request, current_user: dict = Security(get_current_user, scopes=["system", "administrador", "supervisor"]))
❽ eliminar_unidad(self, id: int = Form(...), current_user: dict = Security(get_current_user, scopes=["system", "administrador", "supervisor"]))
```

### 3.7.3. Microservicio: EstadoMantenimiento

#### FASE 1: Análisis y Diseño Funcional del Microservicio

- **Microservicio:** maintenance\_status\_service
- **Tareas puntuales:**
  - **Identificar el bounded context:**
    - El dominio "Estado de Mantenimiento" incluye la unidad, el tipo de mantenimiento, el estado y la fecha de inicio/fin.
    - Se separa maintenance\_status\_service de otros servicios relacionados.
  - **Definir la entidad principal:**
    - Entidad: MaintenanceStatus
    - Atributos: id, unit\_id, type, status, start\_date, end\_date, description
  - **Delimitar funcionalidad del microservicio:**

- CRUD completo para el estado de mantenimiento.
  - Filtrar estados de mantenimiento por unidad, tipo, estado y rango de fechas.
  - Historial de estados de mantenimiento para una unidad.
- **Diagramar caso de uso:**
    - Usuario crea estado de mantenimiento -> Pide datos -> Recibe confirmación.
    - Usuario ve detalle del estado -> Pide ID -> Recibe detalles.
    - Usuario actualiza estado -> Pide ID y datos -> Recibe confirmación.
    - Usuario elimina estado -> Pide ID -> Recibe confirmación.
    - Usuario filtra estados -> Pide criterios de filtrado -> Recibe lista de estados.
    - Usuario ve historial de estados de una unidad -> Pide ID de unidad -> Recibe lista de estados.
  - **Diseñar endpoint inicial (borrador):**
    - GET /maintenance\_status/{id}: Obtener detalle del estado de mantenimiento.
    - POST /maintenance\_status: Crear un nuevo estado de mantenimiento.
    - PUT /maintenance\_status/{id}: Actualizar un estado de mantenimiento.
    - DELETE /maintenance\_status/{id}: Eliminar un estado de mantenimiento.
    - GET /maintenance\_status: Listar y filtrar estados de mantenimiento.
    - GET /maintenance\_status/unit/{unit\_id}/history: Obtener el historial de estados de mantenimiento para una unidad.
- **Validar con stakeholders:**
    - Revisar con el líder de producto para definir los estados válidos, tipos de mantenimiento, etc.

## FASE 2: Diseño Técnico y de Interfaces

- **Tareas puntuales:**
  - **Diseñar arquitectura interna en 3 capas:**
    - API Layer (FastAPI controllers)
    - Service Layer (MaintenanceStatusService)
    - Repository Layer (SQL Database)

### **FASE 3: Implementación del Servicio**

- Se han creado los servicios asociados a operaciones CRUD mediante endpoints
  - Mostrar formulario de creación.
  - Mostrar formulario de actualización.
  - Mostrar formulario de eliminación.
  - Crear nuevo estado de mantenimiento.
  - Actualizar estado de mantenimiento existente.
  - Eliminar estado de mantenimiento.
  - Listar estados de mantenimiento.
  - Obtener detalles del estado de mantenimiento.
  - Filtrar estados de mantenimiento.
  - Obtener historial de estados de mantenimiento por unidad.
- Implementar lógica en MaintenanceStatusController y MaintenanceStatusService para manejar operaciones sobre estados de mantenimiento
  - Validar la existencia del estado de mantenimiento y de la unidad antes de realizar operaciones.
  - Manejar la lógica de negocio para crear, actualizar y eliminar estados.
  - Implementar la lógica de filtrado de estados de mantenimiento.

- Implementar la lógica para obtener el historial de estados de mantenimiento de una unidad
- Uso del patrón de inyección de dependencias
  - Integración con FastAPI Security para validar los scopes de acceso por rol (Security(get\_current\_user)).
- Se han escrito pruebas unitarias para las operaciones clave
  - Verifica la creación válida de un estado de mantenimiento.
  - Comprueba la actualización exitosa de un estado de mantenimiento existente.
  - Verifica el error esperado al intentar actualizar un estado de mantenimiento inexistente.
  - Valida la eliminación correcta de un estado de mantenimiento.
  - Comprueba la respuesta 404 cuando se intenta eliminar un estado de mantenimiento no registrado.
  - Verifica el filtrado de estados de mantenimiento.
  - Verifica la obtención del historial de estados de mantenimiento de una unidad

#### **FASE 4: Seguridad, Configuración y Middleware**

- Agregar middleware de autenticación con JWT.
  - Se aplicó el siguiente requerimiento a cada ruta: Security(get\_current\_user, scopes=["system", "mantenimiento", "supervisor"])
- Separar configuración sensible en .env
  - Se separó la configuración mediante getenv().
- Roles y scopes
  - "system": "Acceso completo al sistema"

- "mantenimiento": "Permiso para gestionar estados de mantenimiento"
- "supervisor": "Permiso para supervisar estados de mantenimiento"
- Validación de scopes:
  - La función `get_current_user()` se encarga de verificar y validar los scope

## Diagrama Estructural



### 3.7.4. Microservicio: Incidencia

#### FASE 1: Análisis y Diseño Funcional del Microservicio

- **Microservicio:** incidence\_service
- **Tareas puntuales:**
  - **Identificar el bounded context:**
    - El dominio "Incidencia" incluye tipo de incidencia, prioridad, y estado.

- Se separa incidence \_service de otros servicios relacionados (e.g., TicketService, NotificationService).

- **Definir la entidad principal:**

- Entidad: Incidencia
- Atributos: id, ticket\_id, description, type, unit\_id, status, priority, created\_at, resolved\_at

- **Delimitar funcionalidad del microservicio:**

- CRUD completo para incidencias.
- Filtrar incidencias por ticket, unidad, tipo, estado, prioridad, y fecha.
- Cambiar el estado de una incidencia.
- Asignar una incidencia a una unidad.

- **Diagramar caso de uso:**

- Usuario crea incidencia -> Pide datos -> Recibe confirmación.
- Usuario ve detalle de incidencia -> Pide ID -> Recibe detalles.
- Usuario actualiza incidencia -> Pide ID y datos -> Recibe confirmación.
- Usuario elimina incidencia -> Pide ID -> Recibe confirmación.
- Usuario filtra incidencias -> Pide criterios de filtrado -> Recibe lista de incidencias.
- Usuario cambia estado de incidencia -> Pide ID y nuevo estado -> Recibe confirmación
- Usuario asigna incidencia a unidad -> Pide ID y Unidad -> Recibe confirmacion

- **Diseñar endpoint inicial (borrador):**

- GET /incidences/{id}: Obtener detalle de la incidencia.
  - POST /incidences: Crear una nueva incidencia.
  - PUT /incidences/{id}: Actualizar una incidencia.
  - DELETE /incidences/{id}: Eliminar una incidencia.
  - GET /incidences: Listar y filtrar incidencias.
  - POST /incidences/{id}/status/{status\_code}: Cambiar el estado de una incidencia
  - POST /incidences/{id}/unit/{unit\_id}: Asignar una incidencia a una unidad
- **Validar con stakeholders:**
    - Revisar con el líder de producto para definir los estados válidos, prioridades, tipos de incidencia, etc.

## FASE 2: Diseño Técnico y de Interfaces

- **Tareas puntuales:**
  - **Diseñar arquitectura interna en 3 capas:**
    - API Layer (FastAPI controllers)
    - Service Layer (IncidenceService)
    - Repository Layer (SQL Database)

## FASE 3: Implementación del Servicio

- Se han creado los servicios asociados a operaciones CRUD mediante endpoints
  - Mostrar formulario de creación.
  - Mostrar formulario de actualización.
  - Mostrar formulario de eliminación.

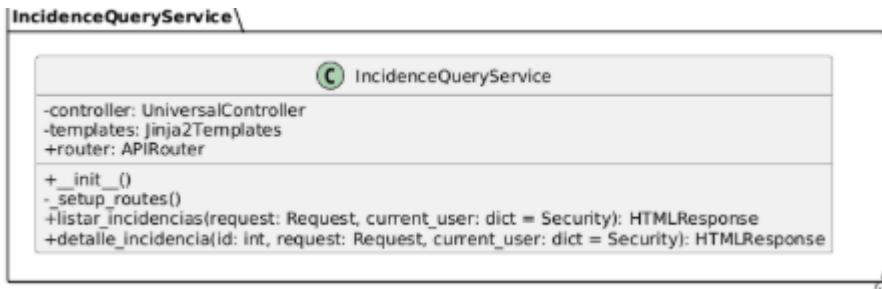
- Crear nueva incidencia.
- Actualizar incidencia existente.
- Eliminar incidencia.
- Listar incidencias
- Obtener detalles de la incidencia
- Filtrar incidencias.
- Cambiar estado de incidencia
- Asignar incidencia a unidad
- Implementar lógica en IncidenceController y IncidenceService para manejar operaciones sobre incidencias
  - Validar la existencia de la incidencia y la unidad antes de realizar operaciones.
  - Manejar la lógica de negocio para asignar incidencias, cambiar estados.
  - Implementar la lógica de filtrado de incidencias.
- Uso del patrón de inyección de dependencias
  - Integración con FastAPI Security para validar los scopes de acceso por rol (Security(get\_current\_user)).
- Se han escrito pruebas unitarias para las operaciones clave
  - Verifica la creación válida de una incidencia.
  - Comprueba la actualización exitosa de una incidencia existente.
  - Verifica el error esperado al intentar actualizar una incidencia inexistente.
  - Valida la eliminación correcta de una incidencia.
  - Comprueba la respuesta 404 cuando se intenta eliminar una incidencia no registrada.
  - Verifica el filtrado de incidencias

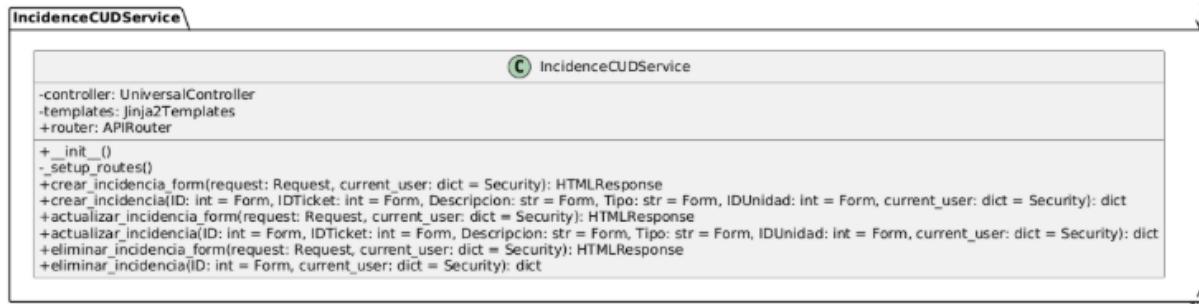
- Verifica el cambio de estado
- Verifica la asignacion a una unidad

## FASE 4: Seguridad, Configuración y Middleware

- Agregar middleware de autenticación con JWT.
  - Se aplicó el siguiente requerimiento a cada ruta: Security(get\_current\_user, scopes=["system", "administrador", "supervisor", "operador"])
- Separar configuración sensible en .env
  - Se separó la configuración mediante getenv().
- Roles y scopes
  - "system": "Acceso completo al sistema"
  - "administrador": "Permiso para gestionar usuarios y recursos"
  - "supervisor": "Permiso para supervisar incidencias"
  - "operador": "Permiso para crear y modificar incidencias"
- Validación de scopes:
  - La función get\_current\_user() se encarga de verificar y validar los scopes.

## Diagrama Estructural





### 3.7.5. Microservicio: Ticket

#### FASE 1: Análisis y Diseño Funcional del Microservicio

- **Microservicio:** ticket\_service
- **Tareas puntuales:**
  - **Identificar el bounded context:**
    - El dominio "Ticket" incluye el tipo de ticket y la prioridad.
    - Se separa ticket\_service de otros servicios relacionados (e.g., NotificationService).
  - **Definir la entidad principal:**
    - Entidad: Ticket
    - Atributos: ticket\_id, title, description, status\_code, priority, created\_at, assigned\_to (usuario asignado), type
  - **Delimitar funcionalidad del microservicio:**
    - CRUD completo para tickets.
    - Asignar un ticket a un usuario.
    - Cambiar el estado de un ticket.
    - Agregar comentarios a un ticket.

- Filtrar tickets por estado, usuario, prioridad, etc.
- **Diagramar caso de uso:**
  - Usuario crea ticket -> Pide datos -> Recibe confirmación.
  - Usuario ve detalle del ticket -> Pide ID -> Recibe detalles.
  - Usuario actualiza ticket -> Pide ID y datos -> Recibe confirmación.
  - Usuario elimina ticket -> Pide ID -> Recibe confirmación.
  - Usuario asigna ticket -> Pide ID de ticket y usuario -> Recibe confirmación.
  - Usuario cambia estado -> Pide ID de ticket y estado -> Recibe confirmación.
  - Usuario agrega comentario -> Pide ID de ticket y comentario -> Recibe confirmación.
  - Usuario filtra tickets -> Pide criterios -> Recibe lista de tickets.
- **Diseñar endpoint inicial (borrador):**
  - GET /tickets/{ticket\_id}: Obtener detalle del ticket.
  - POST /tickets: Crear un nuevo ticket.
  - PUT /tickets/{ticket\_id}: Actualizar un ticket.
  - DELETE /tickets/{ticket\_id}: Eliminar un ticket.
  - POST /tickets/{ticket\_id}/assign/{user\_id}: Asignar ticket.
  - POST /tickets/{ticket\_id}/status/{status\_code}: Cambiar estado.
  - POST /tickets/{ticket\_id}/comments: Agregar comentario.
  - GET /tickets: Listar y filtrar tickets.
- **Validar con stakeholders:**
  - Revisar con el líder de producto para definir los estados válidos, prioridades, tipos de ticket, etc.

## FASE 2: Diseño Técnico y de Interfaces

- **Tareas puntuales:**
  - **Diseñar arquitectura interna en 3 capas:**
    - API Layer (FastAPI controllers)
    - Service Layer (TicketService)
    - Repository Layer (SQL Database)

## FASE 3: Implementación del Servicio

- Se han creado los servicios asociados a operaciones CRUD mediante endpoints
  - Mostrar formulario de creación.
  - Mostrar formulario de actualización.
  - Mostrar formulario de eliminación.
  - Crear nuevo ticket.
  - Actualizar ticket existente.
  - Eliminar ticket.
  - Listar tickets
  - Obtener detalles del ticket
  - Asignar ticket
  - Cambiar estado del ticket
  - Agregar comentarios
- Implementar lógica en TicketController y TicketService para manejar operaciones sobre tickets
  - Validar la existencia del ticket y del usuario antes de realizar operaciones.

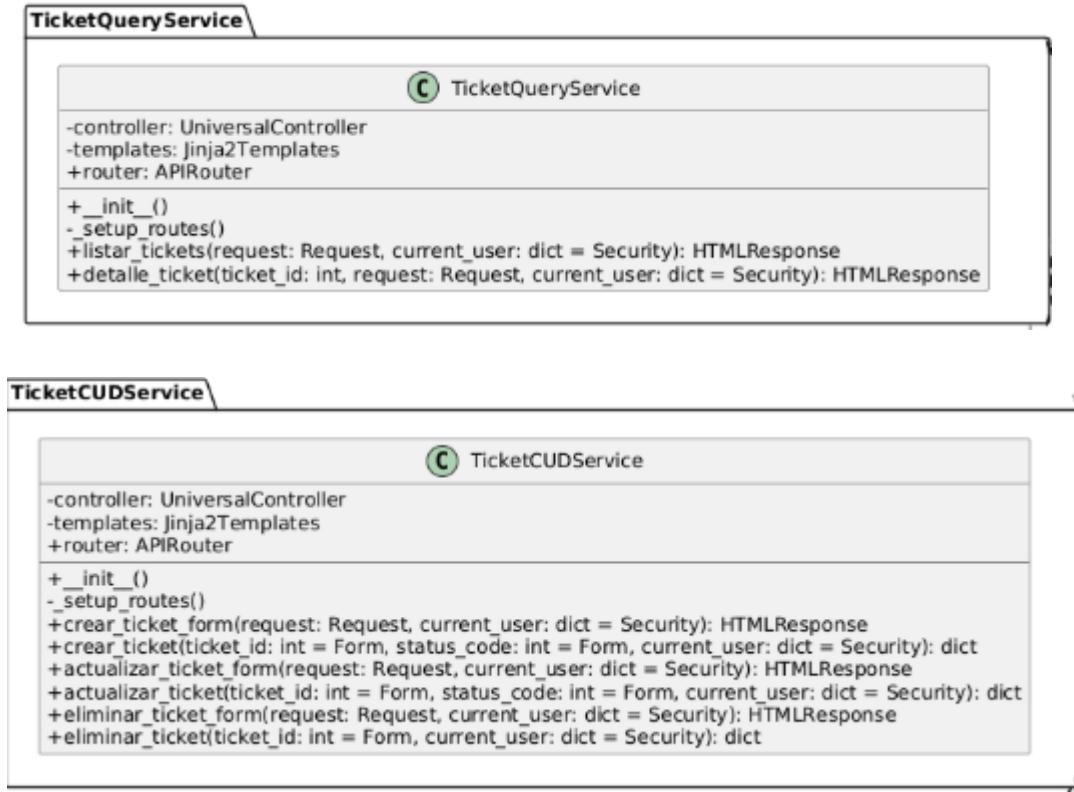
- Manejar la lógica de negocio para asignar tickets, cambiar estados y agregar comentarios.
- Implementar la lógica de filtrado de tickets.
- Uso del patrón de inyección de dependencias
  - Integración con FastAPI Security para validar los scopes de acceso por rol (Security(get\_current\_user)).
- Se han escrito pruebas unitarias para las operaciones clave
  - Verifica la creación válida de un ticket.
  - Comprueba la actualización exitosa de un ticket existente.
  - Verifica el error esperado al intentar actualizar un ticket inexistente.
  - Valida la eliminación correcta de un ticket.
  - Comprueba la respuesta 404 cuando se intenta eliminar un ticket no registrado.
  - Verifica la asignación de un ticket a un usuario.
  - Verifica el cambio de estado de un ticket.
  - Verifica la adición de comentarios a un ticket.
  - Verifica el filtrado correcto de tickets.

#### FASE 4: Seguridad, Configuración y Middleware

- Agregar middleware de autenticación con JWT.
  - Se aplicó el siguiente requerimiento a cada ruta: Security(get\_current\_user, scopes=["system", "administrador", "agente", "cliente", "supervisor\_tickets"])
- Separar configuración sensible en .env
  - Se separó la configuración mediante getenv().
- Roles y scopes

- "system": "Acceso completo al sistema"
  - "administrador": "Permiso para gestionar usuarios y tickets"
  - "agente": "Permiso para gestionar tickets asignados"
  - "cliente": "Permiso para crear y ver sus propios tickets"
  - "supervisor\_tickets": "Permiso para ver y gestionar todos los tickets"
- Validación de scopes:
    - La función `get_current_user()` se encarga de verificar y validar los scopes.

## Diagrama Estructural



### 3.7.6. Microservicio: Ruta

#### FASE 1: Análisis y Diseño Funcional del Microservicio

- **Microservicio:** route\_service

- **Tareas puntuales:**

- **Identificar el bounded context:**

- El dominio "Ruta" incluye información sobre el horario, nombre, descripción, origen, destino y la secuencia de paradas.
    - Se separa route\_service de otros servicios relacionados (e.g., SchedulingService, MapService).

- **Definir la entidad principal:**

- Entidad: Route
    - Atributos: id, schedule\_id, name, description, origin, destination, stops, created\_at, updated\_at

- **Delimitar funcionalidad del microservicio:**

- CRUD completo para rutas.
    - Filtrar rutas por origen, destino, horario y nombre.
    - Obtener detalles de una ruta específica.
    - Listar todas las rutas.
    - Crear una nueva ruta.
    - Actualizar una ruta existente.

- **Diagramar caso de uso:**

- Usuario crea ruta -> Pide datos -> Recibe confirmación.
    - Usuario consulta detalle de ruta -> Pide ID de ruta -> Recibe detalles.
    - Usuario actualiza ruta -> Pide ID de ruta y datos -> Recibe confirmación.
    - Usuario elimina ruta -> Pide ID de ruta -> Recibe confirmación.

- Usuario lista rutas -> Recibe lista de rutas.
- **Diseñar endpoint inicial (borrador):**
  - GET /routes/{id}: Obtener detalle de la ruta.
  - POST /routes: Crear una nueva ruta.
  - PUT /routes/{id}: Actualizar una ruta.
  - DELETE /routes/{id}: Eliminar una ruta.
  - GET /routes: Listar y filtrar rutas.
- **Validar con stakeholders:** \* Revisar con el líder de producto para definir los campos obligatorios, valores válidos, etc.

## FASE 2: Diseño Técnico y de Interfaces

- **Tareas puntuales:**
  - **Diseñar arquitectura interna en 3 capas:**
    - API Layer (FastAPI controllers)
    - Service Layer (RouteService)
    - Repository Layer (SQL Database)

## FASE 3: Implementación del Servicio

- Se han creado los servicios asociados a operaciones CRUD mediante endpoints
  - Mostrar formulario de creación.
  - Mostrar formulario de actualización.
  - Mostrar formulario de eliminación.
  - Crear una nueva ruta.
  - Actualizar una ruta existente.

- Eliminar una ruta.
- Listar las rutas.
- Obtener detalles de una ruta específica.
- Implementar lógica en RouteController y RouteService para manejar operaciones sobre rutas
  - Validar la existencia de la ruta y del horario antes de realizar operaciones.
  - Manejar la lógica de negocio para crear, actualizar y eliminar rutas.
  - Implementar la lógica para obtener detalles de una ruta.
  - Implementar la lógica para listar las rutas.
- Uso del patrón de inyección de dependencias
  - Integración con FastAPI Security para validar los scopes de acceso por rol (Security(get\_current\_user)).
- Se han escrito pruebas unitarias para las operaciones clave
  - Verifica la creación válida de una ruta.
  - Comprueba la actualización exitosa de una ruta existente.
  - Verifica el error esperado al intentar actualizar una ruta inexistente.
  - Valida la eliminación correcta de una ruta.
  - Comprueba la respuesta 404 cuando se intenta eliminar una ruta no registrada.

#### FASE 4: Seguridad, Configuración y Middleware

- Agregar middleware de autenticación con JWT.
  - Se aplicó el siguiente requerimiento a cada ruta: Security(get\_current\_user, scopes=["system", "administrador", "supervisor", "planificador"])
- Separar configuración sensible en .env

- Se separó la configuración mediante getenv().
- Roles y scopes
  - "system": "Acceso completo al sistema"
  - "administrador": "Permiso para gestionar usuarios y rutas"
  - "supervisor": "Permiso para supervisar rutas"
  - "planificador": "Permiso para crear y modificar rutas"
- Validación de scopes:
  - La función get\_current\_user() se encarga de verificar y validar los scopes.

## Diagrama Estructural



### 3.7.7. Microservicio: Horario

#### FASE 1: Análisis y Diseño Funcional del Microservicio

- **Microservicio:** schedule\_service

- **Tareas puntuales:**

- **Identificar el bounded context:**

- El dominio "Horario" incluye horas de llegada y salida, frecuencia, tipo de día, y rutas asociadas.
    - Se separa schedule\_service de otros servicios relacionados (e.g., RouteService, NotificationService).

- **Definir la entidad principal:**

- Entidad: Schedule
    - Atributos: id, arrival\_time, departure\_time, frequency, day\_type, route\_id, created\_at, updated\_at

- **Delimitar funcionalidad del microservicio:**

- CRUD completo para horarios.
    - Filtrar horarios por ruta, tipo de día, y rango de horas.
    - Obtener detalles de un horario específico.
    - Listar todos los horarios.
    - Crear un nuevo horario.
    - Actualizar un horario existente.

- **Diagramar caso de uso:**

- Usuario crea horario -> Pide datos -> Recibe confirmación.
    - Usuario consulta detalle de horario -> Pide ID de horario -> Recibe detalles.
    - Usuario actualiza horario -> Pide ID de horario y datos -> Recibe confirmación.

- Usuario elimina horario -> Pide ID de horario -> Recibe confirmación.
  - Usuario lista horarios -> Recibe lista de horarios.
- **Diseñar endpoint inicial (borrador):**
    - GET /schedules/{id}: Obtener detalle del horario.
    - POST /schedules: Crear un nuevo horario.
    - PUT /schedules/{id}: Actualizar un horario.
    - DELETE /schedules/{id}: Eliminar un horario.
    - GET /schedules: Listar y filtrar horarios.
- **Validar con stakeholders:** \* Revisar con el líder de producto para definir los campos obligatorios, valores válidos, etc.

## FASE 2: Diseño Técnico y de Interfaces

- **Tareas puntuales:**
  - **Diseñar arquitectura interna en 3 capas:**
    - API Layer (FastAPI controllers)
    - Service Layer (ScheduleService)
    - Repository Layer (SQL Database)

## FASE 3: Implementación del Servicio

- Se han creado los servicios asociados a operaciones CRUD mediante endpoints
  - Mostrar formulario de creación.
  - Mostrar formulario de actualización.
  - Mostrar formulario de eliminación.
  - Crear un nuevo horario.

- Actualizar un horario existente.
- Eliminar un horario.
- Listar los horarios.
- Obtener detalles de un horario específico.
- Implementar lógica en ScheduleController y ScheduleService para manejar operaciones sobre horarios
  - Validar la existencia del horario antes de realizar operaciones.
  - Manejar la lógica de negocio para crear, actualizar y eliminar horarios.
  - Implementar la lógica para obtener detalles de un horario.
  - Implementar la lógica para listar los horarios.
- Uso del patrón de inyección de dependencias
  - Integración con FastAPI Security para validar los scopes de acceso por rol (Security(get\_current\_user)).
- Se han escrito pruebas unitarias para las operaciones clave
  - Verifica la creación válida de un horario.
  - Comprueba la actualización exitosa de un horario existente.
  - Verifica el error esperado al intentar actualizar un horario inexistente.
  - Valida la eliminación correcta de un horario.
  - Comprueba la respuesta 404 cuando se intenta eliminar un horario no registrado.

#### **FASE 4: Seguridad, Configuración y Middleware**

- Agregar middleware de autenticación con JWT.
  - Se aplicó el siguiente requerimiento a cada ruta: Security(get\_current\_user, scopes=["system", "administrador", "supervisor", "planificador"])

- Separar configuración sensible en .env
  - Se separó la configuración mediante getenv().
- Roles y scopes
  - "system": "Acceso completo al sistema"
  - "administrador": "Permiso para gestionar usuarios y horarios"
  - "supervisor": "Permiso para supervisar los horarios"
  - "planificador": "Permiso para crear y modificar horarios"
- Validación de scopes:
  - La función get\_current\_user() se encarga de verificar y validar los scopes.

## Diagrama Estructural



### 3.7.8. Microservicio: Turno

## FASE 1: Análisis y Diseño Funcional del Microservicio

- **Microservicio:** shift\_service
- **Tareas puntuales:**
  - **Identificar el bounded context:**
    - El dominio "Turno" incluye tipo de turno, hora de inicio, hora de fin, empleados asignados y fecha.
    - Se separa shift\_service de otros servicios relacionados (e.g., EmployeeService, SchedulingService).
  - **Definir la entidad principal:**
    - Entidad: Shift
    - Atributos: id, shift\_type, start\_time, end\_time, assigned\_employees, date, created\_at, updated\_at
  - **Delimitar funcionalidad del microservicio:**
    - CRUD completo para turnos.
    - Filtrar turnos por tipo, fecha y empleado.
    - Obtener detalles de un turno específico.
    - Listar todos los turnos.
    - Crear un nuevo turno.
    - Actualizar un turno existente.
    - Asignar/desasignar empleados a un turno.
  - **Diagramar caso de uso:**
    - Usuario crea turno -> Pide datos -> Recibe confirmación.
    - Usuario consulta detalle de turno -> Pide ID de turno -> Recibe detalles.

- Usuario actualiza turno -> Pide ID de turno y datos -> Recibe confirmación.
- Usuario elimina turno -> Pide ID de turno -> Recibe confirmación.
- Usuario lista turnos -> Recibe lista de turnos.
- Usuario asigna/desasigna empleado -> Pide ID de turno e ID de empleado -> Recibe confirmación.
- **Diseñar endpoint inicial (borrador):**
  - GET /shifts/{id}: Obtener detalle del turno.
  - POST /shifts: Crear un nuevo turno.
  - PUT /shifts/{id}: Actualizar un turno.
  - DELETE /shifts/{id}: Eliminar un turno.
  - GET /shifts: Listar y filtrar turnos.
  - POST /shifts/{id}/employees: Asignar empleados a un turno.
  - DELETE /shifts/{id}/employees/{employee\_id}: Desasignar un empleado de un turno.
- **Validar con stakeholders:** \* Revisar con el líder de producto para definir los tipos de turno válidos, reglas de asignación de empleados, etc.

## FASE 2: Diseño Técnico y de Interfaces

- **Tareas puntuales:**
  - **Diseñar arquitectura interna en 3 capas:**
    - API Layer (FastAPI controllers)
    - Service Layer (ShiftService)
    - Repository Layer (SQL Database)

## FASE 3: Implementación del Servicio

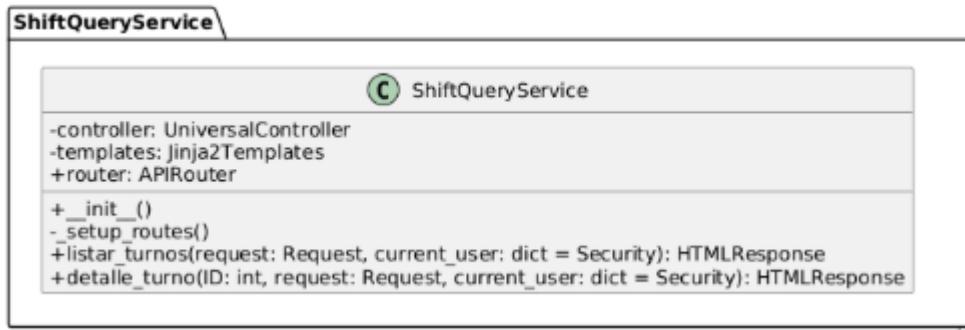
- Se han creado los servicios asociados a operaciones CRUD mediante endpoints
  - Mostrar formulario de creación.
  - Mostrar formulario de actualización.
  - Mostrar formulario de eliminación.
  - Crear un nuevo turno.
  - Actualizar un turno existente.
  - Eliminar un turno.
  - Listar los turnos.
  - Obtener detalles de un turno específico.
- Implementar lógica en ShiftController y ShiftService para manejar operaciones sobre turnos
  - Validar la existencia del turno y de los empleados antes de realizar operaciones.
  - Manejar la lógica de negocio para crear, actualizar y eliminar turnos.
  - Implementar la lógica para obtener detalles de un turno.
  - Implementar la lógica para listar los turnos.
  - Implementar la lógica para asignar y desasignar empleados a un turno.
- Uso del patrón de inyección de dependencias
  - Integración con FastAPI Security para validar los scopes de acceso por rol (Security(get\_current\_user)).
- Se han escrito pruebas unitarias para las operaciones clave
  - Verifica la creación válida de un turno.
  - Comprueba la actualización exitosa de un turno existente.
  - Verifica el error esperado al intentar actualizar un turno inexistente.

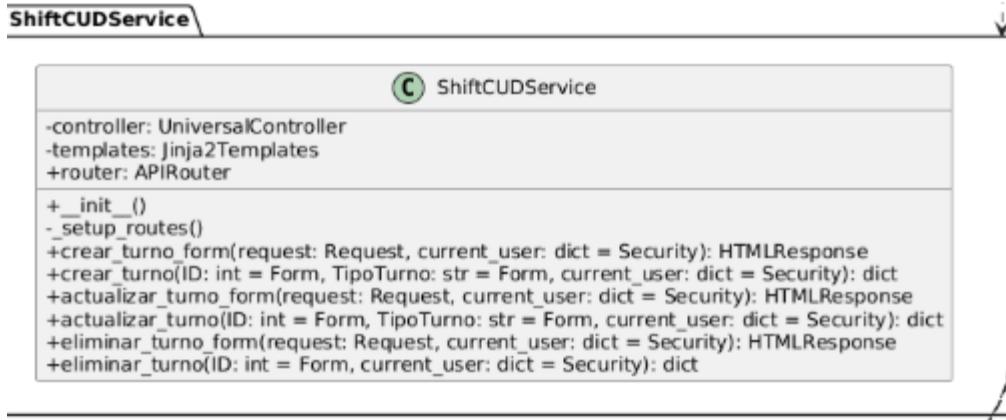
- Valida la eliminación correcta de un turno.
- Comprueba la respuesta 404 cuando se intenta eliminar un turno no registrado.

## FASE 4: Seguridad, Configuración y Middleware

- Agregar middleware de autenticación con JWT.
  - Se aplicó el siguiente requerimiento a cada ruta: Security(get\_current\_user, scopes=["system", "administrador", "supervisor", "planificador"])
- Separar configuración sensible en .env
  - Se separó la configuración mediante getenv().
- Roles y scopes
  - "system": "Acceso completo al sistema"
  - "administrador": "Permiso para gestionar usuarios y turnos"
  - "supervisor": "Permiso para supervisar los turnos"
  - "planificador": "Permiso para crear y modificar turnos"
- Validación de scopes:
  - La función get\_current\_user() se encarga de verificar y validar los scopes.

## Diagrama Estructural





### 3.7.9. Microservicio: Tarjeta

#### FASE 1: Análisis y Diseño Funcional del Microservicio

Microservicio: card\_service

##### Identificar el bounded context:

El dominio "Tarjeta" incluye Tipo\_tarjeta.

Separar card\_service de Payment\_Services

##### Definir la entidad principal:

Entidad: Tarjeta

Atributos: ID, Tipo, Balance

##### Delimitar funcionalidad del microservicio:

Consultar saldo

CRUD

##### Diagramar caso de uso:

Usuario consulta saldo -> Pide datos -> Recibe Balance

Usuario consulta movimiento -> Pide datos -> Recibe lista de movimientos

Usuario usa tarjeta -> Actualiza datos -> Recibe monto actual

##### Diseñar endpoint inicial (borrador):

GET tarjeta/{id}/monto

GET tarjeta/{id}/movimiento

POST tarjeta/{id}/uso

**Validar con stakeholders: (Pendiente con Edwin)**

Revisión con líder de producto para ajustar estados válidos:

Created

Paid

Cancelled

delivered

**FASE 2: Diseño Técnico y de Interfaces**

**Tareas puntuales:**

**Diseñar arquitectura interna en 3 capas:**

API Layer (FastAPI controllers)

Service Layer (CardService)

Repository Layer (.json -> .sql)

**FASE 3: Implementación del Servicio**

**Se han creado los servicios asociados a operaciones CRUD mediante endpoints**

- Mostrar formulario de creación.
- Mostrar formulario de actualización.
- Mostrar formulario de eliminación.
- Crear nueva tarjeta.
- Actualizar tarjeta existente.
- Eliminar tarjeta.

**Implementar lógica en UniversalController para manejar operaciones sobre tarjetas**

- Validar existencia de tarjeta antes de actualizar o eliminar.
- Crear y persistir nueva tarjeta (CardCreate).

- Mantener consistencia del balance al actualizar tipo.

### **Uso del patrón de inyección de dependencias**

- Integración con FastAPI Security para validar los scopes de acceso por rol (Security(get\_current\_user)).

### **Se han escrito pruebas unitarias para las operaciones clave**

- Verifica la creación válida de una tarjeta.
- Comprueba la actualización exitosa de una tarjeta existente.
- Verifica el error esperado al intentar actualizar una tarjeta inexistente.
- Valida la eliminación correcta de una tarjeta
- Comprueba la respuesta 404 cuando se intenta eliminar una tarjeta no registrada.

## **FASE 4: Seguridad, Configuración y Middleware**

### **Agregar middleware de autenticación con JWT.**

Se aplicó el siguiente requerimiento a cada ruta

```
Security(get_current_user,scopes=["system", "administrador", "pasajero", "supervisor",  
"mantenimiento"] ))
```

### **Separar configuración sensible en .env**

Se separó la configuración mediante getenv()

### **Roles y scopes**

```
"system": "Full system access",  
"administrador": "Permission to manage users",  
"pasajero": "Passenger permission",  
"supervisor": "Supervisor permission",
```

"mantenimiento": "Maintenance permission",  
"operador": "Operator permission"

### **Validación de scopes:**

La función `get_current_user()` se encarga de verificar y validar los scopes

## **FASE 5: Validación, Observabilidad y Pruebas**

### **Ejecutar pruebas de contrato con PyTest:**

Verificar la respuesta esperada al consumir GET /card/crear, POST /card/create, POST /card/update, y POST /card/delete.

Validar que el formato de datos (CardOut) sea consistente con lo acordado por el consumidor del API.

### **Añadir logging estructurado:**

Se han añadido loggings de eventos relevantes para esclarecer errores y información.

### **Ejecutar pruebas de contrato con PyTest:**

Verificar la respuesta esperada al consumir GET /card/crear, POST /card/create, POST /card/update, y POST /card/delete.

Validar que el formato de datos (CardOut) sea consistente con lo acordado por el consumidor del API.

### **Añadir logging estructurado:**

Se han añadido loggings de eventos relevantes para esclarecer errores y información.

## Diagrama Estructural

```

©  src.backend.app.api.routes.card_service.CardQueryService
(f) router
(f) controller
(f) templates
@m __init__(self)
@m consultar(self, request: Request, current_user: dict = Security(get_current_user, scopes=["system", "administrador", "pasajero", "supervisor", "mantenimiento"]))
@m get_tarjetas(self, current_user: dict = Security(get_current_user, scopes=["system", "administrador"]))
@m tarjeta(self, request: Request, id: int = Query(...), current_user: dict = Security(get_current_user, scopes=["system", "administrador", "pasajero"]))

©  src.backend.app.api.routes.card_service.CardCUDService
(f) router
(f) controller
(f) templates
@m __init__(self)
@m index_create(self, request: Request, current_user: dict = Security(get_current_user, scopes=["system", "administrador", "pasajero", "supervisor", "mantenimiento"]))
@m index_update(self, request: Request, current_user: dict = Security(get_current_user, scopes=["system", "administrador"]))
@m index_delete(self, request: Request, current_user: dict = Security(get_current_user, scopes=["system", "administrador"]))
@m create_card(self, id: int = Form(...), tipo: str = Form(...), current_user: dict = Security(get_current_user, scopes=["system", "administrador", "pasajero"]))
@m update_card(self, id: int = Form(...), tipo: str = Form(...), current_user: dict = Security(get_current_user, scopes=["system", "administrador"]))
@m delete_card(self, id: int = Form(...), current_user: dict = Security(get_current_user, scopes=["system", "administrador"]))

```

### 3.7.10. Microservicio: Parada

#### FASE 1: Análisis y Diseño Funcional del Microservicio

- **Microservicio:** stop\_service
- **Tareas puntuales:**
  - **Identificar el bounded context:**
    - El dominio "Parada" incluye nombre, ubicación, descripción, ruta asociada y hora de llegada/salida.
    - Se separa stop\_service de otros servicios relacionados (e.g., RouteService, MapService).
  - **Definir la entidad principal:**

- Entidad: Stop
- Atributos: id, name, location, description, route\_id, arrival\_time, departure\_time, created\_at, updated\_at
- **Delimitar funcionalidad del microservicio:**
  - CRUD completo para paradas.
  - Filtrar paradas por nombre, ubicación y ruta.
  - Obtener detalles de una parada específica.
  - Listar todas las paradas.
  - Crear una nueva parada.
  - Actualizar una parada existente.
- **Diagramar caso de uso:**
  - Usuario crea parada -> Pide datos -> Recibe confirmación.
  - Usuario consulta detalle de parada -> Pide ID de parada -> Recibe detalles.
  - Usuario actualiza parada -> Pide ID de parada y datos -> Recibe confirmación.
  - Usuario elimina parada -> Pide ID de parada -> Recibe confirmación.
  - Usuario lista paradas -> Recibe lista de paradas.
- **Diseñar endpoint inicial (borrador):**
  - GET /stops/{id}: Obtener detalle de la parada.
  - POST /stops: Crear una nueva parada.
  - PUT /stops/{id}: Actualizar una parada.
  - DELETE /stops/{id}: Eliminar una parada.
  - GET /stops: Listar y filtrar paradas.

- **Validar con stakeholders:** \* Revisar con el líder de producto para definir los campos obligatorios, valores válidos, etc.

## FASE 2: Diseño Técnico y de Interfaces

- **Tareas puntuales:**
  - **Diseñar arquitectura interna en 3 capas:**
    - API Layer (FastAPI controllers)
    - Service Layer (StopService)
    - Repository Layer (SQL Database)

## FASE 3: Implementación del Servicio

- Se han creado los servicios asociados a operaciones CRUD mediante endpoints
  - Mostrar formulario de creación.
  - Mostrar formulario de actualización.
  - Mostrar formulario de eliminación.
  - Crear una nueva parada.
  - Actualizar una parada existente.
  - Eliminar una parada.
  - Listar las paradas.
  - Obtener detalles de una parada específica.
- Implementar lógica en StopController y StopService para manejar operaciones sobre paradas
  - Validar la existencia de la parada antes de realizar operaciones.
  - Manejar la lógica de negocio para crear, actualizar y eliminar paradas.

- Implementar la lógica para obtener detalles de una parada.
- Implementar la lógica para listar las paradas.
- Uso del patrón de inyección de dependencias
  - Integración con FastAPI Security para validar los scopes de acceso por rol (Security(get\_current\_user)).
- Se han escrito pruebas unitarias para las operaciones clave
  - Verifica la creación válida de una parada.
  - Comprueba la actualización exitosa de una parada existente.
  - Verifica el error esperado al intentar actualizar una parada inexistente.
  - Valida la eliminación correcta de una parada.
  - Comprueba la respuesta 404 cuando se intenta eliminar una parada no registrada.

#### **FASE 4: Seguridad, Configuración y Middleware**

- Agregar middleware de autenticación con JWT.
  - Se aplicó el siguiente requerimiento a cada ruta: Security(get\_current\_user, scopes=["system", "administrador", "supervisor", "planificador"])
- Separar configuración sensible en .env
  - Se separó la configuración mediante getenv().
- Roles y scopes
  - "system": "Acceso completo al sistema"
  - "administrador": "Permiso para gestionar usuarios y paradas"
  - "supervisor": "Permiso para supervisar las paradas"
  - "planificador": "Permiso para crear y modificar paradas"
- Validación de scopes:

- La función get\_current\_user() se encarga de verificar y validar los scopes.

### Diagrama Estructural:



### 3.7.11. Microservicio: Tipo-Tarjeta

#### FASE 1: Análisis y Diseño Funcional del Microservicio

Microservicio: assign\_type\_card\_service

##### Tareas puntuales:

##### Identificar el bounded context:

El dominio "Tarjeta" incluye Tipo\_tarjeta.

**Definir la entidad principal:**

Entidad: Tipo-tarjeta

Atributos: ID, type

**Delimitar funcionalidad del microservicio:**

Consultar los tipos de tarjeta disponibles (Retornar toda la info)

Consultar tipo de tarjeta unidad.

Crear un nuevo tipo de tarjeta.

Elimina un tipo de tarjeta

Edita un tipo de tarjeta

**Diagramar caso de uso:**

- Sistema consulta tipos de tarjeta -> Hace peticion GET -> Recibe informacion .json de toda la tabla
- Sistema consulta tipo de tarjeta -> Hace peticion GET con id -> Recibe informacion .json de la tarjeta con id especifico
- Sistema añade una tarjeta -> Hace peticion POST con id y tipo -> Recibe confirmacion de creacion
- Sistema elimina una tarjeta -> Hace peticion DELETE con id-> Recibe confirmacion de eliminacion
- Sistema edita una tarjeta -> Hace peticion PUT con id y tipo -> Recibe confirmacion de edicion

nota: Sistema puede cambiarse por administrador

**Diseñar endpoint inicial:**

GET /typecards

GET /typecards/{id}

POST /typecards

PUT /typecards/{id}

DELETE /typecards/{id}

## FASE 2: Diseño Técnico y de Interfaces

### Diseñar arquitectura interna en 3 capas:

API Layer (FastAPI controllers)

Service Layer (TypeCardService)

Repository Layer (.json , sql)

### Definir contrato OpenAPI:

Modelar interfaces (puertos y adaptadores):

✓ ITypeCardRepository con métodos: get\_all(), get\_by\_id(id) ,create(typecard) ,update(id, typecard), delete(id)

- GET /typecards → get\_all()
- GET /typecards/{id} → get\_by\_id(id)
- POST /typecards → create(typecard)
- PUT /typecards/{id} → update(id, typecard)
- DELETE /typecards/{id} → delete(id)

### Aplicar patrón DTO (Data Transfer Object):

- CreateTypeCardDTO
- OutTypeCardDTO

**Usar PlantUML o C4 para diagrama de componentes:**

[API]-->[assign\_type\_card\_service]-->[type\_card\_controller]-->[universal\_controller]

**FASE 3: Implementación del Servicio**

Se desarrollaron los endpoints definidos para el CRUD de tipos de tarjeta usando FastAPI. Se integró un controlador genérico (UniversalController) que interactúa con la capa de persistencia.

Las operaciones implementadas permiten:

Crear, editar, eliminar y consultar tipos de tarjeta, tanto en forma general como individual.

Visualización mediante formularios HTML (/crear, /actualizar, /eliminar).

Se utilizaron DTOs (TypeCardCreate, TypeCardOut) para separación de lógica y presentación. La lógica incluye validaciones, manejo de errores (404, 400, 500) y uso de logging estructurado para trazabilidad.

**FASE 4: Pruebas Unitarias y Validaciones**

Se escribieron pruebas unitarias y de integración para las operaciones críticas del microservicio:

- Validación de creación correcta de tipo de tarjeta.
- Detección de intentos de actualización o eliminación sobre IDs inexistentes.
- Confirmación de lectura individual y total de registros.
- Las pruebas aseguran consistencia funcional y respuesta esperada bajo diferentes escenarios.

**FASE 5: Observabilidad y Telemetría**

Se añadió logging estructurado en todos los endpoints clave para auditoría de acciones por parte de administradores. Las respuestas incluyen detalles del estado de la operación. Aunque no se integraron herramientas externas aún (como Prometheus u OpenTelemetry), la estructura permite fácilmente extender hacia métricas y trazabilidad en siguientes fases.

## Diagrama Estructural

```

© src.backend.app.api.routes.type_card_service.TypeCardQueryService
① router
① controller
② __init__(self)
③ read_all(self, current_user: dict = Security(get_current_user, scopes=["system", "administrador"]))
④ get_by_id(self, id: int, current_user: dict = Security(get_current_user, scopes=["system", "administrador"]))

© src.backend.app.api.routes.type_card_service.TypeCardCUDService
① router
① controller
① templates
② __init__(self)
③ create_typecard_form(self, request: Request, current_user: dict = Security(get_current_user, scopes=["system", "administrador"]))
④ update_typecard_form(self, request: Request, current_user: dict = Security(get_current_user, scopes=["system", "administrador"]))
⑤ delete_typecard_form(self, request: Request, current_user: dict = Security(get_current_user, scopes=["system", "administrador"]))
⑥ create_typecard(self, id: int = Form(...), type: str = Form(...), current_user: dict = Security(get_current_user, scopes=["system", "administrador"]))
⑦ update_typecard(self, id: int = Form(...), type: str = Form(...), current_user: dict = Security(get_current_user, scopes=["system", "administrador"]))
⑧ delete_typecard(self, id: int = Form(...), current_user: dict = Security(get_current_user, scopes=["system", "administrador"]))

```

### 3.7.12. Microservicio: Mantenimiento

#### FASE 1: Análisis y Diseño Funcional del Microservicio

Microservicio: maintenance\_service

##### Tareas puntuales:

##### Identificar el bounded context:

El dominio "Mantenimiento" incluye Tipo\_mantenimiento

##### Definir la entidad principal:

Entidad: Mantenimiento

Atributos: ID\_mantenimiento, ID\_unidad, id\_estado, tipo, fecha\_asignada

##### Delimitar funcionalidad del microservicio:

Create nuevo mantenimiento

Eliminar mantenimiento

Actualizar mantenimiento

Listar el mantenimiento con un ID \_ mantenimiento

Listar todos los mantenimientos

Listar todos los mantenimientos de una unidad

#### **Diagramar caso de uso:**

- Técnico consulta mantenimiento -> pide datos (id,null,id,id\_unidad) -> Retorna datos
- Técnico actualiza información -> pide datos id -> pide(estado) -> retorna confirmación
- Administrador crea, elimina mantenimiento -> pide datos -> retorna confirmación

nota: quizas deba separarse para que el técnico solo pueda consultar y actualizar, administrador cree o elimine.

#### **Diseñar endpoint inicial:**

GET /mantainment/{id}

GET /mantainment/{id\_unidad}

GET /mantainment

POST /mantainment

GET /mantainment/{id}

POST /mantainment/{id}

#### **FASE 2: Diseño Técnico y de Interfaces**

##### **Tareas puntuales:**

##### **Diseñar arquitectura interna en 3 capas:**

API Layer (FastAPI controllers)

Service Layer (MantainmentService)

Repository Layer (.json , sql)

**Definir contrato OpenAPI:**

- ✓ Usar Swagger para definir XXX con schema de validación.

**Modelar interfaces (puertos y adaptadores):**

- ✓ ITypeMantainmentRepository con métodos:

- GET /mantainment → get\_all()
  - Respuesta: MantainmentResponseDTO (Lista de mantenimientos)
- GET /mantainment/{id} → get\_by\_id(id)
  - Respuesta: MantainmentResponseDTO (Un mantenimiento)
- GET /mantainment/unit/{unit\_id} → get\_by\_unit(unit\_id)
  - Respuesta: MantainmentResponseDTO (Lista de mantenimientos por unidad)
- POST /mantainment → create(mantainment)
  - Solicitud: CreateMantainmentDTO
  - Respuesta: MantainmentResponseDTO
- PUT /mantainment/{id} → update(id, mantainment)
  - Solicitud: UpdateMantainmentDTO
  - Respuesta: MantainmentResponseDTO
- DELETE /mantainment/{id} → delete(id)
  - Respuesta: Confirmación de eliminación (mensaje)

**Usar PlantUML o C4 para diagrama de componentes:**

```
[API] --> [mantainment_service] --> [mantainment_controller] --> [universal_controller]-->
[maintain_repository] --> [database]
```

**FASE 3: Implementación del Servicio**

Se implementó la lógica en MantainmentService:

- Crear mantenimiento
- Eliminar mantenimiento
- Actualizar mantenimiento
- Eliminar mantenimiento
- Mostrar mantenimiento

#### **FASE 4: Seguridad, Configuración y Middleware**

**Agregar middleware de autenticación con JWT.**

Se aplicó el siguiente requerimiento a cada ruta

```
Security(get_current_user,scopes=["system", "administrador", "pasajero", "supervisor",  
"mantenimiento"] ))
```

**Separar configuración sensible en .env**

Se separó la configuración mediante getenv()

**Roles y scopes**

```
"system": "Full system access",  
"administrador": "Permission to manage users",  
"pasajero": "Passenger permission",  
"supervisor": "Supervisor permission",  
"mantenimiento": "Maintenance permission",  
"operador": "Operator permission"
```

**Validación de scopes:**

La función get\_current\_user() se encarga de verificar y validar los scopes

**FASE 5: Validación, Observabilidad y Pruebas**

**Ejecutar pruebas de contrato con PyTest:**

Verificar la respuesta esperada al consumir GET /mantainment/crear, POST /mantainment/create, POST /mantainment/update, y POST /mantainment/delete.

Validar que el formato de datos (MantainmentOut) sea consistente con lo acordado por el consumidor del API.

### Añadir logging estructurado:

Se han añadido loggins de eventos relevantes para esclarecer errores y informacion.

### Ejecutar pruebas de contrato con PyTest:

Verificar la respuesta esperada al consumir GET /mantainment/crear, POST /mantainment/create, POST /mantainment/update, y POST /mantainment/delete.

Validar que el formato de datos (MantainmentOut) sea consistente con lo acordado por el consumidor del API.

### Añadir logging estructurado:

Se han añadido loggins de eventos relevantes para esclarecer errores y informacion.

## Diagrama Estructural

```

    ⚡ src.backend.app.api.routes.maintance_service.MaintainanceQueryService
    ⓘ router
    ⓘ controller
    ⓘ __init__(self)
    ⓘ get_all(self, current_user: dict = Security(get_current_user, scopes=["system", "administrador", "tecnico"]))
    ⓘ get_by_id(self, id: int, current_user: dict = Security(get_current_user, scopes=["system", "administrador", "tecnico"]))
    ⓘ get_by_unit(self, unit_id: int, current_user: dict = Security(get_current_user, scopes=["system", "administrador", "tecnico"]))

    ⚡ src.backend.app.api.routes.maintainance_service.MaintainanceCUDService
    ⓘ router
    ⓘ controller
    ⓘ templates
    ⓘ __init__(self)
    ⓘ crear_mantenimiento(self, request: Request, current_user: dict = Security(get_current_user, scopes=["system", "administrador", "supervisor", "tecnico", "operador"]))
    ⓘ actualizar_mantenimiento(self, request: Request, current_user: dict = Security(get_current_user, scopes=["system", "administrador", "tecnico"]))
    ⓘ eliminar_mantenimiento(self, request: Request, current_user: dict = Security(get_current_user, scopes=["system", "administrador", "tecnico"]))
    ⓘ add(self, id_unit: int = Form(...), id_status: int = Form(...), type: str = Form(...), date: datetime = Form(...), current_user: dict = Security(get_current_user, scopes=["system", "administrador", "tecnico"]))
    ⓘ update(self, id: int = Form(...), id_unit: int = Form(...), id_status: int = Form(...), type: str = Form(...), date: datetime = Form(...), current_user: dict = Security(get_current_user, scopes=["system", "administrador", "tecnico"]))
    ⓘ delete_maintenance(self, id: int = Form(...), current_user: dict = Security(get_current_user, scopes=["system", "administrador", "tecnico"]))

```

### 3.7.13. Microservicio: Movimiento

## FASE 1: Análisis y Diseño Funcional del Microservicio

Microservicio: movement\_service

### Tareas puntuales:

#### Identificar el bounded context:

El dominio "Movimiento" incluye Tipo\_Movimiento.

#### Definir la entidad principal:

Entidad: Movimiento

Atributos: ID, Tipo\_Movimiento, Monto

#### Delimitar funcionalidad del microservicio:

Consultar los movimientos registrados, teniendo en cuenta o no una tarjeta en específico.

Crear un nuevo movimiento.

#### Diagramar caso de uso:

- Sistema consulta de movimientos -> Hace petición GET -> Recibe información .json de toda la tabla
- Sistema consulta tipo de tarjeta -> Hace petición GET con id -> Recibe información .json de la tarjeta con id específico
- Sistema añade un movimiento -> Hace petición POST con id y tipo -> Recibe confirmación de creación.

#### Diseñar endpoint inicial (borrador):

GET /movement

GET /movement/{id}

POST /movement/{id}

**Validar con stakeholders: (Pendiente con Edwin)**

Revisión con líder de producto para ajustar estados válidos:

Created

Paid

Cancelled

delivered

nota: Sistema puede cambiarse por administrador

**FASE 2: Diseño Técnico y de Interfaces****Tareas puntuales:****Diseñar arquitectura interna en 3 capas:**

API Layer (FastAPI controllers)

Service Layer (TypeCardService)

Repository Layer (.db , sql)

**Definir contrato OpenAPI:**

- ✓ Usar Swagger para definir XXX con schema de validación.

**Modelar interfaces (puertos y adaptadores):**

- ✓ ITypeCardRepository con métodos: get\_all(), get\_by\_id(id) ,create(id\_card)
  - GET /movement → get\_all()
  - GET /movement/{id} → get\_by\_id(id)
  - POST /movement/{id}→ create(id\_card)

**Aplicar patrón DTO (Data Transfer Object):**

- CreateTypeMovementDTO

**Usar PlantUML o C4 para diagrama de componentes:**

```
[API]-->[movement_service]-->[type_card_controller]-->[universal_controller]
```

**FASE 3: Implementación del Servicio**

Se desarrollaron los endpoints definidos para el CRUD de movimientos usando FastAPI. Se integró un controlador genérico UniversalController que interactúa con la base de datos. Las operaciones implementadas incluyen:

- Creación, edición, eliminación y consulta de movimientos.
- Validaciones de datos y manejo de errores (400, 404, 500).
- Uso de DTOs (MovementCreate, MovementOut) para separación de lógica y presentación.

**FASE 4: Pruebas Unitarias y Validaciones**

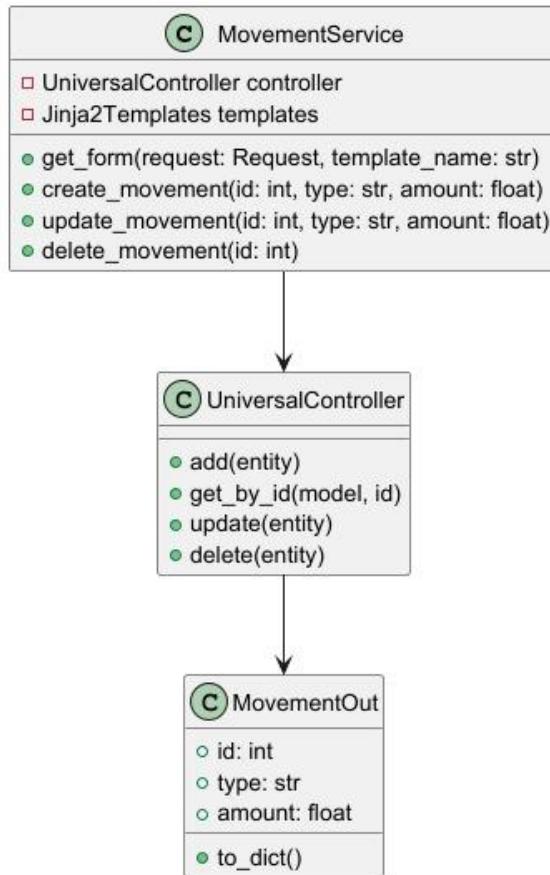
Se diseñaron pruebas unitarias y de integración que validan: Creación correcta de movimientos con atributos válidos. Intentos de actualización o eliminación sobre **IDs inexistentes**. Confirmación de lectura **individual** y **total** de registros. Verificación de respuestas esperadas bajo múltiples escenarios.

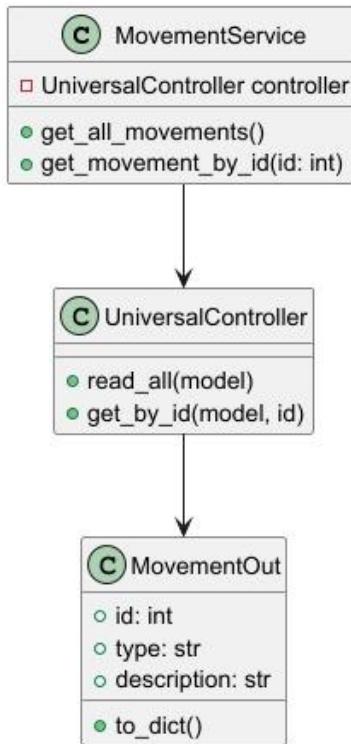
Las pruebas garantizan **consistencia funcional** y respuestas correctas en distintos casos.

**FASE 5: Observabilidad y Telemetría**

Se incorporó logging estructurado en los endpoints clave para auditoría de acciones administrativas. Los logs incluyen detalles del estado de cada operación. La arquitectura permite integración futura con herramientas externas (Prometheus, OpenTelemetry) para métricas y trazabilidad.

## Diagrama Estructural





### 3.7.14. Microservicio: TipoMovimiento

#### FASE 1: Análisis y Diseño Funcional del Microservicio

Microservicio: assign\_type\_movement\_service

**Tareas puntuales:**

**Identificar el bounded context:**

El dominio "Movimiento" incluye Tipo\_Movimiento.

**Definir la entidad principal:**

Entidad: Tipo-Movimiento

Atributos: ID, type

**Delimitar funcionalidad del microservicio:**

Consultar los tipos de movimientos disponibles (Retornar toda la info)

Consultar el tipo de movimiento de la unidad.

Crear un nuevo tipo de movimiento.

Elimina un tipo de movimiento

Edita un tipo de movimiento

#### **Diagramar caso de uso:**

- Sistema consulta tipos de movimiento -> Hace petición GET -> Recibe información .json de toda la tabla
- Sistema consulta tipo de movimiento -> Hace petición GET con id -> Recibe información .json del tipo de movimiento con id específico
- Sistema añade un tipo de movimiento-> Hace petición POST con id y tipo -> Recibe confirmación de creación
- Sistema elimina un tipo de movimiento -> Hace petición DELETE con id-> Recibe confirmación de eliminación
- Sistema edita un tipo de movimiento -> Hace petición PUT con id y tipo -> Recibe confirmación de edición

nota: Sistema puede cambiarse por administrador

#### **Diseñar endpoint inicial (borrador):**

GET /typemovement

GET /typemovement/{id}

POST /typemovement

PUT /typemovement/{id}

DELETE /typemovement/{id}

#### **Validar con stakeholders: (Pendiente con Edwin)**

Revisión con líder de producto para ajustar estados válidos:

Created

Paid

Cancelled

delivered

## FASE 2: Diseño Técnico y de Interfaces

### Tareas puntuales:

#### Diseñar arquitectura interna en 3 capas:

API Layer (FastAPI controllers)

Service Layer (TypeCardService)

Repository Layer (.db , sql)

#### Definir contrato OpenAPI:

- ✓ Usar Swagger para definir XXX con schema de validación.

#### Modelar interfaces (puertos y adaptadores):

✓ ITypeCardRepository con métodos: get\_all(), get\_by\_id(id) ,create(typecard)  
,update(id, typecard), delete(id)

- GET /typemovement → get\_all()
- GET /typemovement/{id} → get\_by\_id(id)
- POST /typemovement → create(typecard)
- PUT /typemovement/{id} → update(id, typecard)
- DELETE /typemovement/{id} → delete(id)

#### Aplicar patrón DTO (Data Transfer Object):

- CreateMovementDTO

- UpdateTypeMovementDTO.
- TypeMovementResponseDTO

### Usar PlantUML o C4 para diagrama de componentes:

```
[API]-->[assign_type_movement_service]-->[type_movement]-->[universal_controll  
er]
```

### FASE 3: Implementación del Servicio

Se desarrollaron los endpoints definidos para el CRUD de tipos de movimientos usando FastAPI. Se integró un controlador genérico UniversalController que interactúa con la base de datos. Las operaciones implementadas incluyen:

- Creación, edición, eliminación y consulta de tipos de movimientos.
- Validaciones de datos y manejo de errores (400, 404, 500).
- Uso de DTOs (TypeMovementCreate, TypeMovementOut) para separación de lógica y presentación.

### FASE 4: Pruebas Unitarias y Validaciones

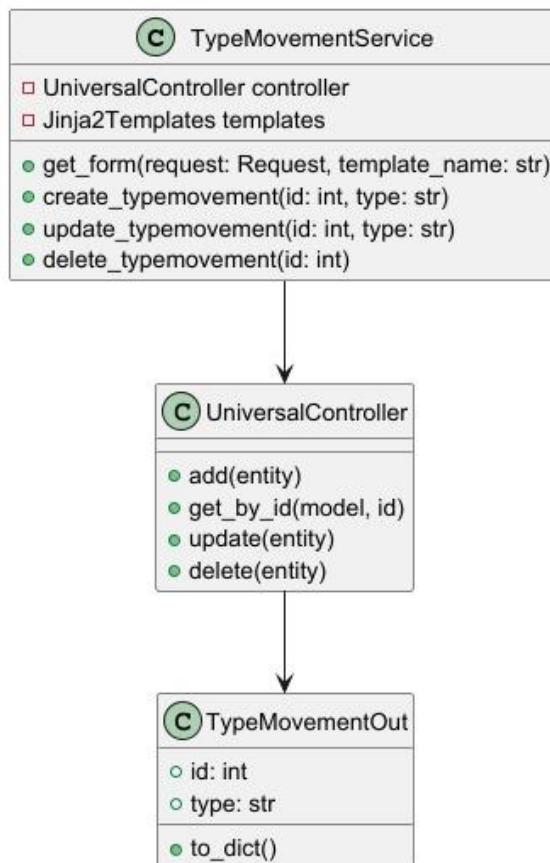
Se diseñaron pruebas unitarias y de integración que validan: Creación correcta de tipos de movimientos con atributos válidos. Intentos de actualización o eliminación sobre **IDs inexistentes**. Confirmación de lectura **individual** y **total** de registros. Verificación de respuestas esperadas bajo múltiples escenarios.

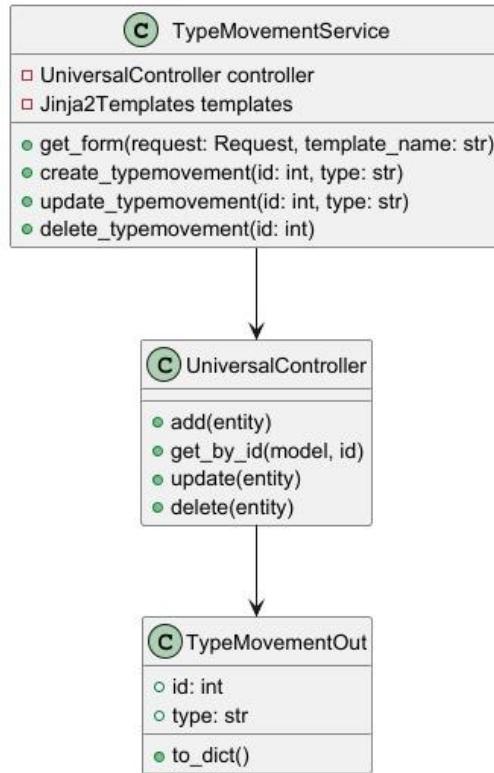
Las pruebas garantizan **consistencia funcional** y respuestas correctas en distintos casos.

## FASE 5: Observabilidad y Telemetría

Se incorporó logging estructurado en los endpoints clave para auditoría de acciones administrativas. Los logs incluyen detalles del estado de cada operación. La arquitectura permite integración futura con herramientas externas (Prometheus, OpenTelemetry) para métricas y trazabilidad.

## Diagrama Estructural





### 3.7.15. Microservicio: Precio

#### FASE 1: Análisis y Diseño Funcional del Microservicio

Microservicio: assign\_price\_service

**Tareas puntuales:**

**Identificar el bounded context:**

El dominio "Price" incluye Tipo\_UnidadTransporte.

**Definir la entidad principal:**

Entidad: Price

Atributos: ID, UnitTransportType, Monto

**Delimitar funcionalidad del microservicio:**

Consultar los tipos de precios de todas los servicios disponibles (Retornar toda la info)

Consultar precio de un servicio en específico (Ruta específica)

Crear un nuevo precio para un servicio

Elimina un precio para un servicio

Edita un precio para un servicio

**Diagramar caso de uso:**

- Sistema consulta precios -> Hace petición GET -> Recibe información .json de toda la tabla
- Sistema consulta precio por servicio en específico -> Hace petición GET con id -> Recibe información .json de precio con id específico
- Sistema añade un precio -> Hace petición POST con id y tipo -> Recibe confirmación de creación
- Sistema elimina un precio-> Hace petición DELETE con id-> Recibe confirmación de eliminación
- Sistema edita un precio -> Hace petición PUT con id y tipo -> Recibe confirmación de edición

nota: Sistema puede cambiarse por administrador

**Diseñar endpoint inicial (borrador):**

GET /price

GET /price/{id}

POST /price

PUT /price/{id}

DELETE /price/{id}

### Validar con stakeholders: (Pendiente con Edwin)

Revisión con líder de producto para ajustar estados válidos:

Created

Paid

Cancelled

delivered

## FASE 2: Diseño Técnico y de Interfaces

### Tareas puntuales:

#### Diseñar arquitectura interna en 3 capas:

API Layer (FastAPI controllers)

Service Layer (TypeCardService)

Repository Layer (.db , sql)

#### Definir contrato OpenAPI:

- ✓ Usar Swagger para definir XXX con schema de validación.

#### Modelar interfaces (puertos y adaptadores):

✓ ITypeCardRepository con métodos: get\_all(), get\_by\_id(id) ,create() ,update(id), delete(id)

- GET /price → get\_all()
- GET /price/{id} → get\_by\_id(id)
- POST /price → create()

- PUT /price/{id} → update(id)
- DELETE /price/{id} → delete(id)

#### Aplicar patrón DTO (Data Transfer Object):

- CreatePriceDTO
- UpdatePriceDTO.
- PriceResponseDTO

#### Usar PlantUML o C4 para diagrama de componentes:

```
[API]-->[assign_price_service]-->[price_controller]-->[universal_controller]
```

#### FASE 3: Implementación del Servicio

Se desarrollaron los endpoints definidos para el CRUD de precios usando FastAPI. Se integró un controlador genérico UniversalController que interactúa con la base de datos. Las operaciones implementadas incluyen:

- Creación, edición, eliminación y consulta de precios.
- Validaciones de datos y manejo de errores (400, 404, 500).
- Uso de DTOs (PriceCreate, PriceOut) para separación de lógica y presentación.

#### FASE 4: Pruebas Unitarias y Validaciones

Se diseñaron pruebas unitarias y de integración que validan:

- Creación correcta de precios con atributos válidos.
- Intentos de actualización o eliminación sobre IDs inexistentes.
- Confirmación de lectura individual y total de registros.
- Verificación de respuestas esperadas bajo múltiples escenarios.

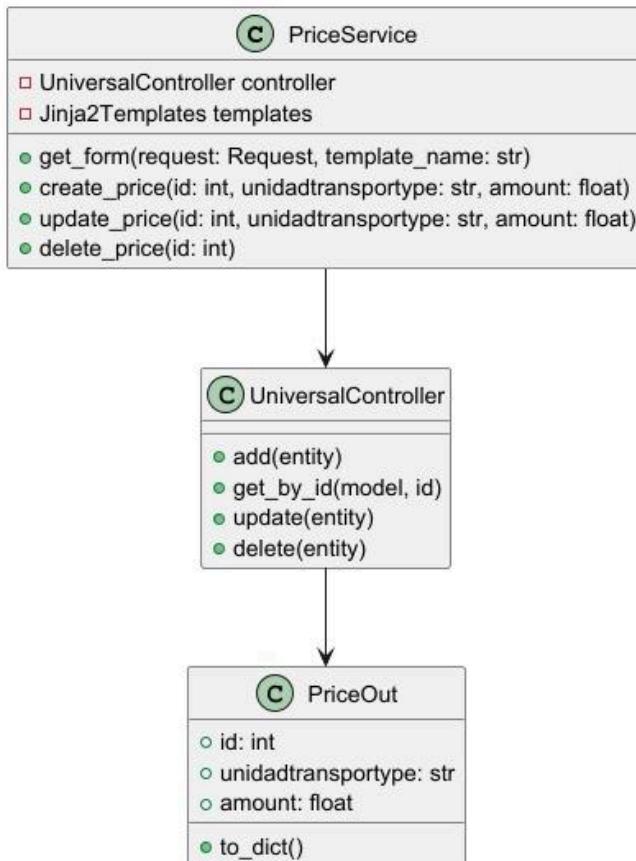
Las pruebas garantizan consistencia funcional y respuestas correctas en distintos casos.

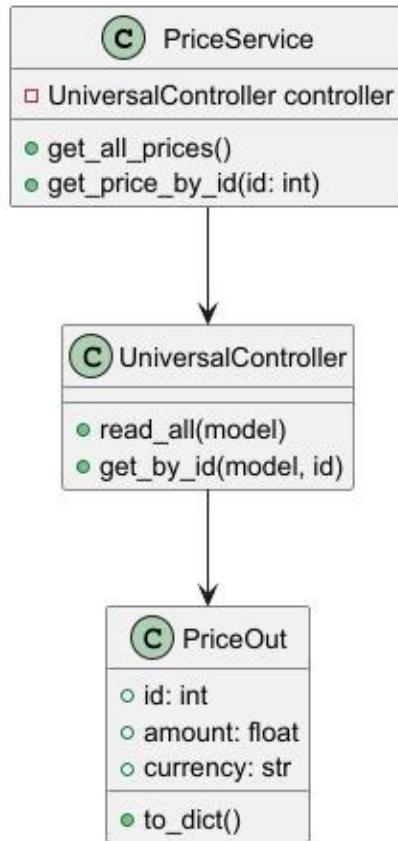
## FASE 5: Observabilidad y Telemetría

Se incorporó logging estructurado en los endpoints clave para auditoría de acciones administrativas.

Los logs incluyen detalles del estado de cada operación. La arquitectura permite integración futura con herramientas externas (Prometheus, OpenTelemetry) para métricas y trazabilidad.

## Diagrama Estructural





### 3.7.16. Microservicio: TipoTransporte

#### FASE 1: Análisis y Diseño Funcional del Microservicio

Microservicio: assign\_type\_transport\_service

**Tareas puntuales:**

**Identificar el bounded context:**

El dominio "TransportUnit" incluye Tipo\_unidadTransporte.

**Definir la entidad principal:**

Entidad: Tipo-Transporte

Atributos: ID, type

**Delimitar funcionalidad del microservicio:**

Consultar los tipos de transporte disponibles (Retornar toda la info)

Consultar tipo de transporte por id.

Crear un nuevo tipo de transporte.

Elimina un tipo de transporte

Edita un tipo de transporte

**Diagramar caso de uso:**

- Sistema consulta tipos de transporte -> Hace petición GET -> Recibe información .json de toda la tabla
- Sistema consulta tipo de transporte -> Hace petición GET con id -> Recibe información .json de transporte con id específico
- Sistema añade un tipo transporte -> Hace petición POST con id y tipo -> Recibe confirmación de creación
- Sistema elimina un tipo transporte -> Hace petición DELETE con id-> Recibe confirmación de eliminación
- Sistema edita un tipo transporte -> Hace petición PUT con id y tipo -> Recibe confirmación de edición

nota: Sistema puede cambiarse por administrador

**Diseñar endpoint inicial (borrador):**

GET /typetransportunit

GET /ttypetransportunit/{id}

POST /typetransportunit

PUT /typetransportunit/{id}

DELETE /typetransportunit/{id}

**Validar con stakeholders: (Pendiente con Edwin)**

Revisión con líder de producto para ajustar estados válidos:

Created

Paid

Cancelled

delivered

**FASE 2: Diseño Técnico y de Interfaces**

**Tareas puntuales:**

**Diseñar arquitectura interna en 3 capas:**

API Layer (FastAPI controllers)

Service Layer (TypeCardService)

Repository Layer (.db , sql)

**Definir contrato OpenAPI:**

- ✓ Usar Swagger para definir XXX con schema de validación.

**Modelar interfaces (puertos y adaptadores):**

✓ ITypeTransportUnitRepository con métodos: get\_all(), get\_by\_id(id)

,create(typecard) ,update(id, typecard), delete(id)

- GET /typetransportunit → get\_all()
- GET /typetransportunit /{id} → get\_by\_id(id)
- POST /typetransportunit → create(typecard)
- PUT /typetransportunit /{id} → update(id, typecard)

- DELETE /typetransportunit/{id} → delete(id)

**Aplicar patrón DTO (Data Transfer Object):**

- CreateTypeTransportUnit DTO
- UpdateTypeTransportUnitDTO.
- TypeTransportUnitdResponseDTO

**Usar PlantUML o C4 para diagrama de componentes:**

```
[API]-->[assign_type_transport_service]-->[type_transport_controller]-->[universal_controller]
```

**FASE 3: Implementación del Servicio**

Se desarrollaron los endpoints definidos para el CRUD de tipos de transporte usando FastAPI. Se integró un controlador genérico UniversalController que interactúa con la base de datos. Las operaciones implementadas incluyen:

- Creación, edición, eliminación y consulta de tipos de transporte.
- Validaciones de datos y manejo de errores (400, 404, 500).
- Uso de DTOs (TypeTransportCreate, TypeTransportOut) para separación de lógica y presentación.

**FASE 4: Pruebas Unitarias y Validaciones**

Se diseñaron pruebas unitarias y de integración que validan:

- Creación correcta de tipos de transporte con atributos válidos.

- Intentos de actualización o eliminación sobre **IDs inexistentes**.
- Confirmación de lectura **individual** y **total** de registros. Verificación de respuestas esperadas bajo múltiples escenarios.

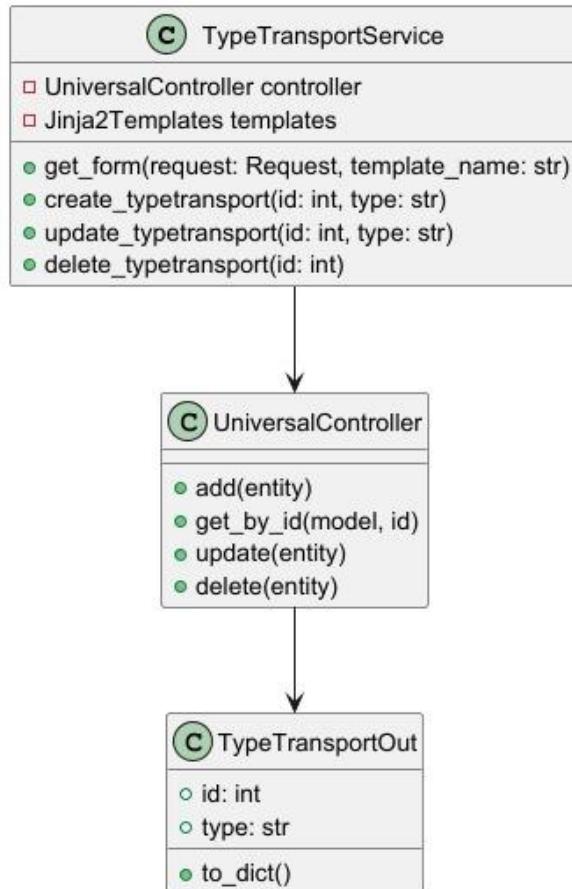
Las pruebas garantizan **consistencia funcional** y respuestas correctas en distintos casos.

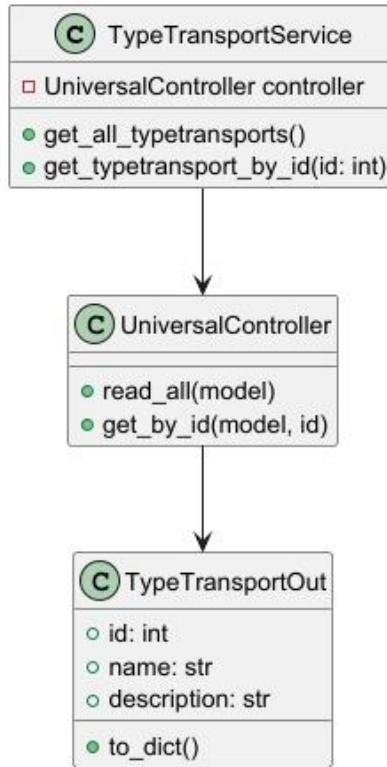
## **FASE 5: Observabilidad y Telemetría**

Se incorporó logging estructurado en los endpoints clave para auditoría de acciones administrativas.

Los logs incluyen detalles del estado de cada operación. La arquitectura permite integración futura con herramientas externas (Prometheus, OpenTelemetry) para métricas y trazabilidad

## Diagrama Estructural





### 3.7.17. Microservicio: RolUsuario

#### FASE 1: Análisis y Diseño Funcional del Microservicio

Microservicio: assign\_rol\_user\_service

**Tareas puntuales:**

**Identificar el bounded context:**

El dominio "Usuario" incluye Rol\_Usuario.

**Definir la entidad principal:**

Entidad: Rol\_Usuario

Atributos: ID, type

**Delimitar funcionalidad del microservicio:**

Consultar los tipos de usuarios disponibles (Retornar toda la info)

Consultar tipo de usuarios por id.

Crear un nuevo tipo de usuario.

Elimina un tipo de usuario

Edita un tipo de usuario

#### **Diagramar caso de uso:**

- Sistema consulta tipos de usuarios -> Hace petición GET -> Recibe información .json de toda la tabla
- Sistema consulta tipo de usuario -> Hace petición GET con id -> Recibe información .json de tipo usuario con id específico
- Sistema añade tipo usuario -> Hace petición POST con id y tipo -> Recibe confirmación de creación
- Sistema elimina un tipo usuario -> Hace petición DELETE con id-> Recibe confirmación de eliminación
- Sistema edita un tipo usuario -> Hace petición PUT con id y tipo -> Recibe confirmación de edición

nota: Sistema puede cambiarse por administrador

#### **Diseñar endpoint inicial (borrador):**

GET /typeuser

GET /typeuser/{id}

POST /typeuser

PUT /typeuser/{id}

DELETE /typeuser/{id}

#### **Validar con stakeholders: (Pendiente con Edwin)**

Revisión con líder de producto para ajustar estados válidos:

Created

Paid

Cancelled

delivered

## FASE 2: Diseño Técnico y de Interfaces

### Tareas puntuales:

#### Diseñar arquitectura interna en 3 capas:

API Layer (FastAPI controllers)

Service Layer (TypeUserService)

Repository Layer (.db , sql)

#### Definir contrato OpenAPI:

- ✓ Usar Swagger para definir XXX con schema de validación.

#### Modelar interfaces (puertos y adaptadores):

- ✓ ITypeUserRepository con métodos: get\_all(), get\_by\_id(id) ,create(typeuser), update(id, typeuser), delete(id)

- GET /typeuser → get\_all()
- GET /typeuser/{id} → get\_by\_id(id)
- POST /typeuser → create(typeuser)
- PUT /typeuser/{id} → update(id, typeuser)
- DELETE /typeuser/{id} → delete(id)

#### Aplicar patrón DTO (Data Transfer Object):

- CreateTypeUserDTO

- UpdateTypeUserdDTO.
- TypeUserResponseDTO

### Usar PlantUML o C4 para diagrama de componentes:

```
[API]-->[assign_type_user_service]-->[type_user_controller]-->[universal_controller]  
]
```

### FASE 3: Implementación del Servicio

Se desarrollaron los endpoints definidos para el CRUD de roles de usuario usando FastAPI. Se integró un controlador genérico UniversalController que interactúa con la base de datos. Las operaciones implementadas incluyen:

- Creación, edición, eliminación y consulta de roles de usuario.
- Visualización mediante formularios HTML (/crear, /actualizar, /eliminar).
- Validaciones de datos y manejo de errores (400, 404, 500).
- Uso de DTOs (RolUserCreate, RolUserOut) para separación de lógica y presentación.

### FASE 4: Pruebas Unitarias y Validaciones

Se diseñaron pruebas unitarias y de integración que validan:

- Creación correcta de roles de usuario con atributos válidos.
- Intentos de actualización o eliminación sobre **IDs inexistentes**.
- Confirmación de lectura **individual** y **total** de registros.
- Verificación de respuestas esperadas bajo múltiples escenarios.

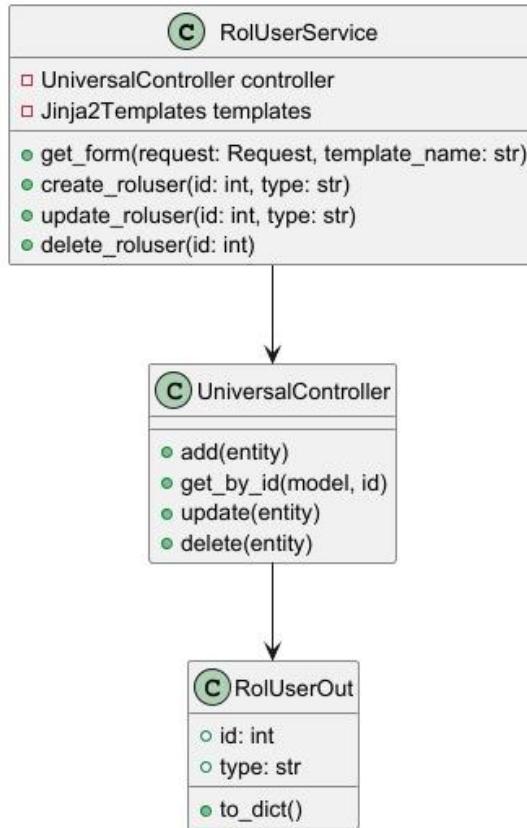
Las pruebas garantizan **consistencia funcional** y respuestas correctas en distintos casos.

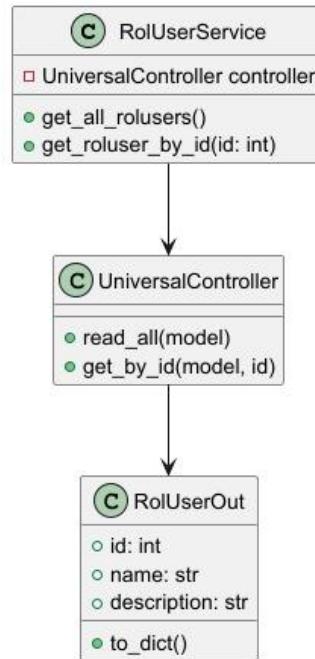
**FASE 5: Observabilidad y Telemetría**

Se incorporó logging estructurado en los endpoints clave para auditoría de acciones administrativas.

Los logs incluyen detalles del estado de cada operación. Aunque aún no se integran herramientas externas como Prometheus u OpenTelemetry, la arquitectura permite una futura integración para métricas y trazabilidad.

## Diagrama Estructural





### 3.7.18. Microservicio: Usuario

#### FASE 1: Análisis y Diseño Funcional del Microservicio

Microservicio: assign\_user\_service

##### Tareas puntuales:

##### Identificar el bounded context:

El dominio "User" incluye Tipo\_Usuario.

##### Definir la entidad principal:

Entidad: User

Atributos: ID, name, lastname, identification, email, password, phone number, IDtype\_user, ID, IDturno.

##### Delimitar funcionalidad del microservicio:

Consultar los usuarios disponibles (Retornar toda la info)

Consultar el usuario teniendo en cuenta el id.

Crear un nuevo usuario.

Elimina un usuario.

Edita información de un usuario.

#### **Diagramar caso de uso:**

- Sistema consulta usuarios -> Hace petición GET -> Recibe información .json de toda la tabla
- Sistema consulta usuarios con base al id -> Hace petición GET con id -> Recibe información .json de la tarjeta con id específico
- Sistema añade un usuario -> Hace petición POST con id y tipo -> Recibe confirmación de creación
- Sistema elimina una tarjeta -> Hace petición DELETE con id-> Recibe confirmación de eliminación
- Sistema edita informacion de un usuario-> Hace petición PUT con id, valor y atributo-> Recibe confirmación de edición

nota: Sistema puede cambiarse por administrador

#### **Diseñar endpoint inicial (borrador):**

GET /user

GET /user/{id}

POST /user

PUT /user/{id}

DELETE /user/{id}

**Validar con stakeholders: (Pendiente con Edwin)**

Revisión con líder de producto para ajustar estados válidos:

Created

Paid

Cancelled

delivered

**FASE 2: Diseño Técnico y de Interfaces****Tareas puntuales:****Diseñar arquitectura interna en 3 capas:**

API Layer (FastAPI controllers)

Service Layer (TypeCardService)

Repository Layer (.db , sql)

**Definir contrato OpenAPI:**

- ✓ Usar Swagger para definir XXX con schema de validación.

**Modelar interfaces (puertos y adaptadores):**

✓ I UserRepository con métodos: get\_all(), get\_by\_id(id) ,create(user) ,update(id, user), delete(id)

- GET /user → get\_all()
- GET /user/{id} → get\_by\_id(id)
- POST /user → create(user)
- PUT /user/{id} → update(id, user)
- DELETE /user/{id} → delete(id)

**Aplicar patrón DTO (Data Transfer Object):**

- CreateUserDTO
- UpdateUserDTO.
- UserResponseDTO

**Usar PlantUML o C4 para diagrama de componentes:**

```
[API]-->[user_service]-->[user_controller]-->[universal_controller]
```

**FASE 3: Implementación del Servicio**

Se desarrollaron los endpoints definidos para el CRUD de usuarios usando FastAPI. Se integró un controlador genérico UniversalController que interactúa con la base de datos. Las operaciones implementadas incluyen:

- Creación, edición, eliminación y consulta de usuarios.
- Visualización mediante formularios HTML (/crear, /actualizar, /eliminar).
- Validaciones de datos y manejo de errores (400, 404, 500).
- Uso de DTOs (UserCreate, UserOut) para separación de lógica y presentación.

**FASE 4: Pruebas Unitarias y Validaciones**

Se diseñaron pruebas unitarias y de integración que validan:

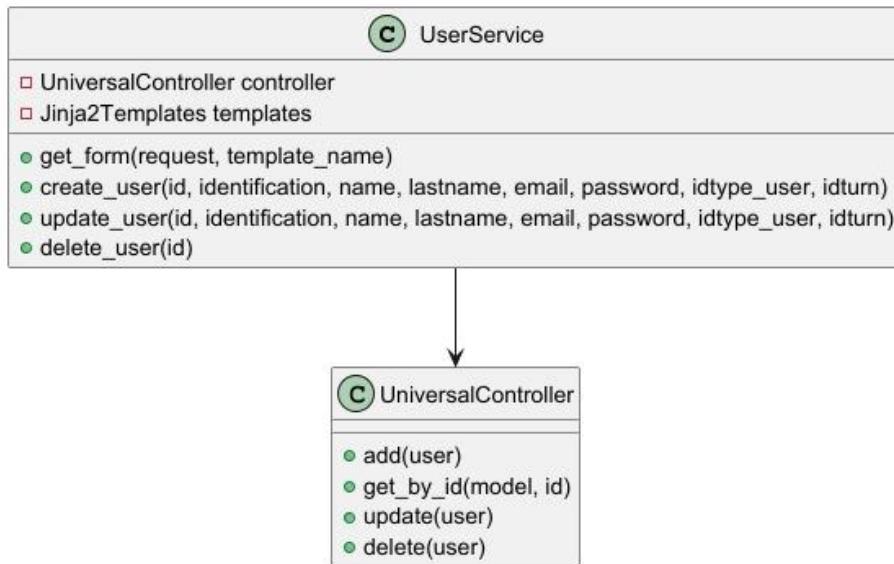
- Creación correcta de usuarios con atributos válidos.
- Intentos de actualización o eliminación sobre IDs inexistentes.
- Confirmación de lectura individual y total de usuarios.
- Verificación de respuestas esperadas bajo múltiples escenarios.

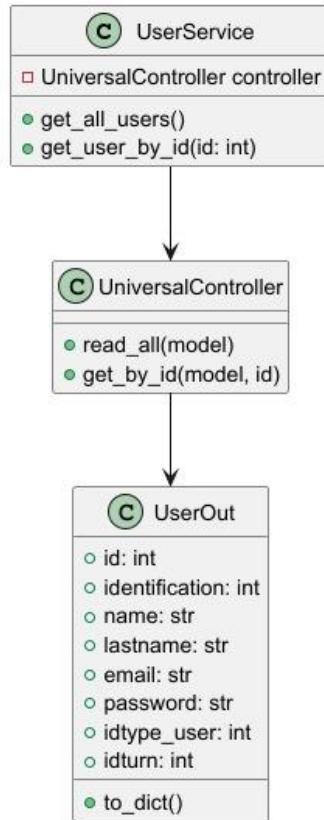
Las pruebas garantizan consistencia funcional y respuestas correctas en distintos casos.

## FASE 5: Observabilidad y Telemetría

Se incorporó logging estructurado en los endpoints clave para auditoría de acciones administrativas. Los logs incluyen detalles del estado de cada operación. Aunque aún no se integran herramientas externas como Prometheus u OpenTelemetry, la arquitectura permite una futura integración para métricas y trazabilidad.

## Diagrama Estructural





### 3.7.19. Microservicio: PQR

#### FASE 1: Análisis y Diseño Funcional del Microservicio

Microservicio: pqr\_service

##### Identificar el bounded context:

El dominio "PQR" abarca el registro, consulta, modificación y eliminación de Peticiones, Quejas y Reclamos de los usuarios del sistema. Cada PQR está asociado a un usuario y puede ser gestionado por administradores.

Este microservicio está desacoplado de otros servicios como user\_service, ticket\_service y notification\_service.

##### Definir la entidad principal:

Entidad: PQR

Atributos:

- id
- tipo\_pqr (Petición, Queja, Reclamo)
- descripcion
- fecha\_creacion
- id\_usuario
- estado (pendiente, en proceso, resuelto, cerrado)

##### Delimitar funcionalidad del microservicio:

- Formulario HTML para crear un PQR (según rol).

- Formulario HTML para actualizar un PQR.
- Formulario HTML para eliminar un PQR.
- Consultar PQRs por ID.
- Consultar PQRs por usuario.
- Listar PQRs (por rol: admin o pasajero).
- Validación previa a operaciones (ej. no actualizar/eliminar inexistente).
- Redirección a vistas de consulta o confirmación.

**Diagramar casos de uso:**

- Pasajero crea PQR → ingresa datos → recibe confirmación.
- Admin actualiza/elimina PQR → busca por ID → edita/elimina → recibe respuesta.
- Ambos consultan PQRs → por usuario o por ID → se listan en tabla.
- El Sistema lista todos los PQRs → según rol del usuario autenticado.

**Diseñar endpoints iniciales:**

- GET /pqr/administrador/crear: Formulario para crear PQR (admin)
- GET /pqr/pasajero/crear: Formulario para crear PQR (pasajero)

- POST /pqr/create: Crear nuevo PQR
- GET /pqr/actualizar: Formulario actualizar PQR
- POST /pqr/update: Actualizar PQR
- GET /pqr/eliminar: Formulario eliminar PQR
- POST /pqr/delete: Eliminar PQR
- GET /pqr/consultar: Formulario general de consulta
- GET /pqr/consultar/administrador: Consultar como admin
- GET /pqr/consultar/pasajero: Consultar como pasajero
- GET /pqr/pasajero/pqrs: Listar PQRs de pasajero
- GET /pqr/administrador/pqrs: Listar PQRs como admin
- GET /pqr/find?ID=...: Buscar PQR por ID
- GET /pqr/user?iduser=...: Buscar PQRs por ID de usuario

**Validar con stakeholders:**

Revisar con el Product Owner:

- Estados válidos de un PQR (pendiente, en proceso, resuelto, cerrado)

- Tipos válidos de PQR (Petición, Queja, Reclamo)
- Fluxos diferenciados para usuarios y administradores

## FASE 2: Diseño Técnico y de Interfaces

Arquitectura en 3 capas:

- API Layer: Endpoints HTML con FastAPI
- Service Layer: Controlador universal para operaciones CRUD (UniversalController)
- Repository Layer: Base de datos SQL Server (via PYODBC o similar)

Plantillas HTML:

- Separadas por tipo de usuario (pasajero / administrador)
- Templates reutilizables: formulario.html, lista\_pqr.html, confirmacion.html, etc.
- Uso de Jinja2 para renderizar dinámicamente los datos.

Componentes:

- Controlador FastAPI (pqr\_controller.py)
- Servicio de base de datos (UniversalController)
- Templates HTML (templates/pqr/)
- Middleware y seguridad (comentada en esta etapa, pero lista para integración)

## FASE 3: Implementación del Servicio

Funcionalidad implementada:

- Crear nuevo PQR
- Actualizar PQR existente
- Eliminar PQR
- Consultar por ID
- Consultar por usuario
- Listar todos los PQRs según el rol
- Formularios HTML diferenciados
- Validaciones de entrada
- Redirecciones tras operaciones
- Control centralizado con UniversalController

#### Detalles clave:

- Validación previa en actualización y eliminación (verifica existencia)
- Consulta por ID y por usuario con parámetros de query (?ID=, ?iduser=)
- Retorno de templates renderizados
- Manejadores de error básicos (404 si no existe el registro)

Pruebas realizadas:

- ✓ Creación válida desde vistas de pasajero y admin
- ✓ Actualización exitosa con ID válido
- ✓ Eliminación correcta y redirección
- ✓ Manejo de error al intentar operar con ID inexistente
- ✓ Consulta individual y por usuario
- ✓ Listado completo para cada tipo de usuario

#### FASE 4: Seguridad, Configuración y Middleware

Seguridad:

- Se preparó el uso de JWT para autenticación

Cada ruta puede protegerse con:

```
Security(get_current_user, scopes=["administrador", "pasajero"])
```

- Actualmente comentado en endpoints por fines de prueba

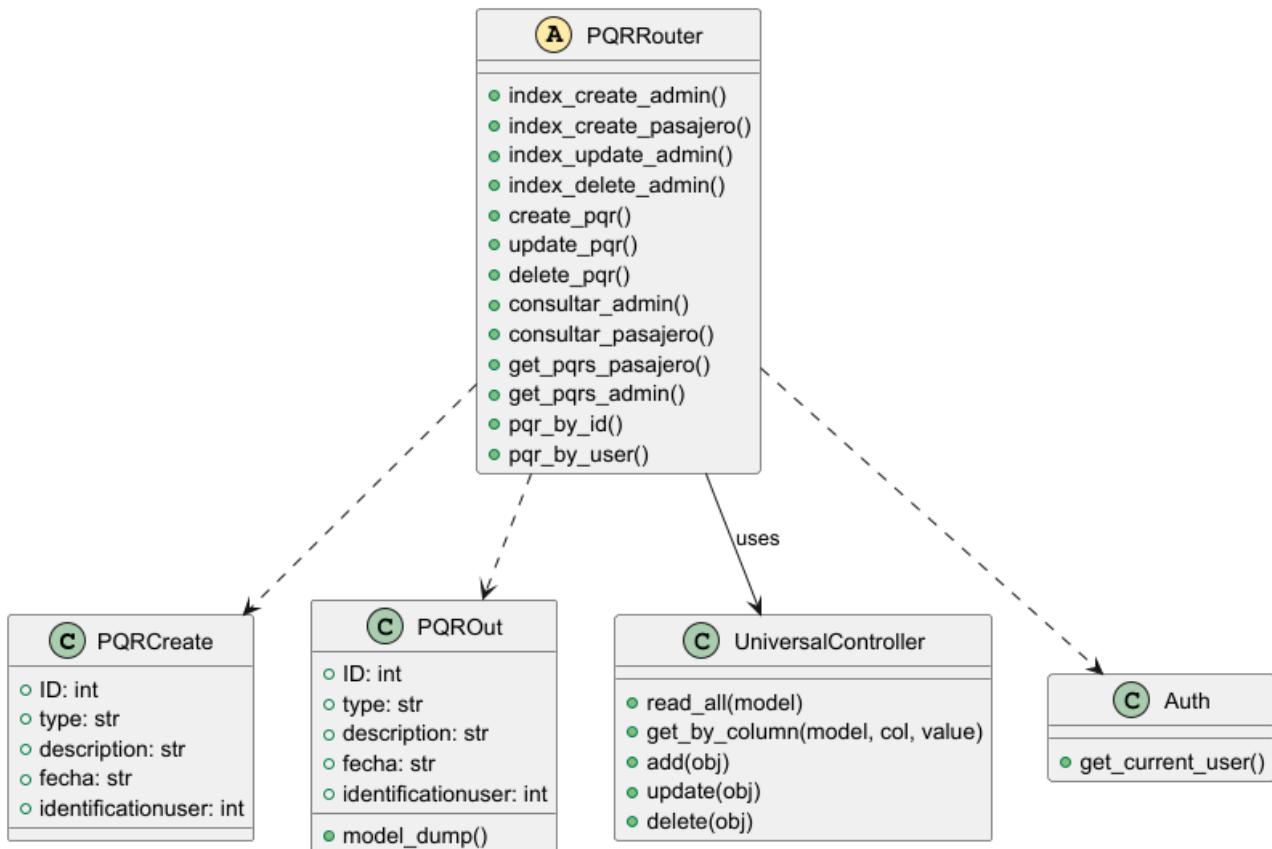
Roles y Scopes definidos:

```
{  
    "system": "Acceso total",  
    "administrador": "Gestión completa de PQRs",  
    "pasajero": "Creación y consulta de sus propios PQRs"  
}
```

Middleware y configuración:

- Configuración sensible preparada para .env (getenv)
- JWT y validación de scopes por función get\_current\_user()
- Patrón de inyección de dependencias presente para futura integración

## Diagrama Estructural



### 3.7.20. Microservicio: Rendimiento

#### FASE 1: Análisis funcional

##### Objetivo del Servicio

Gestionar registros de rendimiento de supervisores en una empresa de transporte o similar. Este módulo permite a administradores y supervisores crear, consultar, actualizar y eliminar registros relacionados con el rendimiento de empleados, como cantidad de rutas, horas trabajadas y observaciones.

##### Roles de Usuario

- Administrador: Acceso completo a todas las operaciones.
- Supervisor: Acceso a creación y consulta de rendimientos.
- (Pasajero no aplica para este módulo)

##### Funcionalidades principales

Funcionalidad	Ruta	Método	Roles autorizados
Ver formulario de creación	/behavior/supervisor/crear	GET	Supervisor
Crear rendimiento	/behavior/create	POST	Administrador / Supervisor (comentado)
Ver formulario de actualización	/behavior/actualizar	GET	Administrador, Supervisor
Actualizar rendimiento	/behavior/update	POST	Administrador

Ver formulario de eliminación	/behavior/eliminar	GET	Administrador
Eliminar rendimiento	/behavior/delete	POST	Administrador
Consultar rendimientos (admin)	/behavior/consultar/administrador	GET	Administrador
Consultar rendimientos (supervisor)	/behavior/consultar/supervisor	GET	Supervisor
Listar todos los rendimientos	/behavior/rendimientos	GET	Administrador
Consultar por ID	/behavior/rendimiento?ID=1	GET	Administrador
Consultar por ID de usuario	/behavior/user?iduser=123	GET	Administrador, Supervisor

## FASE 2: Diseño técnico e interfaces

Estructura de datos

Modelo: BehaviorOut

```
class BehaviorOut(BaseModel):
    ID: int
    iduser: int
    cantidadrutas: int
    horastrabajadas: int
    observaciones: str
    fecha: str
```

Modelo: BehaviorCreate (mismo que el anterior, pero usado para inserción)

## Lógica

El servicio utiliza el UniversalController para conectarse con la base de datos SQL Server.

Opera con funciones como:

- read\_all()
- get\_by\_id()
- get\_by\_column()
- add()
- update()
- delete()

## Plantillas HTML usadas

- CrearSupervisorRendimiento.html
- ActualizarRendimiento.html
- EliminarRendimiento.html
- ConsultarRendimientoViaAdministrador.html
- ConsultarRendimientoViaSupervisor.html
- rendimiento.html
- rendimientos.html
- Supervisorbehaviors.html

## FASE 3: Implementación del servicio

### Crear rendimiento (POST /create)

Valida si el ID ya existe. Si no existe, crea el rendimiento. Registra logs e informa errores específicos o internos.

### Actualizar rendimiento (POST /update)

Busca por ID. Si no encuentra el registro, devuelve 404. Si encuentra, actualiza los campos.

### Eliminar rendimiento (POST /delete)

Valida existencia antes de eliminar. Devuelve 404 si no lo encuentra.

## Consultas

- GET /rendimientos: retorna JSON de todos los registros.
- GET /rendimiento?ID=...: retorna HTML con datos de un rendimiento.
- GET /user?iduser=...: consulta por usuario.
- GET /supervisor/behaviors: HTML con lista completa (vista supervisor).

#### FASE 4: Seguridad y Middleware

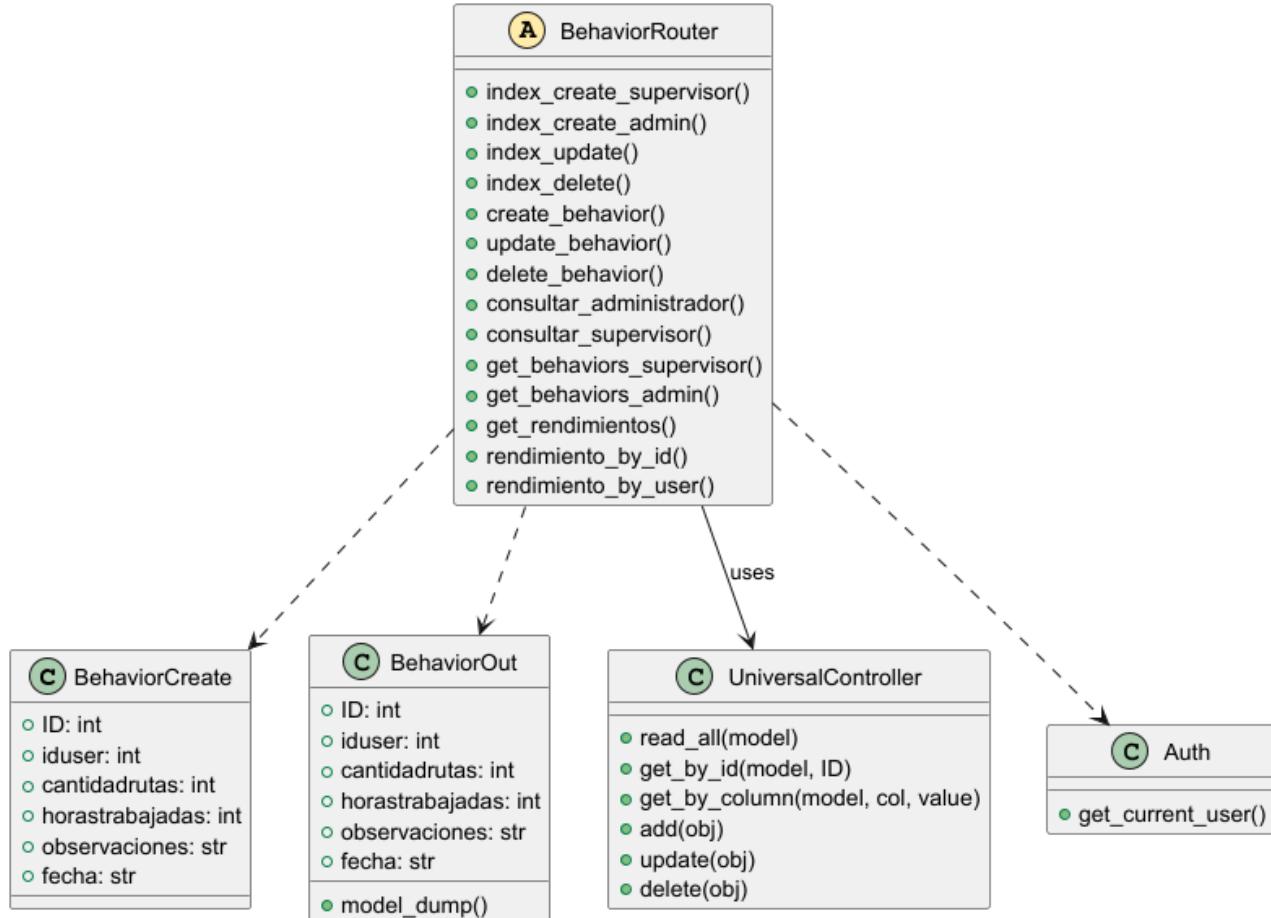
##### Autenticación y Autorización

- Se usa JWT con scopes. Algunas rutas tienen autenticación comentada (probablemente en desarrollo o desactivada para pruebas). Por ejemplo:  
`current_user: dict = Security(get_current_user, scopes=["system", "administrador"])`

##### Manejo de errores

- Usa HTTPException con status 400, 404, y 500.
- Logueo detallado para trazabilidad.

## Diagrama Estructural



### 3.7.21. Microservicio: Asistencia

#### FASE 1: Análisis y Diseño Funcional

##### Objetivo del servicio:

Gestionar los registros de asistencia de los usuarios del sistema, permitiendo su creación, actualización, eliminación y consulta.

##### Actores principales:

- Administrador
  
- Supervisor

- Técnico
- Conductor
- Sistema (automatismos internos)

Casos de uso funcionales:

- Crear una nueva asistencia
- Actualizar una asistencia existente
- Eliminar una asistencia
- Consultar asistencias (todas, por ID o por usuario)
- Visualizar formularios HTML para cada operación según el rol del usuario

## **FASE 2. Diseño Técnico e Interfaces**

**Tecnologías empleadas:**

- FastAPI
- SQL Server (vía controlador universal)
- JWT para autenticación/autorización
- Jinja2 para renderizado HTML
- Logging nativo de Python

### Estructura de endpoints:

Endpoints de Formularios HTML (GET)

Ruta	Propósito	Roles permitidos
/asistencia/crear	Mostrar formulario para crear asistencia	system, administrador, supervisor, técnico, conductor
/asistencia/actualizar	Mostrar formulario para actualizar asistencia	system, administrador
/asistencia/eliminar	Mostrar formulario para eliminar asistencia	system, administrador
/asistencia/consultar	Mostrar página de consulta general	system, administrador, conductor, supervisor, mantenimiento
/asistencia/consultar/user	Página de consulta por usuario	system, administrador

Endpoints CRUD (POST)

Ruta	Propósito	Roles permitidos
/asistencia/create	Crear nueva asistencia	system, administrador
/asistencia/update	Actualizar asistencia existente	system, administrador
/asistencia/delete	Eliminar asistencia existente	system, administrador

Endpoints de Consulta (GET)

Ruta	Descripción	Roles permitidos
/asistencia/asistencias	Consultar todas las asistencias	system, administrador

/asistance/find?id={id}	Consultar asistencia por ID	system, administrador
/asistance/user?iduser={iduse r}	Consultar asistencias por ID de usuario	system, administrador, conductor, técnico, supervisor

### FASE 3. Implementación del Servicio

#### Modelo de datos (AsistanceCreate, AsistanceOut):

- id: Identificador único
- iduser: ID del usuario que marca asistencia
- horainicio: Hora de inicio de asistencia
- horafinal: Hora de finalización
- fecha: Fecha de la asistencia

#### Lógica general de controladores:

Se utiliza un UniversalController con funciones reutilizables para interactuar con la base de datos:

- get\_by\_id
- get\_by\_column
- read\_all
- add

- update
- delete

**Gestión de errores:**

- Validaciones de existencia previa para create
- Excepciones HTTP apropiadas (400, 404, 500)
- Mensajes de log para trazabilidad en cada paso del flujo

**Plantillas HTML:**

- CrearAsistencia.html
- ActualizarAsistencia.html
- EliminarAsistencia.html
- ConsultarAsistencia.html
- ConsultarAsistenciaUsuario.html
- asistencia.html
- asistencias.html

**FASE 4. Seguridad y Middleware**

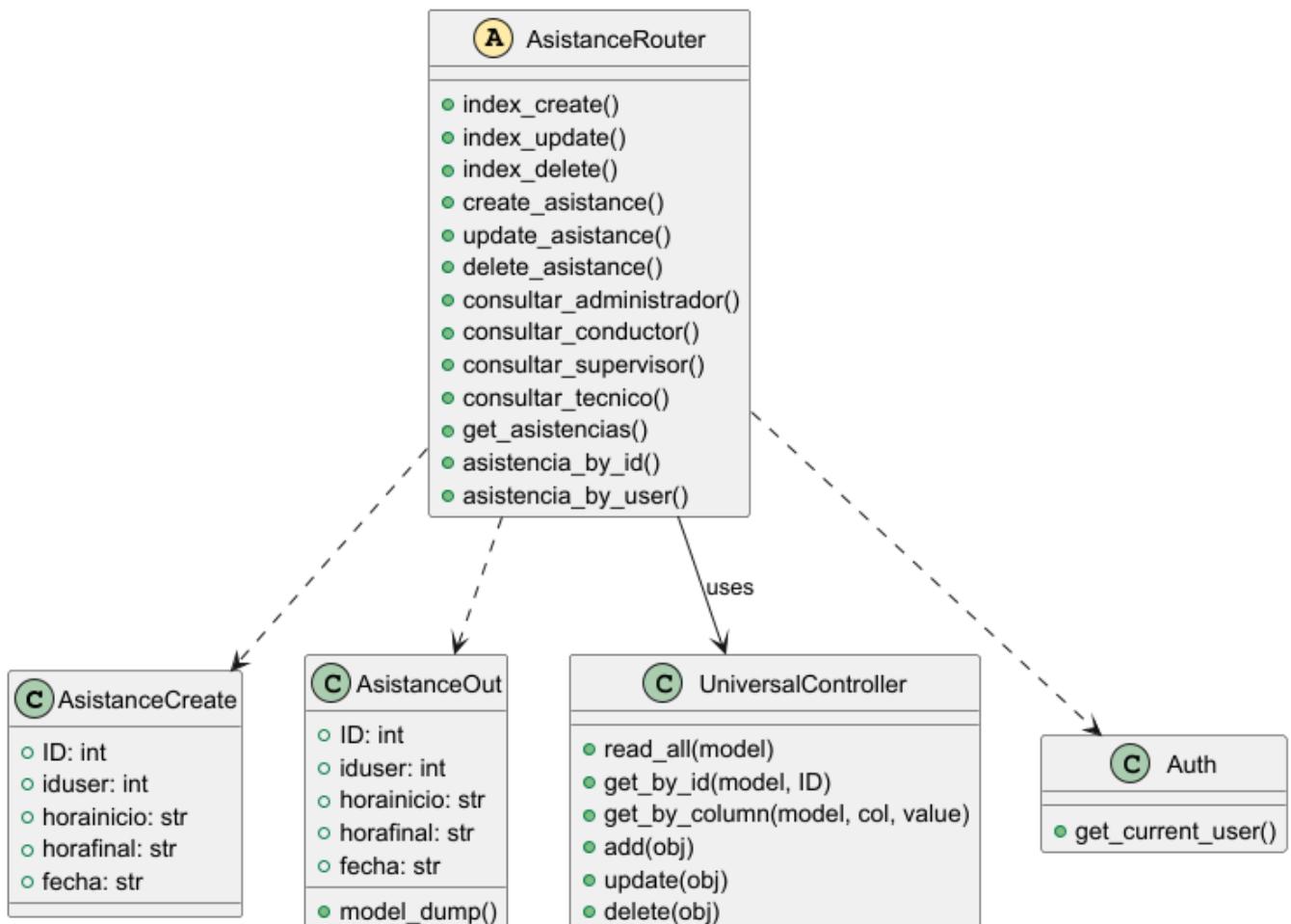
**Autenticación y Autorización:**

- Se implementa seguridad mediante JWT y Security(get\_current\_user, scopes=[...])
- Cada endpoint define los scopes requeridos según el tipo de operación y nivel de acceso

### Logging:

- Se emplea logging para registrar accesos, operaciones y errores
- Diferencia entre logs informativos (info), advertencias (warning) y errores (error)

### Diagrama Estructural



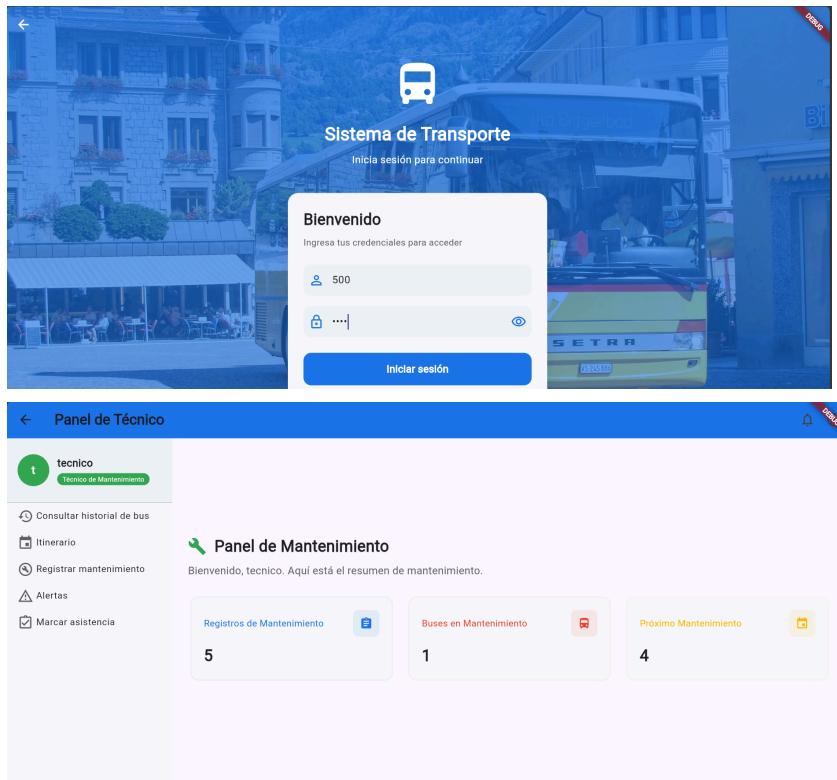
## 3.8 Prueba de Integración

Se elige el caso de uso del usuario: Técnico

### 3.8.1 Parámetros de Éxito

Parámetro	Resultado esperado
Inicio de sesión y asignación de cuenta	Funcional y permite
Dashboard general	Funcional y permite
Consultar Historial de Bus	Funcional y permite
Itinerario	Funcional y permite
Registrar Mantenimiento	Funcional y permite
Alertas	Funcional y permite
Marcar Asistencia	Funcional y permite

### 3.8.2 Imágenes del paso a paso



**Panel de Técnico**

**Alertas de Mantenimiento**

- Mantenimientos Atrasados**  
ID: 1 | Unidad: 1 | Desc: string | Fecha: 2025-05-14 | Técnico: 1
- Mantenimientos Próximos**  
ID: 32 | Unidad: 2 | Desc: Revision estandar | Fecha: 2025-06-12 | Técnico: 2

**Panel de Técnico**

**Agendar Mantenimiento**

**Formulario para Agendar Mantenimiento:**

ID	32
ID Status	2
Tipo de Mantenimiento	Revision estandar
Fecha (YYYY-MM-DD HH:MM:SS)	2025-06-12 07:00:00
ID Unidad	2

**Botón:** Agendar Mantenimiento

**Mensaje de Confirmación:** Mantenimiento agendado exitosamente.

**Panel de Técnico**

**Agendar Mantenimiento**

**Formulario para Agendar Mantenimiento:**

ID	
ID Status	
Tipo de Mantenimiento	
Fecha (YYYY-MM-DD HH:MM:SS)	
ID Unidad	

**Botón:** Agendar Mantenimiento

**Panel de Técnico**

**Unidad #1**  
Ubicación: Estación Central  
Capacidad: 50  
Ruta: 3  
Tipo: 2  
Horarios:  
De 15:00:00 a 16:00:00

**Unidad #2**  
Ubicación: Centro  
Capacidad: 25  
Ruta: 3  
Tipo: 1  
Horarios:  
De 15:00:00 a 16:00:00

**Unidad #3**

**Panel de Técnico**

**Consultar historial de bus**

ID de unidad de transporte:

**Unidad #2**

Ubicación: Centro  
Capacidad: 25  
Ruta: 3  
Tipo: 1

**Panel de Técnico**

**Consultar historial de bus**

ID de unidad de transporte:

**Crear Asistencia**

Complete los datos para crear una asistencia:

ID Asistencia:

ID Usuario:

Hora de Inicio (HH:MM:SS):

Hora Final (HH:MM:SS):

Fecha (YYYY-MM-DD):

## 3.9 Prueba de Seguridad

### 3.9.1. Introducción

Este documento presenta el análisis técnico detallado de las pruebas de seguridad realizadas con OWASP ZAP v2.16.1 sobre el sistema accesible en <http://app:8000>. El objetivo es facilitar al equipo de desarrollo y seguridad la identificación de vulnerabilidades, la evidencia asociada, el contexto de cada hallazgo y las acciones recomendadas para su remediación.

### 3.9.2. Datos Generales del Escaneo

- Sitio escaneado: <http://app:8000>
- Fecha del reporte: 3 de junio de 2025, 13:47:08 UTC
- Herramienta: ZAP by [Checkmarx](#)
- Versión de ZAP: 2.16.1

### 3.9.3. Resumen Ejecutivo

Nivel de Riesgo	Nº de Alertas
Alta (High)	0
Media (Medium)	0
Baja (Low)	3
Informativa	3
Falsos positivos	0

No se detectaron vulnerabilidades de riesgo alto o medio. Sin embargo, se identificaron varias áreas de mejora en cuanto a riesgos bajos y recomendaciones informativas.

### 3.9.4. Detalle de Hallazgos

#### 3.9.4.1. Alertas de Riesgo Bajo

##### 3.9.4.1.1. Insufficient Site Isolation Against Spectre Vulnerability

- **Descripción:**

Falta la cabecera Cross-Origin-Resource-Policy en las respuestas HTTP, lo que puede permitir ataques de canal lateral como Spectre.

- **URLs afectadas (Método GET):**

- /incidences/
- /maintainance/maintenance/token\_info?token\_info=
- /maintenance\_status/
- /shifts/
- /shifts/turnos
- /tickets/
- /user/actualizar
- /user/crear
- /user/eliminar

- **Evidencia:**

Las respuestas de estas rutas no incluyen la cabecera Cross-Origin-Resource-Policy.

- **Recomendación técnica:**

Incluir la cabecera Cross-Origin-Resource-Policy con el valor same-origin en todas las respuestas HTTP, salvo que se requiera compartir recursos entre dominios, en cuyo caso utilizar cross-origin.

- **Referencia:**

[MDN: Cross-Origin Resource Policy](#)

- **CWE: 693**

- **WASC: 14**

### *3.9.4.1.2. Timestamp Disclosure - Unix*

- **Descripción:**

El servidor expone timestamps en formato Unix en las respuestas, lo que puede permitir a un atacante inferir información sobre la lógica interna, actividad o historial del sistema.

- **URL afectada (GET):**

- /movement/administrador/movements

- **Evidencia:**

Ejemplo de valores encontrados:

- 1486480508 (2017-02-07 15:15:08)
- 1679225126 (2023-03-19 11:25:26)
- 1947832306 (2031-09-22 08:31:46)

- **Recomendación técnica:**

No exponer valores de timestamp en las respuestas a menos que sea estrictamente necesario. Si deben exponerse, considerar formatear la fecha y limitar el contexto entregado al usuario.

- **CWE: [200](#)**

- **WASC: 13**

#### ***3.9.4.1.3. X-Content-Type-Options Header Missing***

- **Descripción:**

Ausencia de la cabecera X-Content-Type-Options: nosniff en las respuestas HTTP, lo que puede permitir ataques de tipo MIME sniffing.

- **URLs afectadas (GET):** Coincidén con las de la alerta 4.1.1.

- **Recomendación técnica:**

Añadir la cabecera X-Content-Type-Options: nosniff en todas las respuestas HTTP.

- **Referencia:**

[OWASP Secure Headers](#)

#### **3.9.4.2. Alertas Informativas**

##### ***3.9.4.2.1. Authentication Request Identified***

- **Descripción:**

Se identificó la presencia de una solicitud de autenticación. Aunque no es una vulnerabilidad, se recomienda asegurar estos endpoints y monitorear su uso.

#### ***3.9.4.2.2. Information Disclosure - Sensitive Information in URL***

- **Descripción:**

Se detectó información potencialmente sensible transmitida en los parámetros de la URL. Esto puede quedar registrado en logs, navegadores y proxies intermedios.

- **Instancias:** 4

- **Recomendación técnica:**

Evitar enviar información sensible por la URL. Utilizar el cuerpo de la petición (POST) o cifrar los parámetros cuando sea posible.

#### ***3.9.4.2.3. Non-Storable Content***

- **Descripción:**

Determinadas respuestas HTTP no son almacenables en caché por el navegador. Esto no es necesariamente un riesgo de seguridad, pero puede afectar el rendimiento o la experiencia de usuario.

- **Instancias:** 9

### **3.9.5. Evidencia Técnica**

#### ***3.9.5.1. Ejemplo de Respuesta sin CORP***

##### **Request:**

- GET /incidences/ HTTP/1.1
- Host: app:8000

**Response sin cabecera CORP:**

HTTP/1.1 200 OK  
Content-Type: application/json  
...  
[Sin Cross-Origin-Resource-Policy]

**3.9.5.2. Ejemplo de Timestamp Disclosure****Response:**

```
{  
  "created_at": 1486480508,  
  "updated_at": 1679225126  
}
```

**3.9.5.3. Ejemplo de Falta de X-Content-Type-Options****Response headers:**

HTTP/1.1 200 OK  
Content-Type: application/json  
[Sin X-Content-Type-Options]

**3.9.6. Recomendaciones Técnicas Generales****1. Seguridad en cabeceras HTTP:**

- Añadir            Cross-Origin-Resource-Policy: same-origin            y  
                      X-Content-Type-Options: nosniff a las respuestas.

**2. Gestión de información sensible:**

- No exponer timestamps Unix ni información sensible en URLs.

- Auditar logs y endpoints que puedan registrar o mostrar información confidencial.

### 3. Revisión de endpoints de autenticación:

- Fortalecer la seguridad y monitoreo en rutas de autenticación.

### 4. Caché y almacenamiento:

- Revisar políticas de almacenamiento en caché si se identifican problemas de rendimiento.

### 5. Repetir escaneos periódicamente:

- Realizar pruebas de seguridad de forma rutinaria tras aplicar remediaciones o cambios en la aplicación.

## 3.9.7. Referencias

- [OWASP ZAP Documentation](#)
- [Cross-Origin Resource Policy – MDN](#)
- [OWASP Secure Headers](#)
- [CWE-693: Protection Mechanism Failure](#)
- [CWE-200: Information Exposure](#)

## 3.9.8. Conclusión

El escaneo realizado no detectó vulnerabilidades críticas, pero sí varias áreas de mejora técnica relacionadas con cabeceras de seguridad y exposición de información. Se recomienda implementar las acciones correctivas descritas y mantener un ciclo de pruebas de seguridad constante para proteger la aplicación ante nuevas amenazas.

## 3.10 Prueba de Estrés

### Prueba en bruto:

Nombre del hilo: Thread Group 1-93  
Comienzo de muestra: 2025-05-27 23:01:10 COT  
Tiempo de carga: 77

Connect Time:0  
 Latencia:77  
 Tamaño en bytes:1164  
 Sent bytes:159  
 Headers size in bytes:219  
 Body size in bytes:945  
 Conteo de muestra:1  
 Conteo de error:0  
 Data type ("text"|"bin"|""):text  
 Código de respuesta:200  
 Mensaje de respuesta:OK

HTTPSampleResult campos:  
 ContentType: text/html; charset=utf-8  
 DataEncoding: utf-8

### Prueba en paseado:

Nombre del hilo	Thread Group 1-93
Comienzo de muestra	2025-05-27 23:01:10 COT
Tiempo de carga	77
Connect Time	0
Latencia	77
Tamaño en bytes	1164
Sent bytes	159
Headers size in bytes	219
Body size in bytes	945
Conteo de muestra	1
Conteo de error	0
Código de respuesta	200
Mensaje de respuesta	OK

Cabecera de respuesta	Valor
HTTP/1.1 200 OK	
Content-Length	945
Content-Type	text/html; charset=utf-8
Date	Wed, 28 May 2025 04:01:10 GMT
Server	railway-edge
X-Railway-Edge	railway/us-east4
X-Railway-Request-Id	7TcVAxVHTc67_0xMO8K9hQ

### Reporte general

Etiqueta	# Muestras	Media	Mín	Máx	Desv. Estándar	% Error	Rendimiento	Kb/sec	Sent KB/sec	Media de Bytes
HTTP Request	3000	109	67	1800	127,85	0,00%	387,3/sec	440,30	60,14	1164,0
Total	3000	109	67	1800	127,85	0,00%	387,3/sec	440,30	60,14	1164,0

## Conclusiones

Con base en los resultados obtenidos en las pruebas de estrés realizadas a través de JMeter, se puede concluir que el sistema evaluado presenta un desempeño altamente eficiente y estable bajo carga. En la ejecución masiva, se enviaron 3000 solicitudes HTTP, obteniendo una media de tiempo de respuesta de solo 109 ms, con un mínimo de 67 ms y un máximo de 1800 ms. La desviación estándar de 127,85 ms indica una moderada dispersión, lo cual es aceptable en pruebas de alto volumen. Lo más destacable es que no se registraron errores (0,00%), evidenciando que el servidor es capaz de mantener la integridad operativa incluso en condiciones exigentes.

El rendimiento fue constante, alcanzando un promedio de 387,3 solicitudes por segundo, con una tasa de transferencia de 440,30 KB/s, lo cual representa un volumen de datos manejado eficientemente, incluso con un tamaño medio de respuesta de 1164 bytes por petición. Estos valores confirman que el sistema no solo es rápido, sino también capaz de escalar adecuadamente frente a un número creciente de usuarios simultáneos.

Complementando esta evaluación, se realizó una prueba individual con el hilo número 93, la cual presentó un tiempo de respuesta de 77 ms, sin errores, y con un tamaño de respuesta también de 1164 bytes. Este resultado, en línea con el comportamiento observado en las pruebas masivas, confirma la consistencia y solidez del sistema en distintas condiciones de carga. La ausencia de errores, el tiempo de conexión nulo (posiblemente por el uso de conexiones persistentes), y la baja latencia reflejan una arquitectura de red y aplicación bien optimizada para servicios concurrentes.

## 4. Otros requisitos No funcionales

### 4.1 Requerimientos de desempeño

**REQ-1:** El sistema debe tener un tiempo máximo por petición en API de 5000 ms

**REQ-2:** El sistema debe estar disponible las 24 horas del día, los 7 días de la semana, con un mínimo de tiempo de inactividad.

**REQ-3:** El sistema debe ser capaz de manejar, 100000 transacciones por hora

**REQ-4:** El sistema debe optimizar su flujo de datos para minimizar el uso innecesario de memoria.

### 4.2 Requisitos de seguridad

**REQ-5:** El sistema debe tener un sistema de comprobación y verificación de cambios.

**REQ-6:** El sistema debe asegurar los datos y su recorrido hacia las APIs a través de gateways

**REQ-7:** El sistema debe encriptar y desencriptar los datos sensibles.

**REQ-8:** El sistema debe autenticar y autorizar la identidad del usuario.

#### **4.3 Atributos de calidad del software**

***REQ-9:*** *El sistema debe contar con una, guía intuitiva para el usuario*

***REQ-10:*** *El sistema debe tener un tiempo de aprendizaje para nuevos usuarios que no debe exceder los 30 minutos.*

***REQ-11:*** *El sistema debe estar abierto a actualizaciones.*

***REQ-12:*** *El sistema debe estar disponible 24/7*

***REQ-13:*** *El sistema debe ser gestionado cada 2 días*

## 5. Requerimientos de Sistemas

### 5.1 Arquitectura orientada a Servicios

- **REQS-1:** *El sistema debe estar desarrollado bajo una infraestructura basada en la nube.*
- **REQS-2:** *El sistema debe estar desarrollado bajo la integración de diversas APIs autónomas.*

### 5.2 Lenguajes de Programación

- **REQS-3:** *El sistema deberá estar desarrollado en los lenguajes Python, Java y JavaScript.*
- **REQS-4:** *Todo el código debe seguir estándares de codificación específicos para Python (PEP 8), Java (Java Code Conventions) y JavaScript (Airbnb Style Guide o similar).*
- **REQS-5:** *Se deben implementar pruebas unitarias para cada componente de código, aprovechando frameworks como PyTest, JUnit o Jest según corresponda.*

### 5.3 Base de Datos

- **REQS-6:** *El sistema deberá utilizar una base de datos en Microsoft Azure SQL.*
- **REQS-7:** *La base de datos debe tener configuradas políticas de respaldo automático y puntos de recuperación (restore points) en Microsoft Azure SQL.*

### 5.4 Repositorio

- **REQS-8:** *El sistema, especialmente toda la implementación, deberá ser almacenado en un repositorio de GitHub.*
- **REQS-9:** *Se debe integrar el repositorio con herramientas de integración continua/entrega continua (CI/CD) como GitHub Actions para automatizar despliegues.*

## 6. Sprints

### 6.1. Sprint #1

#### 6.1.1. Backlog

**Objetivo del Sprint:** Definir el diseño general del MVP, detallando los requerimientos y estructura del sistema.

**Duración:** 1 semana.

ID	Historia de Usuario	Tareas	Prioridad
1	Como equipo de desarrollo, se espera un diagrama de contexto del sistema detallado para la planificación del MVP.	<ul style="list-style-type: none"> <li>-Crear un diagrama de contexto del sistema.</li> <li>- Identificar los principales actores y sistemas externos.</li> <li>- Validar el diagrama con el equipo.           <ul style="list-style-type: none"> <li>- Ajustar el diagrama según observaciones.</li> </ul> </li> </ul>	Alta
2	Como equipo de desarrollo, se realizará un diseño estructural (diagrama de clase)para analizar la interacción de los agentes dentro del sistema.	<ul style="list-style-type: none"> <li>-Definir estructura de bases de datos, objetos y componentes.</li> <li>-Identificar entidades principales.</li> <li>-Establecer relaciones entre objetos clave.</li> <li>-Definir restricciones y reglas de negocio.</li> </ul>	Alta
3	Como equipo de desarrollo, se creará un diagrama de caso de uso correspondiente a cada acción que se realizará dentro del sistema.	<ul style="list-style-type: none"> <li>-Diagramar escenarios de interacción del usuario con el sistema</li> <li>-Definir los principales flujos de usuario.</li> </ul>	Media
4	Como equipo de desarrollo, se necesita de un modelado basado en datos, con el fin de analizar el flujo de los datos dentro del sistema	<ul style="list-style-type: none"> <li>Diagramar el flujo de entrada, procesamiento y salida de datos en el sistema.</li> <li>-Modelado del flujo de datos.</li> <li>-Validación del modelo con el equipo</li> </ul>	Media

5	Como equipo de desarrollo, se necesita de un modelo Entity-Relationship (E-R), con el propósito de diseñar y normalizar la base de datos.	-Diagramar modelo E-R -Identificar entidades y atributos. -Normalizar las relaciones. -Diagramar el modelo relacional	Alta
---	---	--	------

### 6.1.2. Reporte Semanal

#### 6.1.2.1. Introducción

Sprint: 1

Fecha de Inicio: 17/03/2025

Fecha de Fin: 24/03/2025

Semana Cubierta: 17 de marzo - 24 de marzo

#### 6.1.2.2. Objetivos

1. Definir el diseño general del MVP
2. Detallar los requerimientos
3. Detallar la estructura del sistema.

#### 6.1.2.3. Progreso de las Historias de Usuario

---

NO.	ASSIGNED TO	PRIORITIES	PRIORITY LEVEL	PROGRESS	PROGRES S STATUS	PROBLEMS AND COMMENTS
1	Rubiano A.F	Levantamiento de requerimientos: Requerimientos de sistema	High	Temprano	Ready to Start	
2	Rubiano A.F	Diagrama de Contexto	High	Temprano	In Progress	
3	Julio M.A.	Modelo Entidad - Relación	High	En espera de No.5	On Hold	Comentará al equipo si necesita asistencia
4	Julio M.A.	Modelado Basado en datos	Medium	En espera de No.5	On Hold	Comentará al equipo si necesita

						asistencia
5	Steinman A.P	Diseño Estructural	High	Presenta desarrollo visible	In Progress	Debe realizar cambios al diseño
6	Steinman A.P	Diagrama de caso de uso	Medium	Presenta desarrollo visible	On Hold	Debe esperar la publicación del No. 2

22  
/03/  
**SAT 2025**

NO.	ASSIGNED TO	PRIORITIES	PRIORITY LEVEL	PROGRESS	PROGRES S STATUS	PROBLEMS AND COMMENTS
1	Rubiano A.F	Levantamiento de requerimientos: Requerimientos de sistema	High	Completado	Complete	
2	Rubiano A.F	Diagrama de Contexto	High	Completado	Complete	
3	Julio M.A.	Modelo Entidad - Relacion	High	Faltan entidades débiles, Atributos Fuertes y Relaciones	In Progress	
4	Julio M.A.	Modelado Basado en datos	Medium	Delegado al siguiente sprint	On Hold	No podrá terminarlo para este spring
5	Steinman A.P	Diseño Estructural	High	Completado	Complete	
6	Steinman A.P	Diagrama de caso de uso	Medium	Completado	Complete	

#### 6.1.2.4. Impedimentos y Obstáculos

- Julio M.A. requiere más tiempo para realizar el modelado basado en datos.

#### 6.1.2.5. Próximos Pasos

1. El equipo solicitará la creación de un repositorio integrado con SonarQube, Docker, Azure
2. Plantear reunión con el cliente
3. Diseño de características
4. Realizar el modelado basado en datos

#### 6.1.2.6. Archivos Vinculados

 Sprint # 1

Daily Meeting # 1: [Grabación de la reunión.mp4](#)

Deily Meeting # 2: [Grabación de la reunión.mp4](#)

### 6.2. Sprint #2

#### 6.2.1. Backlog

**Objetivo del Sprint:** Definir el diseño del Backend y FrontEnd, con el fin de presentar los primeros bocetos del diseño del sistema.

**Duración:** 3 semanas.

ID	Tareas	Descripción	Prioridad	Historia de Usuario
1	Diseño modelo relacional v.1.0	Diagrama el modelo relacional	Alta	Como equipo de desarrollo, se necesita de un modelo relacional, y la normalización de la base de datos
2	Normalizar el modelo relacional	Normalizar el modelo relacional	Media	
2	Diseño estructural v1.0	Crear diagrama de arquitectura, modelo de datos y documentar estructura.	Alta	Como arquitecto de software, quiero un diseño estructural documentado para asegurar que el sistema sea comprensible y mantenable.

3	Diseño de Mockup UI	Crear wireframes, definir estilo visual y obtener feedback.	Alta	Como usuario del sistema, quiero una interfaz clara y moderna para navegar fácilmente y realizar mis tareas sin confusión.
4	Creación estructura de proyecto en GitHub	Solicitar creacion de repositorio, definir estructura de carpetas y configurar CI/CD, Docker, SonarQube.	Alta	Como desarrollador, quiero una estructura organizada en GitHub para facilitar la colaboración y el versionado del código.
5	Definición de restricciones del Proyecto	Identificar limitaciones técnicas y de negocio, definir alcance y métricas de éxito.	Alta	Como gestor de proyectos, quiero conocer las restricciones para planificar mejor el desarrollo del producto.

### 6.2.2. Reporte Semanal

#### *Introducción*

Sprint: 2

Fecha de Inicio: 24/03/2025

Fecha de Fin: 31/03/2025

Semana Cubierta: 24 de marzo - 31 de marzo

#### *Objetivos*

1. Diseño de Mockup UI
2. Definición de restricciones del Proyecto
3. Diseño modelo relacional v.1.0
4. Modelado basado en datos
5. Normalizar el modelo relacional
6. Diseño estructural v1.0
7. Creación estructura de proyecto en GitHub

**Progreso de las Historias de Usuario**

29 / 03 /

**WED 2025**

NO.	ASSIGNED TO	PRIORITIES	PRIORITY LEVEL	PROGRESS	PROGRESS STATUS	PROBLEMS AND COMMENTS
1	Rubiano A.F	Diseño de Mockup UI	High	Analisis e Investigacion	Ready to Start	
2	Rubiano A.F	Definición de restricciones del Proyecto	High	Analisis e Investigacion	Ready to Start	
3	Julio M.A.	Diseño modelo relacional v.1.0	High	Completa do	Complete	
4	Julio M.A.	Modelado basado en datos	High	Completa do	Complete	
5	Julio M.A.	Normalizar el modelo relacional	Medium	No iniciado	Ready to Start	No cree tener el tiempo para realizarlo
6	Julio M.A.	Diseño estructural v1.0	High	Analisis e Investigacion	Ready to Start	
7	Steinman A.P	Creación estructura de proyecto en GitHub	High	En progreso	In Progress	Pendiente a aceptacion del grupo

30 / 03 /

**SAT 2025**

NO.	ASSIGNED TO	PRIORITIES	PRIORITY LEVEL	PROGRESS	PROGRESS STATUS	PROBLEMS AND COMMENTS
1	Rubiano A.F	Diseño de Mockup UI	High	Se encuentra en desarrollo	In Progress	
2	Rubiano	Definición de	High	Completa	Complete	

	A.F	restricciones del Proyecto		da		
3	Julio M.A.	Diseño modelo relacional v.1.0	High	Completa da	Complete	
4	Julio M.A.	Modelado basado en datos	High	Completa da	Complete	
5	Julio M.A.	Normalizar el modelo relacional	Medium	No iniciada	Ready to Start	
6	Julio M.A.	Diseño estructural v1.0	High	Completa da	Complete	
7	Steinman A.P	Creación estructura de proyecto en GitHub	High	En progreso	In Progress	

### ***Impedimentos y Obstáculos***

Ninguno

### ***Próximos Pasos***

1. Integrar base de datos SQL al repositorio
2. Actualizar repositorio para considerar la utilización de flúter
3. Consignar reunión con el cliente.

### ***Archivos Vinculados***

Daily Meeting # 1: [Grabacion.mp4](#)

Daily Meeting # 2: [Grabacion.mp4](#)

## 6.3. Sprint #3

### 6.3.1. Backlog

**Objetivo del Sprint:** Definir la implementación de las clases y empezar a trabajar los diseños de nuestra base de datos principal.

**Duración:** 3 semanas.

ID	Tareas	Descripción	Prioridad	Historia de Usuario
1	Implementación de Clases	<ul style="list-style-type: none"> <li>- Creación e implementación de las siguientes clases en el repositorio: Pago, Incidencia, Turno, Horario, Reporte, GPS, Tarjeta, Parada, Notificación, TarjetaUsuario, TarjetaConductor, Mantenimiento, UnidadTransporte, Ruta, Usuario, UsuarioPasajero, Conductor, Administrador, SupervisorOperativo, TecnicoMantenimiento.</li> <li>- Creación e implementación de los “test” para las clases anteriormente mencionadas en el repositorio.</li> </ul>	Alta	Como equipo de desarrollo se espera una implementación sólida de las clases ya establecidas en nuestro diagrama, que se usarán en el backend de nuestro proyecto.
2	Diseño de Servicios CRUD por entidades.	<ul style="list-style-type: none"> <li>- Diseño de servicios tipo CRUD</li> <li>- Definición de operaciones CRUD debe realizar cada entidad.</li> <li>- Retroalimentación con el grupo, para comprobar la calidad del diseño.</li> </ul>	Alta	Como equipo de desarrollo, se espera un diseño CRUD sólido, que cumpla con los objetivos de mantenimiento y escalabilidad propuestos.

3	Implementación de un servicio CRUD, esencial Fast API.	<ul style="list-style-type: none"> <li>- Implementar correctamente el diseño CRUD previamente establecido</li> <li>- Implementar correctamente “FastAPI”, incluyendo esto sus endpoints en el repositorio establecido</li> </ul>	Alta	Como equipo de desarrollo, se debe garantizar rapidez y eficiencia en el ingreso y consulta de los datos, Siendo esto lo que se busca al implementar una herramienta como “fastAPI”.
4	Preparación de los ambientes para desarrollo.	<ul style="list-style-type: none"> <li>-Preparar los ambientes de desarrollos pertinentes, como Flutter, Python, Dart, Microsoft Azure, Docker.</li> <li>- Integrar y dejar operativo Azure SQL</li> </ul>	Media	Como equipo de desarrollo es vital que tengamos un espacio para empezar a desarrollar nuestro backend.

### 6.3.2. Reporte Semanal

#### *Introducción*

Sprint: 3

Fecha de Inicio: 31/03/2025

Fecha de Fin: 21/04/2025

Semana Cubierta: 31 de marzo - 21 de abril

#### *Objetivos*

Rediseño de Mockup UI

Definición de microservicios CRUD

Diseño de microservicios CRUD

implementación de microservicios CRUD

Diseño e implementación de clases (aspecto lógico)

#### *Progreso de las Historias de Usuario*

**THU 03 / 04****R / 2025**

NO.	ASSIGNED TO	PRIORITIES	PRIORITY LEVEL	PROGRESS	PROGRESS STATUS	PROBLEMS AND COMMENTS
1	Rubiano A.F	Implementación de Clases	High	En progreso	In Progress	
2	Rubiano A.F	Preparación de los ambientes para desarrollo.	High	En progreso	In Progress	
3	Julio M.A.	Implementación de Clases	High	En progreso	In Progress	
4	Julio M.A.	Diseño de Servicios CRUD por entidades	High	En progreso	In Progress	
5	Steinman A.P	Diseño de Servicios CRUD por entidades	High	En progreso	In Progress	
6	Steinman A.P	Implementación de Clases	High	Finalizado	Complete	Pendiente a aceptacion del grupo

**13/04/****SUN 2025**

NO.	ASSIGNED TO	PRIORITIES	PRIORITY LEVEL	PROGRESS	PROGRESS STATUS	PROBLEMS AND COMMENTS
1	Rubiano A.F	Diseño de Mockup UI	High	Se encuentra en desarollo	Complete	
2	Rubiano A.F	Definición de restricciones del Proyecto	High	Completada	Complete	
3	Julio M.A.	Diseño modelo relacional v.1.0	High	Completada	Complete	
4	Julio M.A.	Modelado basado en datos	High	Completada	Complete	
5	Julio M.A.	Normalizar el modelo relacional	Medium	No iniciada	Complete	
6	Julio M.A.	Diseño estructural v1.0	High	Completada	Complete	
7	Steinman A.P	Creación e Implementacion de Microservicios	High	En progreso	In Progress	
7	Julio M.A.	Creacion e Implementacion de Microservicios	High	En progreso	In Progress	

7	Rubiano A.F	Creacion e Implementacion de Microservicios	High	En progreso	In Progress	
---	-------------	---	------	-------------	-------------	--

**TUE** 15/04/

**S** 2025

NO.	ASSIGNED TO	PRIORITIES	PRIORITY LEVEL	PROGRESS	PROGRESS STATUS	PROBLEMS AND COMMENTS
1	Rubiano A.F	Diseño de Mockup UI	High	Se encuentra en desarrollo	Complete	
2	Rubiano A.F	Definición de restricciones del Proyecto	High	Completada	Complete	
3	Julio M.A.	Diseño modelo relacional v.1.0	High	Completada	Complete	
4	Julio M.A.	Modelado basado en datos	High	Completada	Complete	
5	Julio M.A.	Normalizar el modelo relacional	Medium	No iniciada	Complete	
6	Julio M.A.	Diseño estructural v1.0	High	Completada	Complete	
7	Steinman A.P	Creación e Implementacion de Microservicios	High	En progreso	Complete	
7	Julio M.A.	Creacion e Implementacion de Microservicios	High	En progreso	Complete	
7	Rubiano A.F	Creacion e Implementacion de Microservicios	High	En progreso	Complete	

19 / 04

**SAT** / 2025

NO.	ASSIGNED TO	PRIORITIES	PRIORITY LEVEL	PROGRESS	PROGRESS STATUS	PROBLEMS AND COMMENTS
1	Rubiano A.F	Diseño de Mockup UI	High	Se encuentra en desarrollo	Complete	
2	Rubiano A.F	Definición de restricciones del Proyecto	High	Completada	Complete	
3	Julio M.A.	Diseño modelo relacional v.1.0	High	Completada	Complete	
4	Julio M.A.	Modelado basado en datos	High	Completada	Complete	

5	Julio M.A.	Normalizar el modelo relacional	Medium	No iniciada	Complete	
6	Julio M.A.	Diseño estructural v1.0	High	Completada	Complete	
7	Steinman A.P	Creación e Implementacion de Microservicios	High	En progreso	Complete	
7	Julio M.A.	Creacion e Implementacion de Microservicios	High	En progreso	Complete	
7	Rubiano A.F	Creacion e Implementacion de Microservicios	High	En progreso	Complete	

20 / 04

**SUN / 2025**

NO.	ASSIGNED TO	PRIORITIES	PRIORITY LEVEL	PROGRESS	PROGRESS STATUS	PROBLEMS AND COMMENTS
1	Rubiano A.F	Diseño de Mockup UI	High	Completada	Complete	
2	Rubiano A.F	Definición de restricciones del Proyecto	High	Completada	Complete	
3	Julio M.A.	Diseño modelo relacional v.1.0	High	Completada	Complete	
4	Julio M.A.	Modelado basado en datos	High	Completada	Complete	
5	Julio M.A.	Normalizar el modelo relacional	Medium	Completada	Complete	
6	Julio M.A.	Diseño estructural v1.0	High	Completada	Complete	
7	Steinman A.P	Creación e Implementacion de Microservicios	High	Completada	Complete	
8	Julio M.A.	Creacion e Implementacion de Microservicios	High	Completada	Complete	
9	Rubiano A.F	Creacion e Implementacion de Microservicios	High	Completada	Complete	

***Impedimentos y Obstáculos***

Ninguno

***Próximos Pasos***

1. Integrar base de datos SQL al repositorio
2. Actualizar repositorio para considerar la utilización de flúter
3. Consignar reunión con el cliente.

***Archivos Vinculados***

Daily Meeting # 1: [Daily #1](#)

Daily Meeting # 2: [Grabación.mp4](#)

Daily Meeting # 3: [Grabación.mp4](#)

Daily Meeting # 4: [Grabación.mp4](#)

Daily Meeting # 5: [Grabacion.mp4](#)

## 6.4. Sprint #4

### 6.4.1. Backlog

**Objetivo del Sprint:** Corregir el mockup acorde al MVP

**Duración:** 1 semana.

ID	Historia de Usuario	Tareas	Responsable(s)	Prioridad	Notas
1	Como desarrollador, quiero que los microservicios CRUD estén correctamente implementados para asegurar su funcionalidad completa.	Corregir errores existentes. Implementar correctamente cada operación CRUD. Realizar pruebas de funcionalidad.	Todos	Alta	Cada microservicio debe ser 100% funcional y será testeado.
2	Como desarrollador, necesito integrar una base de datos SQL viable para garantizar persistencia y consistencia de datos.	Investigar opciones SQL viables. Seleccionar una base de datos adecuada. Integrarla con los microservicios.	Andres Rubiano	Media	Asegurar compatibilidad y rendimiento.
3	Como desarrollador, quiero que los microservicios interactúen correctamente con la base de datos seleccionada.	Establecer conexión con la BD. Configurar queries y operaciones. Validar la integración.	Todos	Alta	Debe probarse con datos reales.

4	Como diseñador, necesito verificar el mockup y alineararlo con el MVP para evitar inconsistencias visuales o funcionales.	Comparar mockup vs MVP. Detectar y documentar diferencias. Corregir el mockup.	Alejandro Steinman	Alta	Feedback al grupo en caso de errores.
5	Como equipo, necesitamos implementar la Fase 4 del sistema con todos los microservicios definidos.	Desarrollar las funcionalidades asignadas a Fase 4. Integrar microservicios. Probar la integración.	Todos	Media	Seguir especificaciones del roadmap.
6	Como equipo, necesitamos implementar la Fase 5 del sistema.	Implementar Fase 5 según diseño. Validar funcionamiento.	Todos	Media	
7	Como desarrollador, necesito implementar correctamente los test de las entidades para garantizar su estabilidad.	Crear test para cada entidad. Ejecutar y validar sin errores.	Todos	Alta	Usar librerías de testing estándar.
8	Como desarrollador, quiero testear cada microservicio para asegurar que su comportamiento es el esperado.	Crear test unitarios por servicio. Validar cada uno sin errores.	Todos	Alta	

9	Como cliente, quiero tener los nombres de la documentación SRS y weekly report estandarizada según mis parámetros.	Estandarizar el nombre de SRS Estandarizar el nombre de weekly report	Alejandro Steinman Mario Julio	Baja	Actualizacion de SRS (Nombre_Proyecto SRS v3)  Informes de los Weekly_report 1, 2 y 3
10	Como cliente, quiero una documentación clara y definida para el código desarrollado y los servicios	Documentar código Documentar servicios	Todos	Alta	
11	Como cliente, quiero una documentación clara y definida para la api	Documentar api	Todos	Alta	
12	Como desarrollador, quiero la implementación CI en github, coverage funcional	Implementar CI github Implementar coverage	Alejandro Steinman	Alta	

13	Como desarrollador, debe reducirse la duplicidad a menor del 3%	Optimizar html	Mario Julio	Media	
14	Como desarrollador, debe implementarse cambios de implementación a los servicios separando la api del servicio y implementando basemodel	-separar logica del servicio de la api - mover cada archivo a su directorio -cambiar lógica por BaseModel -hacer test de BaseModel	Todos	Media	

#### 6.4.2. Reporte Semanal

---

24 / 04

THU / 2025

NO.	ASSIGNED TO	PRIORITIES	PRIORITY LEVEL	PROGRESS	PROGRESS STATUS	PROBLEMS AND COMMENTS
1	Rubiano A.F  Julio M. A  Steinman A.P	Implementación Crud.	High	En progreso	In progress	
2	Rubiano A.F	Integración de base de datos SQL	High	En progreso	In progress	

	Rubiano A.F					
3	Julio M. A Steinman A.P	Correcta interacción de la base de datos con los microservicios	High	En progreso	In progress	
4	Steinman A.P	Analisis y verificacion de MockUp	High	Completado	Completed	
5	Rubiano A.F Julio M. A Steinman A.P	Integración de los microservicios.	Medium	En progreso	In progress	
6	Rubiano A.F Julio M. A Steinman A.P	Implementación de la fase 5	Medium	En progreso	In progress	
7	Rubiano A.F Julio M. A Steinman A.P	Corrección e implementación de los test	High	En progreso	In progress	
8	Rubiano A.F Julio M. A Steinman A.P	Testeo de los microservicios	High	En progreso	In progress	
9	Steinman A.P	Estandarización de los nombres de el SRS y los weekly reports	Medium	Completado	Completed	
10	Rubiano A.F Julio M. A	Documentación del código	High	En progreso	In progress	

	Steinman A.P					
11	Rubiano A.F					
	Julio M. A	Documentación de la API	High	En progreso	In progress	
	Steinman A.P					
12	Steinman A.P	Implementación del CL y el coverage	High	En progreso	In progress	
13	Julio M.A	Optimización del HTML para realizar CRUD	Medium	En progreso	In progress	
14	Rubiano A.F					
	Julio M. A	Creación de DTOs	High	En progreso	In progress	
	Steinman A.P					

26 / 04

SAT / 2025

NO.	ASSIGNED TO	PRIORITIES	PRIORITY LEVEL	PROGRESS	PROGRESS STATUS	PROBLEMS AND COMMENTS
1	Rubiano A.F					
	Julio M. A	Implementación Crud.	High	Completado	Completed	
	Steinman A.P					
2	Rubiano A.F	Integración de base de datos SQL	High	En progreso	In progress	
3	Rubiano A.F					
	Julio M. A	Correcta interacción de la base de datos con los microservicios	High	Completado	Completed	
	Steinman A.P					

4	Steinman A.P	Analisis y verificacion de MockUp	High	Completado	Completed	
5	Rubiano A.F Julio M. A Steinman A.P	Integración de los microservicios.	Medium	Completado	Completed	
6	Rubiano A.F Julio M. A Steinman A.P	Implementación de la fase 5	Medium	En progreso	In progress	
7	Rubiano A.F Julio M. A Steinman A.P	Corrección e implementación de los test	High	Completado	Completed	
8	Rubiano A.F Julio M. A Steinman A.P	Testeo de los microservicios	High	Completado	Completed	
9	Steinman A.P	Estandarización de los nombres de el SRS y los weekly reports	Medium	Completado	Completed	
10	Rubiano A.F Julio M. A Steinman A.P	Documentación del código	High	Completado	Completed	Documentación: <a href="#">PDF Service Docu...</a>
11	Rubiano A.F Julio M. A	Documentación de la API	High	Completado	Completed	Documentación: <a href="#">PublicTransitAgency/docs/openapi.json at main · ISCODEVUTB/PublicT</a>

	Steinman A.P					<a href="#">transitAgency</a>
12	Steinman A.P	Implementación del CL y el coverage	High	En progreso	In progress	
13	Julio M.A	Optimización del HTML para realizar CRUD	Medium	En progreso	In progress	
14	Rubiano A.F					
	Julio M. A	Creación de DTOs	High	Completado	Completed	
	Steinman A.P					

### ***Impedimentos y Obstáculos***

Ninguno

### ***Próximos Pasos***

1. Completar las tareas que quedaron pendientes.
2. Actualizar repositorio para considerar la utilización de flutter.
3. Consignar reunión con el cliente.

### ***Archivos Vinculados***

Daily Meeting # 1: [Daily #1](#)

Daily Meeting # 2: [Daily #2](#)

## **6.5. Sprint #5**

### **6.5.1. Backlog**

**Objetivo del Sprint:** Optimizar y validar la operatividad de los servicios

**Duración:** 1 semana.

ID	Historia de Usuario	Tareas	Responsable(s)	Prioridad	Notas
	Como desarrollador, los servicios crud deben estar implementados y funcionales	-implementar microservicios funcionales -comentar los servicios -utilizar estándar PEP8	Todos	Alta	Cada microservicio debe ser 100% funcional y será testeado.
	Como desarrollador, todos los servicios deben contar con pruebas unitarias con 100% de passing	-Implementar pruebas unitarias a cada servicio	Todos	Alta	
	Como desarrollador, debe implementarse el diseño estructural por cada servicio	-Desarrollar diseño estructural por cada servicio	Todos	Alta	
	Como desarrollador, debe implementarse pruebas de integración para los servicios	-implementarse pruebas de integracion para los servicios -asegurar total funcionalidad	Todos	Alta	

	Como desarrollador, quiero la implementación del coverage funcional	Implementar coverage	Alejandro Steinman	Alta	
	Como desarrollador, el cálculo del coverage debe tener un porcentaje del 80%	Validar creación de test que cumpla con el requerimiento porcentual	Todos	Alta	
	Como desarrollador, necesito integrar una base de datos SQL viable para garantizar persistencia y consistencia de datos.	Integrar una base de datos SQL	Andres Rubiano	Media	Asegurar compatibilidad , rendimiento y disponibilidad a largo plazo.
	Como desarrollador, quiero que los microservicios interactúen correctamente con la base de datos seleccionada.	Establecer conexión con la BD. Configurar queries y operaciones. Validar la integración.	Todos	Alta	Debe probarse con datos reales.
	Como equipo, necesitamos implementar la Fase 4 del sistema con todos los microservicios definidos.	Desarrollar las funcionalidades asignadas a Fase 4. Integrar microservicios. Probar la integración.	Todos	Alta	Seguir especificaciones del roadmap.

	Como equipo, necesitamos implementar la Fase 5 del sistema.	Implementar Fase 5 según diseño. Validar funcionamiento .	Todos	Alta	Seguir especificaciones del roadmap.
	Como cliente, quiero una documentación clara y definida para el código desarrollado y los servicios	Documentar código Documentar servicios	Todos	Media	
	Como cliente, quiero una documentación clara y definida para la api	Documentar api	A.steinman (todos)	Media	
	Como desarrollador, debe reducirse la duplicidad a menor del 3%	Optimizar html	Mario Julio	Media	
	Como equipo, debemos corregir errores asociados al mockup	-Implementar tabla QuejaSugerencia y entidad CRUD QuejaSugerencia  -Implementar tabla Asistencia y entidad CRUD Asistencia	A.Rubiano (los 6 primeros) todos (ultimo punto)		

		<p>-trigger/script desempeño de conductores.</p> <p>Implementación de triggers para la asignación automática de horarios</p> <p>-</p> <p>Implementación de tabla para el desempeño, y entidad para el desempeño o función para generar un reporte de desempeño</p> <p>-preguntar a mauricio sobre trabajar con transporter</p> <p>-Investigar viajes multimodal en la aplicación.</p>			
	Como desarrolladores, debemos asegurar clean code en el repositorio	-basado en el análisis de sonar cloud, implementar correcciones	Todos		

		respecto al código			
--	--	--------------------	--	--	--

### 6.5.2. Reporte Semanal

1 / 05 /

THU 2025

NO.	ASSIGNED TO	PRIORITIES	PRIORITY LEVEL	PROGRESS	PROGRESS STATUS	PROBLEMS AND COMMENTS
1	Rubiano A.F	Implementación microservicios	High	En progreso	In progress	
	Julio M. A					
	Steinman A.P					
2	Rubiano A.F	Integración de base de datos SQL	High	En progreso	In progress	
3	Rubiano A.F	Correcta interacción de la base de datos con los microservicios	High	En progreso	In progress	
	Julio M. A					
	Steinman A.P					
4	Rubiano A.F	Integración de la fase 4.	Medium	En progreso	In progress	
	Julio M. A					
	Steinman A.P					
5	Rubiano A.F	Implementación de la fase 5	Medium	En progreso	In progress	
	Julio M. A					
	Steinman					

	A.P					
6	Rubiano A.F	Corrección e implementación de los test para los servicios	High	En progreso	In progress	
	Julio M. A					
	Steinman A.P					
7	Rubiano A.F	Testeo de los microservicios	High	En progreso	In progress	
	Julio M. A					
	Steinman A.P					
8	Rubiano A.F	Estandarización de la documentación	High	En progreso	In progress	
	Julio M. A					
	Steinman A.P					
9	Steinman A.P	Implementación del CL y el coverage	High	En progreso	In progress	
10	Rubiano A.F	Creación de los test para base models	High	En progreso	In progress	
	Julio M. A					
	Steinman A.P					
11	Rubiano A.F	Implementación de clean code	High	En progreso	In progress	
	Julio M. A					
	Steinman A.P					
12	Rubiano A.F	Creación de diagrama estructural	High	En progreso	In progress	
	Julio M. A					
	Steinman A.P					

13	Julio M. A	Optimizacion HTMLs	High	En progreso	Completed	
----	------------	--------------------	------	-------------	-----------	--

---

3 / 05 /

**SAT** 2025

NO.	ASSIGNED TO	PRIORITIES	PRIORITY LEVEL	PROGRESS	PROGRESS STATUS	PROBLEMS AND COMMENTS
1	Rubiano A.F Julio M. A Steinman A.P	Implementación microservicios	High	En progreso	Completed	
2	Rubiano A.F	Integración de base de datos SQL	High	En progreso	Completed	
3	Rubiano A.F Julio M. A Steinman A.P	Correcta interacción de la base de datos con los microservicios	High	En progreso	Completed	
4	Rubiano A.F Julio M. A Steinman A.P	Integración de la fase 4.	Medium	En progreso	Completed	
5	Rubiano A.F Julio M. A Steinman A.P	Implementación de la fase 5	Medium	En progreso	In progress	

	Rubiano A.F					
6	Julio M. A Steinman A.P	Corrección e implementación de los test para los servicios	High	En progreso	Completed	
7	Rubiano A.F Julio M. A Steinman A.P	Testeo de los microservicios	High	En progreso	In progress	
8	Rubiano A.F Julio M. A Steinman A.P	Estandarización de la documentación	High	En progreso	In progress	
9	Steinman A.P	Implementación del CL y el coverage	High	En progreso	In progress	
10	Rubiano A.F Julio M. A Steinman A.P	Creación de los test para base models	High	En progreso	Completed	
11	Rubiano A.F Julio M. A Steinman A.P	Implementación de clean code	High	En progreso	In progress	
12	Rubiano A.F Julio M. A Steinman A.P	Creación de diagrama estructural	High	En progreso	In progress	

ID	Responsable(s)	Tareas	Prioridad	Estado	Notas
13	Julio M. A	Optimizacion HTMLs	High	En progreso	Completed

### ***Impedimentos y Obstáculos***

Ningunos

### ***Próximos Pasos***

1. Completar las tareas que quedaron pendientes.
2. Actualizar repositorio para considerar la utilización de flutter.

### ***Archivos Vinculados***

Daily Meeting # 1: [Daily #1](#)

Daily Meeting # 2: [Daily #2](#)

## **6.6. Sprint #6**

### **6.6.1. Backlog**

**Objetivo del Sprint:** Optimizar y validar la operatividad de los servicios

**Duración:** 1 semana.

ID	Historia de Usuario	Tareas	Responsable(s)	Prioridad	Notas

1	Como desarrollador, todos los servicios deben estar optimizados, completados y correctamente testeados con controlador actualizado.	<ul style="list-style-type: none"> <li>-Testear microservicios funcionales</li> <li>-Comentar los servicios</li> <li>-Actualizar los microservicios con el controlador actualizado.</li> <li>-Utilizar estándar PEP8</li> </ul>	Todos	Alta	Cada microservicio debe ser 100% completo.
2	Como desarrollador, debe tener por lo menos un test de integración completo y aprobado, además de realizarse pruebas de estrés y pruebas de seguridad de todos los servicios.	<ul style="list-style-type: none"> <li>-Implementación de pruebas de integración de servicios.</li> <li>-A partir de la primera prueba, se recomienda realizar otras pruebas de integración con los otros servicios.</li> <li>-Realizar las pruebas de integración con POSTMAN</li> <li>-Investigar las maneras para realizar las pruebas de estrés y de seguridad de los servicios.</li> </ul>	Mario -> tecnico  Steinman-> supervisor  Rubiano -> pasajero	Alta	<p>La prueba de integración debe ser 100% aprobada y funcional.</p> <p>Al inicio del sprint, cada desarrollador deberá escoger un caso de uso para realizar la prueba de integración de servicios.</p> <p>Cada desarrollador deberá realizar por lo menos una prueba de integración de servicios.</p>

					Los resultados deben estar documentados en el SRS.
3	Como desarrollador, debe desplegarse los servicios.	-Investigar y desarrollar el despliegue de los servicios.	Julio M. Steinman A	Media - Alta	Algunos de los servicios pueden ser Azure con Kubernetes, o AWS.
4	Como desarrollador, todas las plantillas de frontend deben ser actualizadas y optimizadas.	-Los templates realizados deben revisarse y optimizarse.	Todos	Alta	
5	Como desarrollador, debe realizarse el diseño del home page e interfaz de usuario.	-Después del desarrollo de las pruebas de integración de los servicios.  -Debe investigar templates y demás recursos para un buen diseño de interfaz de usuario.	Steinman A	Media - Baja	

6	Como desarrollador, necesito integrar una base de datos SQL en AZURE SQL, viable para garantizar persistencia y consistencia de datos.	Integrar una base de datos SQL en AZURE.  Implementar funciones SQL y funcionales de caso de usuario..	Rubiano A	Media - Alta	Asegurar compatibilidad , rendimiento y disponibilidad a largo plazo.  Creación de procedures y triggers para operación de SQL.
7	Como equipo, necesitamos verificar y optimizar la implementación de la Fase 5 del sistema con todos los microservicios definidos.	Revisar y optimizar la fase 5 de todos los servicios.  Validar funcionamiento.	Steinman A	Alta	Seguir especificaciones del roadmap.
8	Como cliente, quiero una revisión de la documentación de SRS.	Documentar servicios.  Documentar tests de integración.  Corregir esquema de SRS para versión 0.7.	Steinman A	Media	
9	Como desarrolladores, debemos asegurar clean code en el repositorio	-Basado en el análisis de sonar cloud, implementar correcciones respecto al código.	Todos	Alta	

10	Negociar con el cliente, cambio del mvp	Comunicarse con el product manager, y negociar una nueva estructura.	Rubiano A	Alta	
----	---	--	-----------	------	--

### 6.6.2 Weekly Report

10 /

05 /

**SAT** 2025

NO.	ASSIGNE D TO	PRIORITIES	PRIORITY LEVEL	PROGRESS	PROGRES S STATUS	PROBLEMS AND COMMENTS
1	Julio M.W	Test de Integracion: Usuario: Tecnico	High	Investigaci on	Ready to Start	
2	Julio M.W	Despliege de los servicios	High	Desplegad o, debe comprobar lo	Testing	
3	Rubiano A.F	Test de Integracion: Usuario: Pasajero	High	No iniciado	Ready to Start	
4	Rubiano A.F	Integrar base de datos Azure SQL	High	falta coneccion	In Progress	
5	Rubiano A.F	Nogociacion con el cliente	High	Cita agendada	On Hold	
6	Rubiano A.F, Julio M.W, Steinman A.P	Implementacion, Optimizacion funcional, test: Microservicios con el nuevo controlador	High	Rubiano A.F: Hecho, Julio.W: debe corregir, Steinman: hecho	In Progress	
7	Rubiano A.F, Julio	Actualizar y optimizar plantillas frontend	Medium	Rubiano A.F: En	In Progress	

	M.W, Steinman A.P			progreso, Julio M: Hecho, Steinman: Progreso		
8	Rubiano A.F, Julio M.W, Steinman A.P	Implementar Clean Code	Medium	Rubiano A.F: En progreso, Julio M: en progreso, Steinman: Progreso	In Progress	
9	Steinman A.P	Test de Integracion: Usuario: Supervisor	High	En espera	On Hold	
10	Steinman A.P	Diseño home e interfaz de usuario	Low	Home completad o	In Progress	
11	Steinman A.P	Implementar Fase 5	High	No iniciado		
12	Steinman A.P	Revision de la documentacion del SRS	Medium	Hecho	Complete	

14 /  
**WE** 05 /  
**D** 2025

NO.	ASSIGNE D TO	PRIORITIES	PRIORITY LEVEL	PROGRESS	PROGRES S STATUS	PROBLEMS AND COMMENTS
1	Julio M.W	Test de Integracion: Usuario: Tecnico	High	Investigaci on	Ready to Start	
2	Julio M.W	Despliege de los servicios	High	Buscando nuevas alternativa s	In Progress	

3	Rubiano A.F	Test de Integracion: Usuario: Pasajero	High	Esperando despliege	On Hold	
4	Rubiano A.F	Integrar base de datos Azure SQL	High	faltan tigres y tablas	In Progress	
5	Rubiano A.F	Nogociacion con el cliente	High	Completa do	Complete	Se han aprobado
6	Rubiano A.F, Julio M.W, Steinman A.P	Implementacion, Optimizacion funcional, test: Microservicios con el nuevo controlador	High	Rubiano A.F: En progreso, Julio.W: Hecho, Steinman: hecho	In Progress	
7	Rubiano A.F, Julio M.W, Steinman A.P	Actualizar y optimizar plantillas frontend	Medium	Rubiano A.F: En progreso, Julio M: Hecho, Steinman: Progreso	In Progress	
8	Rubiano A.F, Julio M.W, Steinman A.P	Implementar Clean Code	Medium	Rubiano A.F: En progreso, Julio M: en progreso, Steinman: Progreso	In Progress	
9	Steinman A.P	Test de Integracion: Usuario: Supervisor	High	Esperando despliege	On Hold	
10	Steinman A.P	Diseño home e interfaz de usuario	Low	Home completad o	Complete	
11	Steinman A.P	Implementar Fase 5	High	Completa do	Complete	
12	Steinman A.P	Revision de la documentacion del SRS	Medium	Actualizaci on del srs con los	In Progress	

				cambios del cliente		
--	--	--	--	------------------------	--	--

### ***Impedimentos y Obstáculos***

Falta de opciones para despliegue

### ***Próximos Pasos***

- Integración con flutter
- Despliegue del microservicio

### ***Archivos Vinculados***

Daily Meeting # 1: [Daily #1](#)

Daily Meeting # 2: [Daily #2](#)

## **6.7. Sprint #7**

### **6.7.1. Backlog**

**Objetivo del Sprint:** Últimas mejoras, test e integración con el front-end

**Duración:** 1 semana.

ID	Historia de Usuario	Tareas	Responsable(s)	Prioridad	Notas
1	Como desarrollador, debe tener todos los test de integración completo y aprobado, además de realizarse pruebas de estrés y pruebas de seguridad de todos los servicios.	-Investigar las maneras para realizar las pruebas de estrés y de seguridad de los servicios.	Steinman A - Integración Julio M - Estrés Rubiano A-Seguridad	Alta	La prueba de integración debe ser 100% aprobada y funcional.  Los resultados deben estar documentados en el SRS.

2	Como desarrollador, debe desplegarse el frontend	-Investigar y desarrollar el despliegue de los servicios.  -Investigar y desarrollar despliegue de frontend.	Rubiano A	Media - Alta	Algunos de los servicios pueden ser Azure
3	Como desarrollador, todas las plantillas de flutter deben estar listas.	-Los widgets deben crearse, revisarse y optimizarse.	Todos	Alta	
4	Como cliente, quiero una revisión de la documentación de SRS.	Documentar servicios.  Documentar tests de integración, estrés y seguridad.  Corregir esquema de SRS para versión 0.8.	Steinman A	Media	
5	Como desarrollador, se debe realizar test de IU sobre los templates de flutter.	Realizar tests para la ejecución de widgets.	Julio M	Media - Baja	
6	Como cliente, quiero una revisión de README	Actualizar README con base a las características del repositorio.	Julio M	Media - Alta	

7	Como desarrollador, quiero una optimización de servicios existentes.	Realizar mantenimiento y optimización de servicios existentes.	Julio M - Servicio Ruta-Parada Steinman A - Servicio Pago y Recarga Rubiano A Registro de nuevos usuarios	Alta	
8	Como equipo, necesitamos definir una fecha con el cliente para presentar el proyecto	Plasmar fecha y hora de reunion con el cliente	Rubiano A	Alta	

### 6.7.1. Weekly Report

---

27 /  
**TU** 05 /  
**E** 2025

NO.	ASSIG NED TO	PRIORITIES	PRIORITY LEVEL	PROGR ESS	PROGR ESS STATUS	PROBLEMS AND COMMENTS
1	Julio M.W	Implementacion e optimizacion de servicio Ruta-Parada	High	Complet ado	Comple te	
2	Julio M.W	Prueba de estres	High	Complet ado	Comple te	

3	Julio M.W	Revision y Actualizacion del README	High	En progreso	In Progress	
4	Julio M.W	Prueba funcional de test de UI	Low	No ha iniciado	Ready to Start	
5	Rubian o A.F	Despliegue funcional del frontend	High	No ha iniciado	Ready to Start	
6	Rubian o A.F	Implementar flutter y backend registros de nuevos usuarios	High	No ha iniciado	Ready to Start	
7	Rubian o A.F	Prueba de seguridad	High	No ha iniciado	Ready to Start	
8	Rubian o A.F	Concretar horario de reunion con cliente	High	No ha iniciado	Ready to Start	
9	Rubian o A.F, Julio M.W, Steinman A.P	Plantillas flutter en lista con microservicios implementados	High	Steinman A.P: Completado	In Progress	
10	Steinman A.P	Test de Integracion: Usuario: Supervisor	High	No puede realizarse hasta que se complete NO.8	On Hold	
11	Steinman A.P	Implementar flutter Pago y Recarga	High	Completado pago y recarga widgets,	On Hold	Se deben hacer añadidos a los microservicios vinculados para el correcto funcionamiento de los botones

				incompleto		
12	Steinman A.P	Revision de la documentacion del SRS	High	Listo para añadidos, esperando NO.9,2,7	On Hold	

29 /  
**TH** 05 /  
**U** 2025

NO.	ASSIG NED TO	PRIORITIES	PRIORITY LEVEL	PROGRESS	PROGRESS STATUS	PROBLEMS AND COMMENTS
1	Julio M.W	Implementacion e optimizacion de servicio Ruta-Parada	High	Funcional	Complete	
2	Julio M.W	Prueba de estres	High	Documentadas	Complete	
3	Julio M.W	Revision y Actualizacion del README	High	Done	Complete	
4	Julio M.W	Prueba funcional de test de UI	Low	Completado	Complete	
5	Rubiano A.F	Despliegue funcional del frontend	High	Implementando otro	In Progress	Se le acabo los creditos de azure
6	Rubiano A.F	Implementar flutter y backend registros de nuevos usuarios	High		In Progress	

7	Rubiano A.F	Prueba de seguridad	High		Ready to Start	
8	Rubiano A.F	Concretar horario de reunion con cliente	High	Cita apartada	Complete	
9	Rubiano A.F, Julio M.W, Steinman A.P	Plantillas flutter en lista con microservicios implementados	High	Steinman A.P: Completado, Mario: En Progreso, Rubiano: En Progreso	In Progress	
10	Steinman A.P	Test de Integracion: Usuario: Supervisor	High	No puede realizarse hasta que se complete NO.8	On Hold	
11	Steinman A.P	Implementar flutter Pago y Recarga	High	Completado pago y recarga widgets, incompleto	On Hold	Se deben hacer añadidos a los microservicios vinculados para el correcto funcionamiento de los botones
12	Steinman A.P	Revision de la documentacion del SRS	High	Listo para añadidos, esperando NO.9,2,7	On Hold	

### ***Impedimentos y Obstáculos***

Créditos acabados en Azure

### ***Próximos Pasos***

- Debe actualizarse el flutter

***Archivos Vinculados***

Daily Meeting # 1: [Daily #1](#)

Daily Meeting # 2: [Daily #2](#)

## **Apéndice A: Glosario**

*<Defina todos los términos necesarios para interpretar correctamente el SRS, incluidos los acrónimos y las abreviaturas. Puede crear un glosario independiente que abarque varios proyectos o toda la organización, y limitarse a incluir términos específicos de un solo proyecto en cada SRS.>*

## Apéndice B: Modelos de análisis

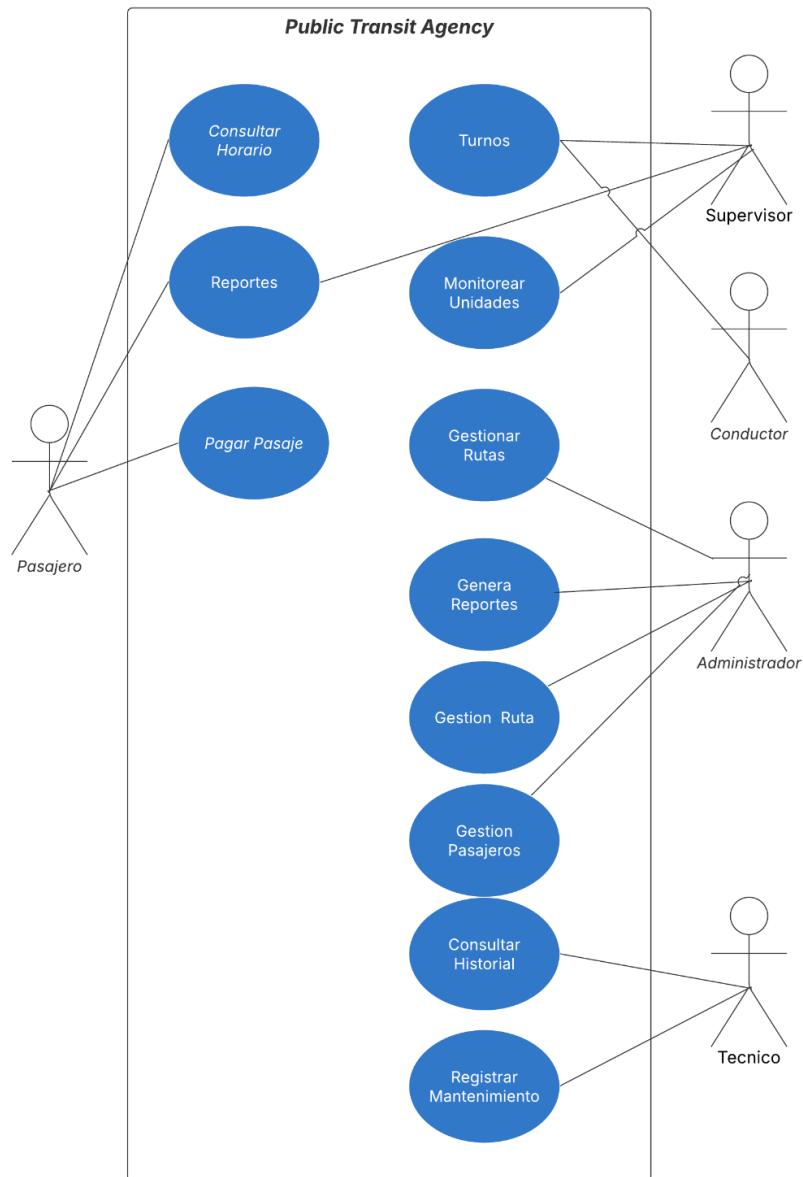


Imagen 1. Diagrama caso de uso

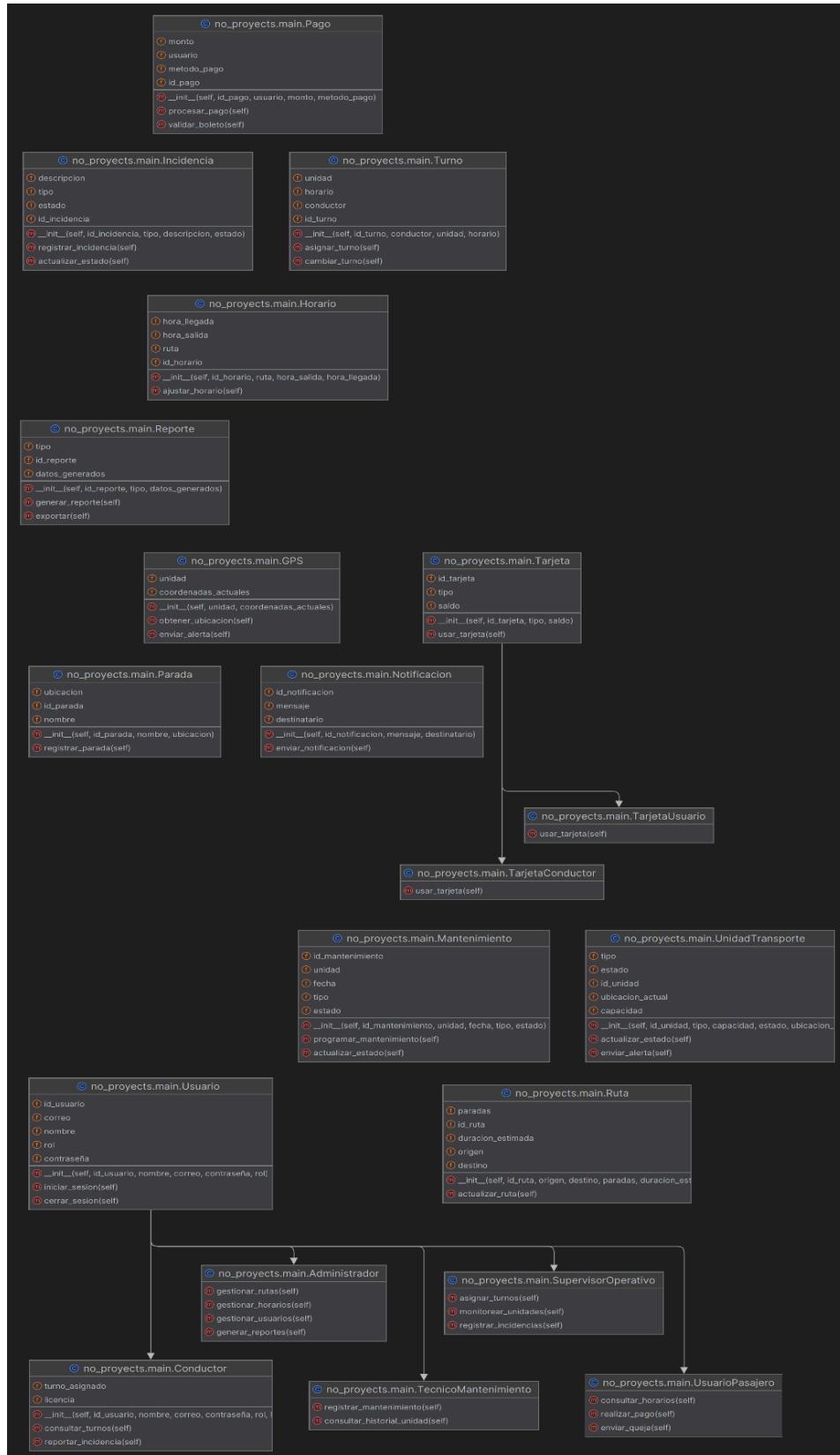


Imagen 2. Diagrama de Clases

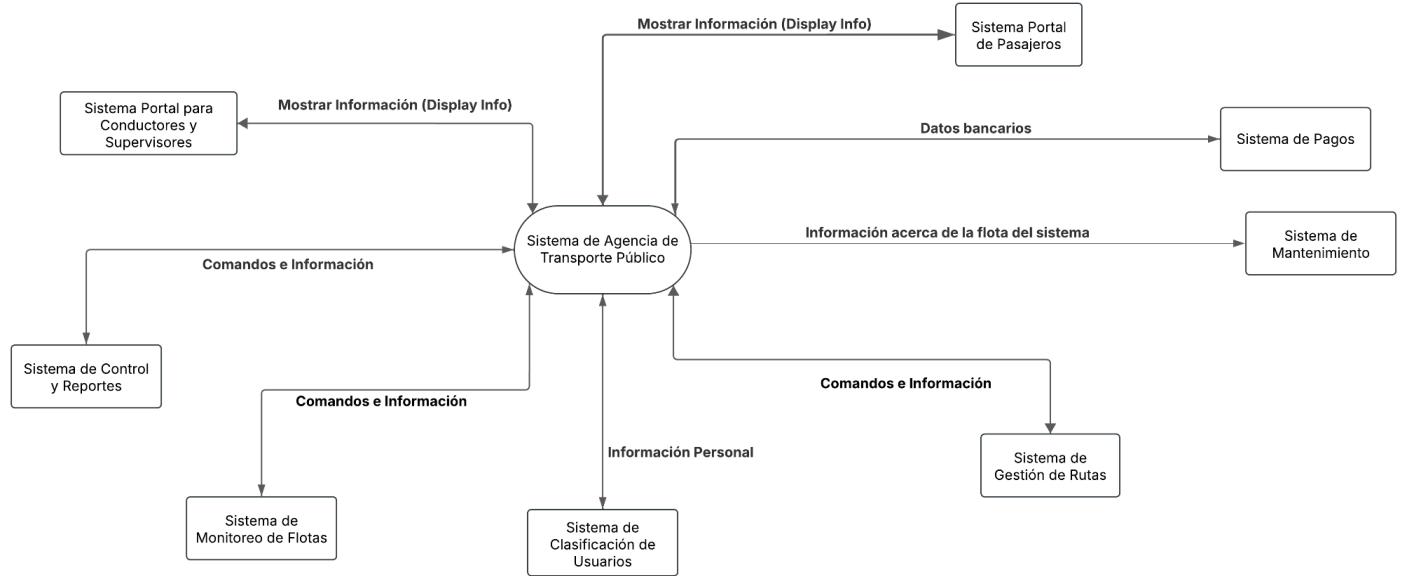


Imagen 3. Diagrama de Contexto

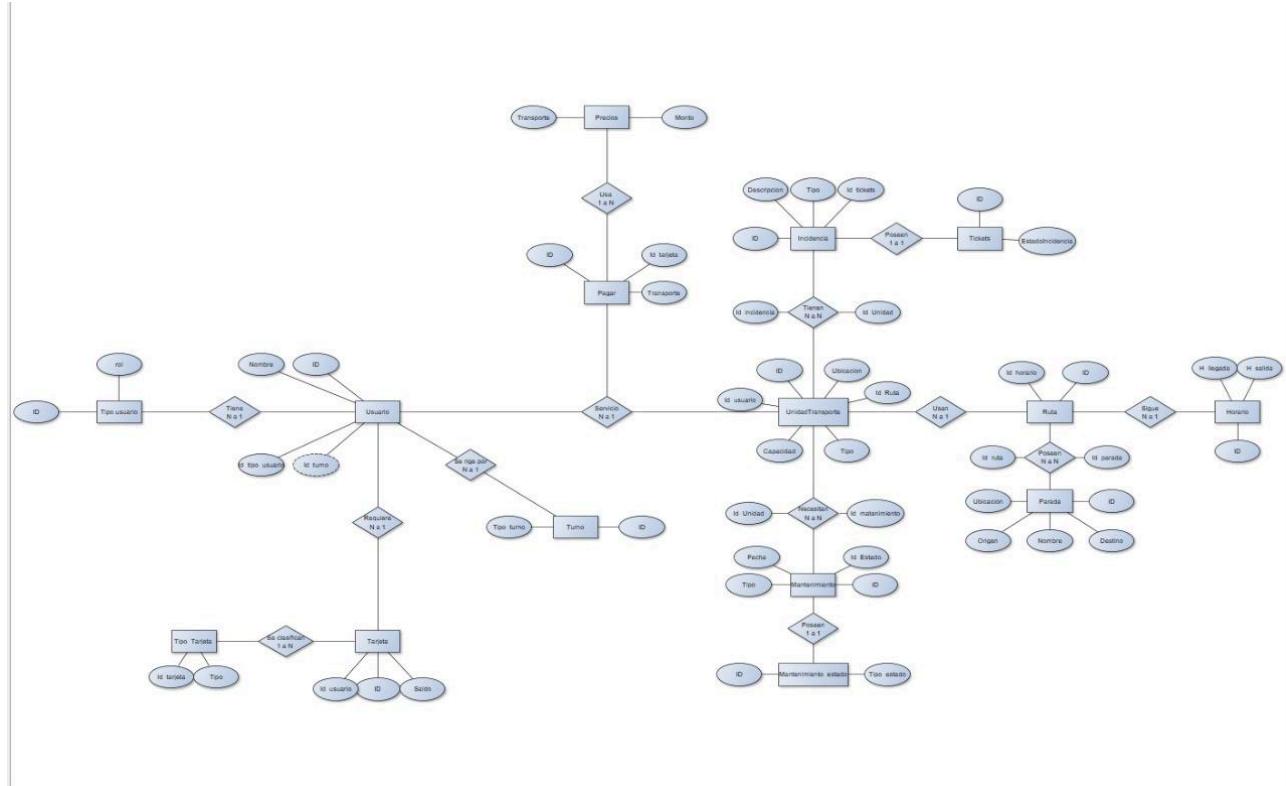
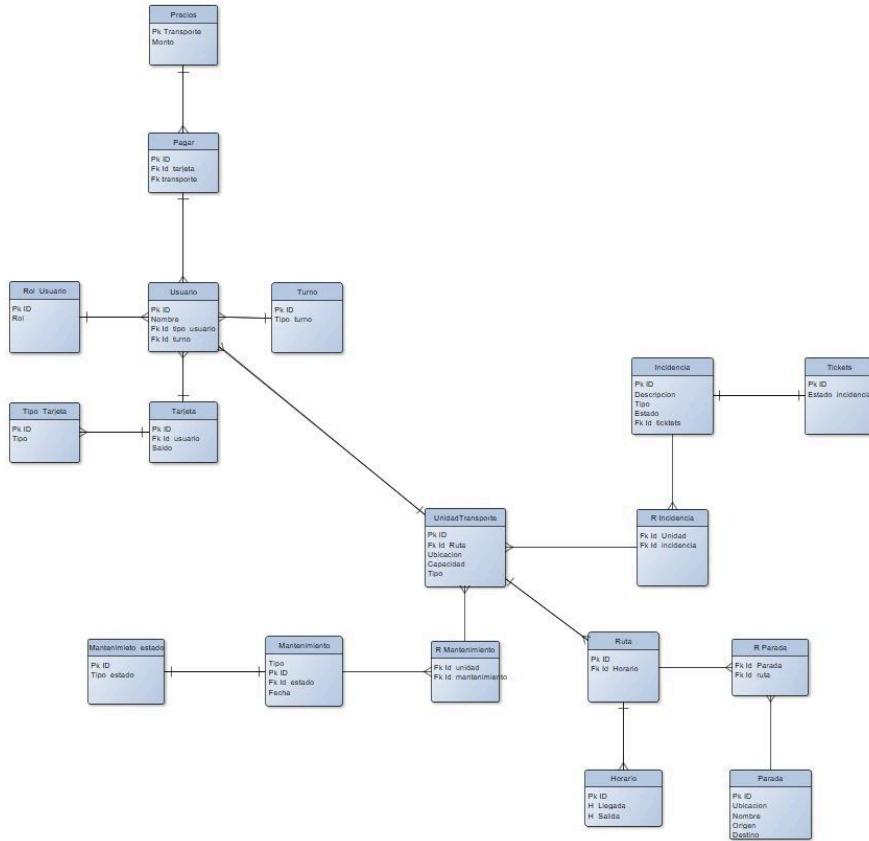


Imagen 4. Modelo Entidad - Relación



**Imagen 5. Modelo Relacional.**

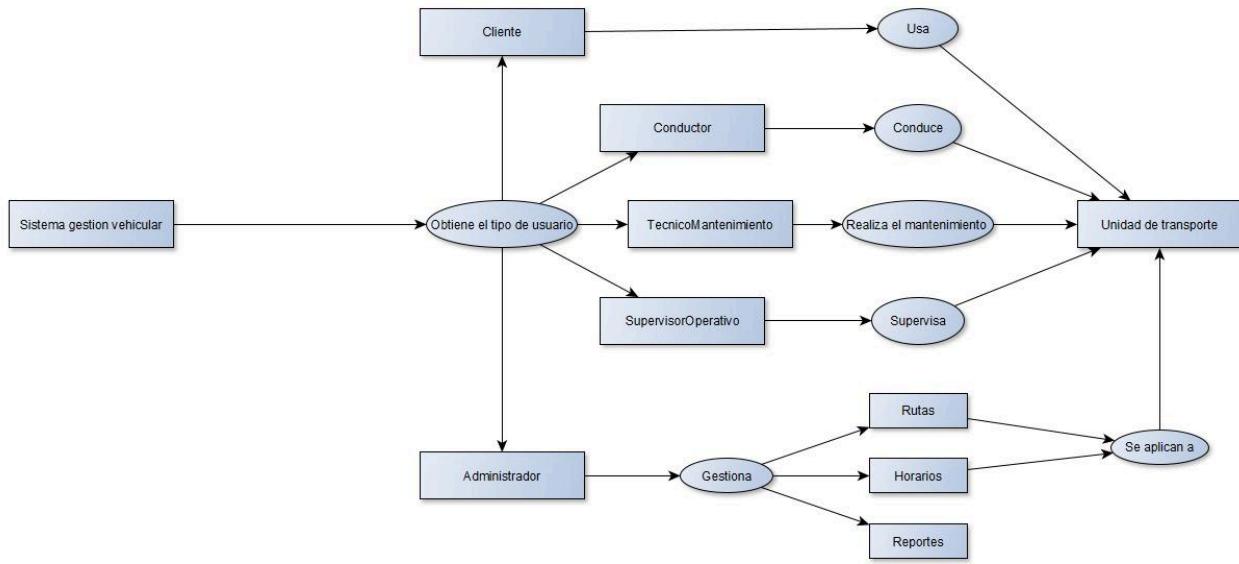


Imagen 6. Modelado basado en datos.



Imagen 7. Mockup UI: Inicio de sesión



Imagen 8. Mockup UI: Registro de pasajeros.

A screenshot of a mobile application interface titled "Panel General del Técnico". On the left, a sidebar titled "Técnico" contains four buttons: "Consultar historial de bus", "Itinerario", "Registrar Mantenimiento", and "Alertas". The main area has three cards: "Buses en Mantenimiento" (12), "Próximos Mantenimientos" (5), and "Historial Técnico" (45 registros). The background is a blurred image of a yellow bus.

Imagen 7. Mockup UI: Técnico.



Imagen 8. Mockup UI: Administrador.

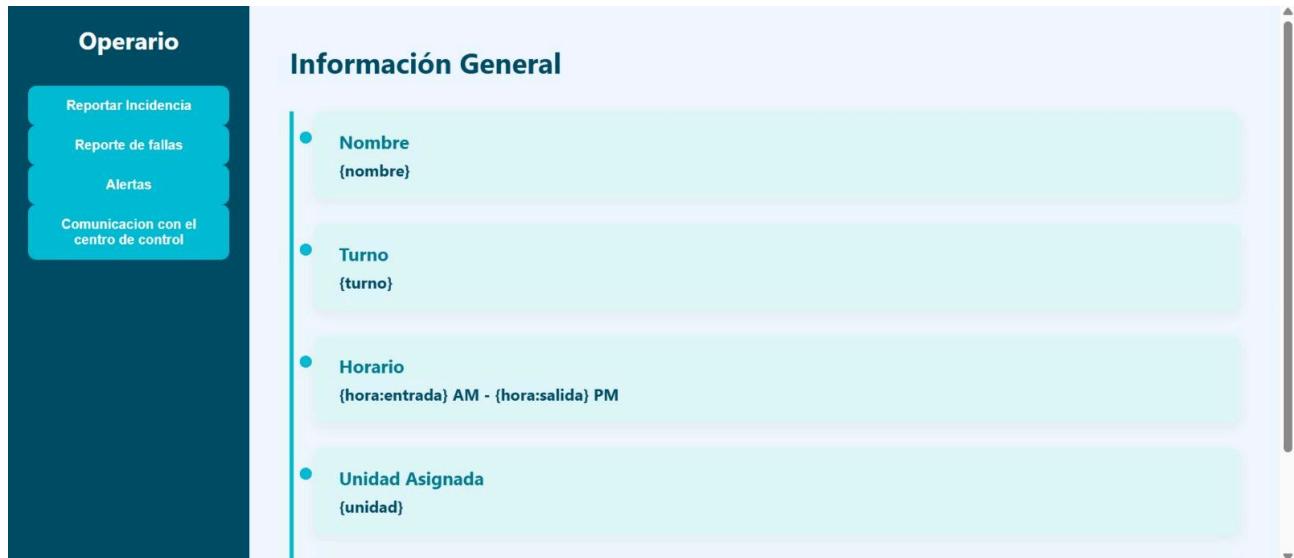


Imagen 9. Mockup UI: Operario.

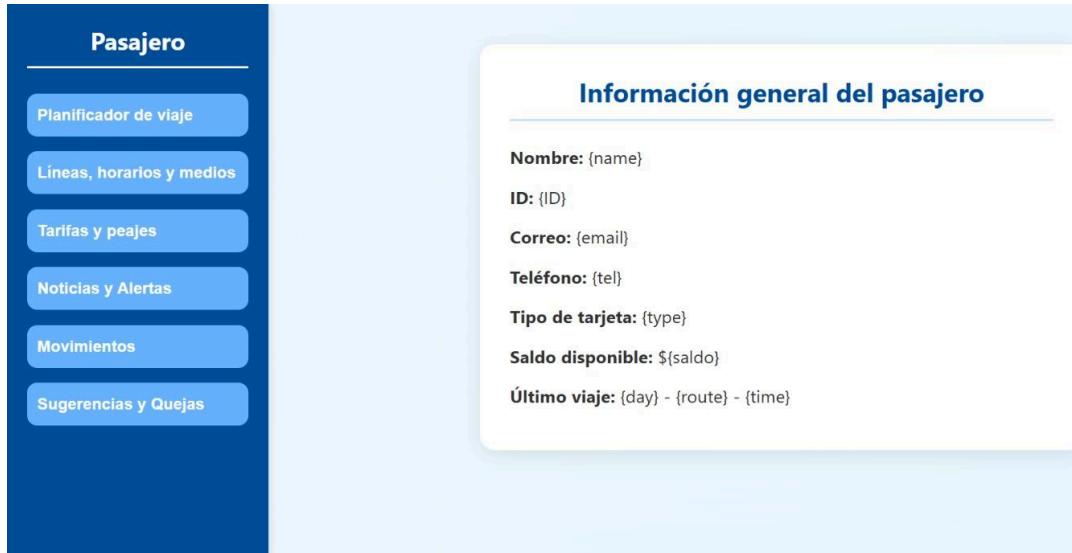
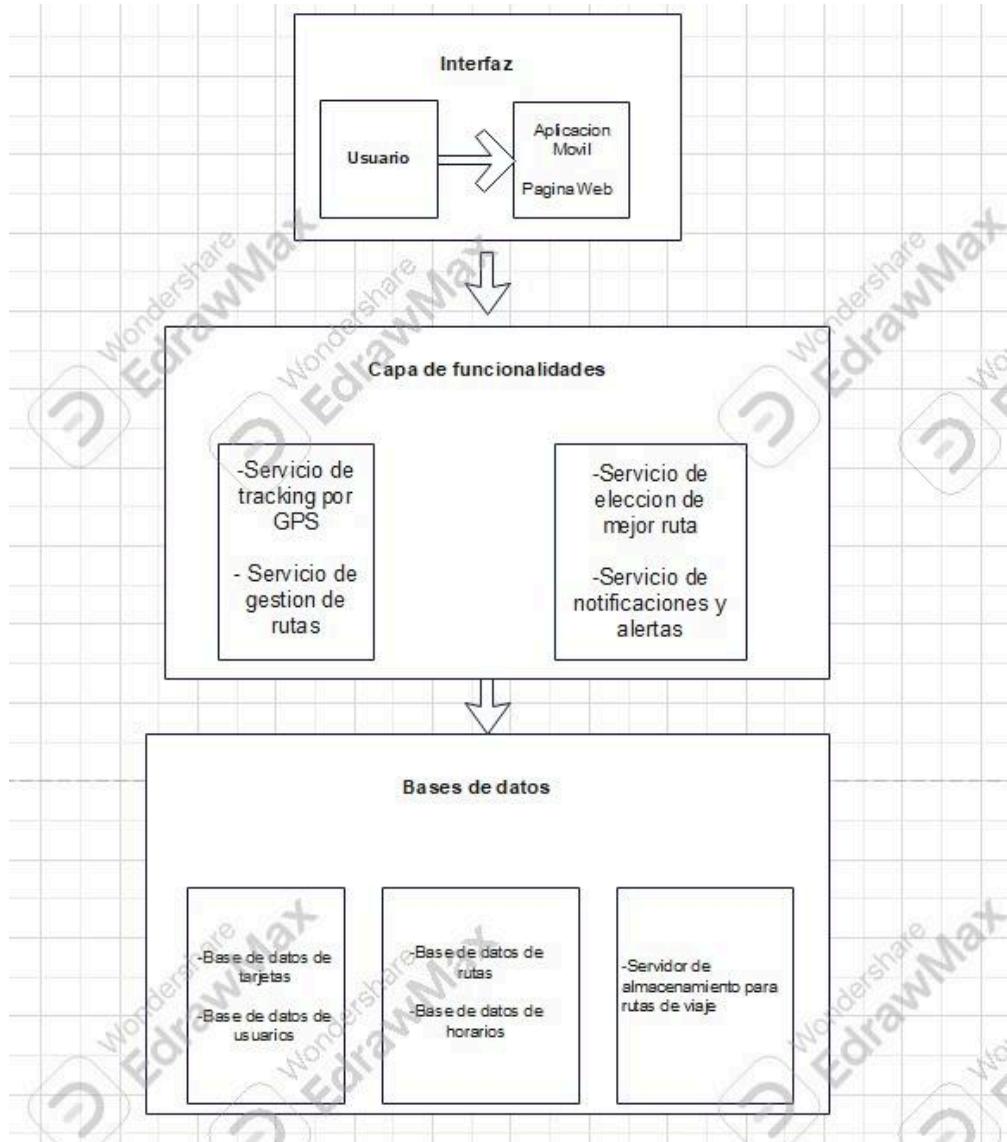


Imagen 10. Mockup UI: Pasajero.



Imagen 11. Mockup UI: Supervisor.



**Imagen 8. Modelo de arquitectura de software**

## Apéndice C: Lista de problemas

*<Esta es una lista dinámica de los problemas de requisitos abiertos que quedan por resolver, incluyendo TBDs, decisiones pendientes, información que se necesita, conflictos que esperan ser resueltos, y similares.>*