UNIVERSIDAD TECNICA
FEDERICO SANTA MARIA

DEPARTAMENTO DE ELECTROTECNIA E INFORMÁTICA
INGENIERÍA EN INFORMÁTICA

# Plataforma backend basada en microservicios para aplicación cliente del sistema Ecovuelta

Luis Muñoz Aguilera

luis.munoza@sansano.usm.cl

Carlos Felipe Alten López
Profesor Guía

**Resumen:** En la industria logística, los viajes de camiones sin carga representan una ineficiencia significativa en el uso de recursos como tiempo, combustible y mano de obra, además de tener un impacto ambiental negativo. Este proyecto se centra en el diseño un backend la interacción y flexibilidad de una plataforma web destinada a abordar este problema. El sistema incluye dos microservicios clave: uno para gestionar las APIs, que maneja las solicitudes de clientes y la promoción de servicios de transporte, y otro dedicado a la integración con un modelo que realiza cálculos como la estimación de paradas y la verificación de autorizaciones de vehículos. La solución propuesta maximiza el aprovechamiento de los viajes de retorno de camiones, permitiendo a las empresas de logística monetizar trayectos que actualmente no generan ingresos, y contribuye indirectamente a la reducción de la huella de carbono vacía.

**Palabras Clave**: Microservicios, APIs, Logística.

## 1 Introducción

### 1.1 Contexto y antecedentes

El sector logístico constituye un pilar fundamental en la economía chilena, y un claro ejemplo es cuando estos se movilizan y pueden paralizar el país. Sin embargo, uno de los más grandes y críticos desafíos de esta industria son los denominados "viajes vacíos", es decir, los trayectos de retorno de camiones sin carga. Estos desplazamientos no solo representan una considerable ineficiencia operativa, manifestada en el desperdicio de recursos como tiempo, combustible y mano de obra, sino que también contribuyen de manera significativa al aumento innecesario de las emisiones de gases de efecto invernadero. En un escenario global en el que la reducción de la huella de carbono se ha convertido en una prioridad, el sector logístico se encuentra bajo una creciente presión para adoptar prácticas más sostenibles y eficientes.

Actualmente, diversas empresas están explorando y comenzando a implementar soluciones innovadoras con el objetivo de mitigar los impactos ambientales asociados a estos viajes vacíos, incluyendo la adopción de tecnologías más limpias, como los camiones eléctricos. No obstante, a pesar de que estos vehículos ofrecen la promesa de reducir las emisiones directas de gases contaminantes, aún existen importantes desafíos vinculados a la infraestructura de carga y a la limitada autonomía de estos en escenarios de largos trayectos. En consecuencia, mientras se avanza en la transición hacia tecnologías más sostenibles, la optimización del uso de los camiones en los trayectos de retorno se ve como una estrategia indispensable para mejorar la eficiencia y sostenibilidad del sector.

En el contexto chileno, la logística se enfrenta a una serie de obstáculos que dificultan la eficiencia en la gestión de los viajes de camiones. La falta de sincronización entre las empresas de transporte y los clientes que requieren servicios de carga se traduce en una elevada frecuencia de viajes vacíos. Este fenómeno no solo eleva los costos operativos para las empresas, sino que también incrementa las emisiones de carbono, reflejando un lamentable uso de los recursos. La fragmentación del mercado, caracterizado por una

predominancia de pequeñas y medianas empresas con limitados recursos tecnológicos, complica la implementación de sistemas de gestión avanzados que podrían mejorar dicha coordinación.

Aunque se han desarrollado diversas soluciones tecnológicas, tales como plataformas de carga compartida y sistemas de gestión de transporte (TMS), que buscan conectar de manera más eficiente a transportistas con clientes, optimizando así la utilización de los vehículos, estas soluciones no siempre logran adaptarse de manera efectiva a las particularidades del mercado chileno. Entre las limitaciones que enfrentan destacan la elevada inversión inicial requerida, la complejidad en la integración tecnológica, y la necesidad de una mayor automatización en la gestión de datos. Aunque algunas empresas han comenzado a incorporar tecnologías más limpias, como los camiones eléctricos, estas iniciativas aún se encuentran en fases iniciales y no abordan la problemática oculta de la falta de carga en los trayectos de retorno. En consecuencia, existe una necesidad urgente de desarrollar y adoptar soluciones que no solo mejoren la eficiencia del transporte, sino que también reduzcan los costos y contribuyan al cumplimiento de los objetivos de sostenibilidad del sector logístico en Chile.

### 1.2 Definición del problema

En la industria logística chilena, uno de los dolores más persistentes y de alto impacto es la gestión ineficiente de los viajes de retorno de camiones sin carga, comúnmente denominados "viajes vacíos". Estos trayectos ocurren cuando los camiones regresan a sus puntos de origen o se desplazan a nuevas ubicaciones sin transportar mercancías, lo que resulta en una utilización ineficiente significativa de los recursos disponibles. Esta ineficiencia no solo tiene repercusiones económicas, aumentando los costos operativos para las empresas de transporte, sino que también agrava el impacto ambiental del sector, debido a un consumo innecesario de combustible y la consecuente emisión de gases de efecto invernadero.

El problema se ve aumentado por la falta de una coordinación efectiva entre las diferentes partes involucradas en el proceso logístico, incluyendo transportistas, operadores logísticos y empresas que demandan servicios de transporte. A menudo, estas partes operan de manera aislada, lo que dificulta la sincronización entre la oferta de vehículos disponibles y la demanda de transporte para los trayectos de retorno. Como resultado, los camiones realizan largos trayectos sin carga, desperdiciando tiempo valioso y recursos que podrían haberse utilizado de manera más eficiente.

Además, la falta de adopción de tecnologías avanzadas y sistemas de información integrados en la logística chilena agrava esta problemática. A pesar de la existencia de soluciones tecnológicas que podrían mitigar estos problemas, como sistemas de gestión de transporte (TMS) y plataformas de carga compartida, su adopción ha sido limitada debido a factores como los altos costos de implementación, la complejidad de la integración con sistemas existentes y la falta de personal capacitado para utilizarlas de manera efectiva.

En este contexto, el problema principal es la incapacidad de maximizar la utilización de los camiones en los trayectos de retorno, lo que genera no solo pérdidas económicas para las empresas de logística, sino también un impacto negativo en el medio ambiente, contrarrestando los esfuerzos por reducir la huella de carbono y mejorar la sostenibilidad del sector.

### 1.3 Breve descripción sobre la propuesta de solución

La solución propuesta para EcoVuelta consiste en una plataforma web diseñada para optimizar la logística de transporte terrestre. El objetivo principal es reducir los viajes vacíos de camiones, optimizando recursos logísticos y mejorando la eficiencia operativa. La plataforma permite a las empresas gestionar solicitudes de transporte, calcular rutas óptimas, y asignar recursos, como camiones y conductores, de manera dinámica y eficiente.
La propuesta se basa en una arquitectura distribuida compuesta por microservicios independientes, cada uno enfocado en un dominio específico. Esto asegura una alta modularidad, escalabilidad y facilidad de mantenimiento, adaptándose a las necesidades actuales y futuras de la plataforma.

El rol del backend en esta solución es fundamental, ya que constituye el núcleo lógico y de procesamiento del sistema. Sus principales responsabilidades incluyen:

- Gestionar las solicitudes de transporte, validando datos y almacenándolos de manera eficiente.
- Calcular rutas óptimas para los trayectos mediante un modelo de optimización alojado como un servicio externo.
- Administrar los recursos logísticos disponibles (camiones y conductores), garantizando una asignación eficiente basada en capacidad y restricciones operativas.
- Asegurar la autenticación y autorización de los usuarios mediante tokens JWT, protegiendo la integridad y seguridad de la plataforma.
- Facilitar la comunicación entre los componentes del sistema, integrando microservicios y bases de datos mediante APIs RESTful.
-

**Límites del proyecto:**

- **Incluido en el desarrollo:**
  - Implementación de la plataforma web con los módulos principales:
    - **Gestión de Solicitudes**: Permitir a los usuarios crear, consultar y gestionar solicitudes de transporte.
    - **Planificación de Rutas**: Procesar datos de solicitudes y calcular rutas óptimas mediante un modelo de optimización.
    - **Gestión de Recursos Logísticos**: Controlar la información y disponibilidad de camiones y conductores.
    - **Autenticación y Autorización**: Implementar un sistema seguro basado en JWT.
  - Diseño e implementación de un backend modular con microservicios independientes desplegados en contenedores Docker.
  - Gestión de bases de datos NoSQL (MongoDB) para almacenar datos de usuarios, solicitudes, rutas y recursos logísticos.
  - Integración con un modelo de optimización desarrollado en Python, accedido a través de un servicio externo.
- **Fuera del alcance del desarrollo:**
  - Diseño detallado del frontend (a cargo de un equipo diferente).
  - Desarrollo de un sistema de notificaciones en tiempo real o alertas (posibles mejoras futuras).
  - Integraciones externas con servicios de geolocalización o tráfico en tiempo real.
  - Orquestación avanzada de contenedores mediante Kubernetes (posible mejora futura).
  - Optimización avanzada del modelo de cálculo de rutas (se utilizará un modelo funcional existente).

Este enfoque busca entregar una solución eficiente y modular, que permita abordar las necesidades inmediatas de optimización logística, mientras establece una base sólida para futuras expansiones.

## 1.4 Objetivos Generales y Específicos de la Tesina

### 1.4.1 Objetivo General

Diseñar e implementar un backend para optimizar la gestión logística del transporte terrestre, permitiendo la reducción de viajes vacíos, el cálculo de rutas óptimas y la asignación eficiente de recursos como camiones y conductores, garantizando seguridad, modularidad y escalabilidad en la integración de sus componentes.

### 1.4.2 Objetivos Específicos

- Diseñar e implementar un mecanismo de autenticación y autorización de usuarios:
  Garantizar la seguridad en las interacciones mediante una solución robusta para la validación de usuarios y el control de acceso a las diferentes funcionalidades.
- Diseñar e implementar una solución para la gestión de solicitudes y ubicaciones:
  Facilitar la creación, consulta y administración de solicitudes de transporte, asegurando la correcta organización y flujo de información asociada a los puntos de origen y destino.
- Diseñar e implementar un mecanismo para integrar el sistema de planificación y optimización de rutas:
  Conectar de manera eficiente y segura con el sistema externo de optimización, asegurando el envío de datos relevantes y la correcta recepción de las rutas calculadas, contribuyendo a la reducción de costos y mejoras de la logística operativa.
- Diseñar una herramienta para la gestión de recursos logísticos:
  Centralizar y organizar la información relacionada con recursos disponibles, como camiones y conductores, asegurando su adecuada asignación en función de las necesidades operativas.
- Garantizar la integración efectiva entre los diferentes componentes del sistema:
  Establecer mecanismos que permitan la interoperabilidad entre los distintos elementos, asegurando la consistencia y disponibilidad de los datos.
- Definir estrategias de manejo de errores y monitoreo del sistema:
  Implementar métodos para identificar, registrar y gestionar fallos en el sistema, permitiendo la resolución ágil de problemas y asegurando la continuidad operativa.
- Elaborar documentación detallada sobre la solución propuesta y sus componentes:
  Proporcionar una guía clara sobre la estructura del sistema, los procedimientos implementados y los flujos de trabajo para facilitar su mantenimiento y evolución.

## 1.5 Justificación de proyecto

El proyecto de diseño de un sistema backend para la gestión de camiones en la industria logística chilena se justifica por varias razones fundamentales. En primer lugar, aborda una necesidad crítica en el sector logístico: la optimización de los viajes de retorno de camiones, comúnmente conocidos como "viajes vacíos". Estos viajes representan una ineficiencia significativa, ya que se estima que entre el 20% y el 30% de los trayectos de camiones se realizan sin carga, lo que genera pérdidas económicas considerables y aumenta las emisiones de $CO2$ debido al consumo innecesario de combustible [1]. Además, los viajes vacíos contribuyen negativamente al impacto ambiental, incrementando las emisiones de gases de efecto invernadero. Según la European Environment Agency, el transporte de mercancías por carretera contribuye con el 10-15% de las emisiones globales de gases de efecto invernadero, y reducir los viajes vacíos es clave para disminuir este porcentaje [2].

Desde un punto de vista técnico, un sistema backend basado en un enfoque modular es una solución moderna y flexible que se adapta bien a la complejidad de la gestión logística. Este tipo de sistema facilita que las diferentes partes, como el frontend, la base de datos y los servicios que asignan cargas a los camiones, trabajen de manera eficiente y segura. Además, permite manejar grandes cantidades de datos de forma rápida, lo que es esencial para procesar muchas solicitudes y gestionar información en tiempo real. Esta arquitectura no solo hace que el sistema sea más fácil de mantener y ajustar, sino que también mejora su rendimiento y capacidad para responder a las demandas de un entorno logístico dinámico.

Desde una perspectiva económica, la implementación de este sistema backend tiene el potencial de generar ahorros significativos para las empresas de transporte al reducir los costos asociados con los viajes vacíos. Al maximizar la utilización de los camiones en los trayectos de retorno, las empresas pueden mejorar su rentabilidad operativa y, al mismo tiempo, ofrecer precios más competitivos a sus clientes [3]. Se estima que

optimizar los trayectos de los camiones puede generar ahorros de hasta un 15% en los costos operativos, lo que beneficiaría no solo a las empresas directamente involucradas, sino también a toda la cadena de suministro, al mejorar la eficiencia general del sector logístico [4].

Finalmente, desde un punto de vista social y ambiental, el proyecto contribuye a los esfuerzos globales por reducir la huella de carbono y promover prácticas más sostenibles en la industria del transporte. Al optimizar los trayectos de los camiones y minimizar el número de viajes vacíos, se reduce la cantidad de combustible consumido y, en consecuencia, las emisiones de gases de efecto invernadero. Este enfoque no solo responde a las crecientes demandas de sostenibilidad por parte de la sociedad y los reguladores, sino que también posiciona a las empresas que adopten esta tecnología como líderes en responsabilidad ambiental dentro del sector [5].

## 1.6 Metodología

El desarrollo del backend para el sistema EcoVuelta se basará en la metodología ágil Scrum, seleccionada por su capacidad para adaptarse a proyectos en los que los requisitos pueden evolucionar con el tiempo y donde la retroalimentación continua es fundamental. Scrum permite organizar el trabajo en sprints cortos, típicamente de dos a tres semanas, lo que facilita la entrega incremental de funcionalidades completas y revisables. Este enfoque fomenta la colaboración constante entre los miembros del equipo, asegurando que el desarrollo esté alineado con los objetivos del proyecto en cada etapa.
La estructura del proyecto incluye la planificación detallada al inicio de cada sprint, donde se definen las tareas prioritarias a abordar. Estas tareas se dividen en incrementos pequeños y manejables, enfocados en aspectos específicos del backend, como el diseño y desarrollo de APIs, la integración con la base de datos MongoDB y con el modelo de optimización en Python, y la implementación de patrones de diseño como el Circuit Breaker. Durante el sprint, se realizan reuniones breves y diarias para revisar el progreso, resolver obstáculos y garantizar que todos los miembros estén alineados. Al final del sprint, se realiza una revisión para presentar los avances y recopilar retroalimentación, seguida de una retrospectiva para evaluar el proceso y proponer mejoras para el siguiente ciclo.
Scrum también permite ajustarse rápidamente a cambios en las prioridades del proyecto. Por ejemplo, si se identifican necesidades adicionales relacionadas con la funcionalidad de ubicaciones (locations) o ajustes en la integración con el modelo de optimización, estas pueden incorporarse de manera organizada en sprints futuros sin afectar significativamente el cronograma general. Esta flexibilidad es particularmente importante dado que el backend involucra múltiples tecnologías y microservicios interconectados.
Para garantizar la calidad de las funcionalidades desarrolladas, se utilizarán herramientas como Postman y JMeter para verificar las respuestas de los endpoints. Estas pruebas asegurarán que los servicios respondan correctamente a las solicitudes y que las interacciones entre los microservicios se desarrollen sin problemas. Este enfoque centrado en la verificación de los endpoints permite identificar y corregir errores de manera eficiente antes de avanzar al siguiente sprint.
El desarrollo se apoyará en reuniones regulares, como las reuniones de planificación al inicio de cada sprint y las revisiones y retrospectivas al final, que asegurarán que el equipo mantenga una mejora continua. La entrega incremental de funcionalidades permitirá al equipo iterar rápidamente y adaptarse a las necesidades cambiantes del proyecto, asegurando un backend funcional, seguro y eficiente que cumpla con los objetivos de EcoVuelta.

## 1.7 Breve descripción de la organización del informe en capítulos

El presente documento está estructurado en capítulos que guían al lector a través del desarrollo del proyecto EcoVuelta, desde su concepción hasta los resultados y conclusiones. A continuación, se describe brevemente el contenido de cada capítulo:

Capítulo 1: Introducción
Este capítulo establece el contexto y antecedentes del proyecto, definiendo el problema y planteando una breve descripción de la solución propuesta. También presenta los objetivos generales y específicos, la justificación, y la metodología utilizada.

Capítulo 2: Marco Teórico
En este capítulo se desarrollan los fundamentos teóricos del desarrollo backend, explorando las tecnologías, frameworks y herramientas seleccionadas, así como la arquitectura de software utilizada. Se explican los patrones de diseño aplicados y su importancia en la construcción del sistema.

Capítulo 3: Diseño e Implementación
Este capítulo detalla el diseño del sistema, describiendo la arquitectura, la estructura de los componentes y las prácticas de desarrollo aplicadas. Se incluye información sobre la codificación, los métodos de comprobación empleados y las estrategias utilizadas para la resolución de errores detectados y la optimización del sistema.

Capítulo 4: Conclusiones
Aquí se reflexiona sobre las lecciones aprendidas, los logros alcanzados y los desafíos enfrentados durante el desarrollo. Además, se plantean recomendaciones y posibles mejoras para futuras iteraciones del sistema.

# 2 Marco Teórico

Durante el capítulo del marco teórico se establecen los fundamentos conceptuales y técnicos que sustentan el desarrollo del backend de la plataforma EcoVuelta. Este capítulo presenta las bases necesarias para comprender los elementos clave que intervienen en el sistema, abarcando desde los principios del desarrollo backend hasta las tecnologías y metodologías seleccionadas para su implementación.

Se detallan conceptos esenciales como las arquitecturas basadas en microservicios, las ventajas de un diseño modular y desacoplado, y el uso de APIs RESTful para garantizar una comunicación eficiente entre los componentes del sistema. Además, se explican los patrones de diseño adoptados, como Circuit Breaker y Repository, los cuales permiten mejorar la resiliencia, flexibilidad y mantenimiento del sistema.

Asimismo, el capítulo aborda las tecnologías específicas utilizadas, incluyendo Spring Boot, FastAPI, MongoDB, y herramientas de testing como Postman y JMeter, justificando su elección en función de las necesidades del proyecto. También se describe la metodología Scrum aplicada durante el desarrollo, resaltando su importancia en la organización del trabajo en sprints iterativos para lograr entregas incrementales y adaptativas.

## 2.1 Fundamentos del desarrollo backend

El desarrollo backend es una parte fundamental en la creación de sistemas de software, ya que se encarga de manejar la lógica de negocio, el procesamiento de datos y la comunicación entre las diferentes partes del sistema. A diferencia del frontend, que es la capa visible para los usuarios (interfaz gráfica, botones, formularios, etc.), el backend trabaja detrás de escena para asegurarse de que la aplicación funcione correctamente. Este componente interactúa con la base de datos para almacenar, recuperar y manipular datos, procesando solicitudes del frontend y respondiendo con la información requerida, todo de manera eficiente y segura.

El desarrollo backend también se enfoca en asegurar la modularidad y flexibilidad del sistema, lo que significa que cada componente puede operar de forma independiente, facilitando el mantenimiento y la adaptación a futuros cambios o expansiones. Esta modularidad es clave en proyectos complejos, ya que permite dividir responsabilidades, optimizar recursos y mantener un control más claro sobre las funcionalidades específicas. En este contexto, el marco teórico no solo incluye las tecnologías utilizadas, sino también los métodos y enfoques que facilitan la gestión del proyecto. La elección de arquitecturas como la basada en microservicios refleja una tendencia moderna que optimiza la separación de responsabilidades y la escalabilidad del sistema. Los microservicios dividen la lógica del backend en componentes pequeños e independientes, como la gestión de usuarios, solicitudes, planificación de rutas y recursos logísticos, cada uno con su propia base de datos y tecnología adecuada. Este enfoque también facilita el uso de patrones de diseño como Observer y Circuit Breaker, que aseguran la flexibilidad y tolerancia a fallos en el sistema.

Para la gestión del proyecto, se utilizó Scrum como metodología ágil, que organiza el trabajo en ciclos iterativos llamados sprints. Cada sprint tiene una duración definida, generalmente de dos a tres semanas, y su objetivo es entregar incrementos funcionales del producto que permitan una retroalimentación constante y ajustes rápidos. Scrum fomenta la colaboración continua entre los miembros del equipo y con los interesados externos, asegurando que las prioridades estén alineadas con las necesidades del proyecto. Este enfoque fue ideal para abordar tareas relacionadas con la integración de APIs, bases de datos y otros componentes críticos del backend, ajustando las prioridades según la evolución del sistema.

Desde el punto de vista técnico, el backend de EcoVuelta incorpora tecnologías como Spring Boot, FastAPI y MongoDB, seleccionadas por su capacidad para manejar grandes volúmenes de datos, asegurar la integridad y optimizar el rendimiento. Estas tecnologías permiten integrar el modelo de optimización con el sistema, gestionar recursos logísticos y proporcionar una experiencia fluida para los usuarios finales. Además, el uso

de herramientas como Postman y JMeter asegura que las APIs sean verificadas en cada iteración, garantizando la funcionalidad y fiabilidad del sistema.

Este enfoque combinado de metodologías ágiles, arquitecturas modernas y herramientas técnicas asegura que el backend sea robusto, adaptable y eficiente. Así, se establece una base sólida para el desarrollo de una plataforma que no solo cumpla con los requerimientos actuales, sino que también se mantenga preparada para responder a las demandas futuras del sector logístico.

## 2.2 Arquitectura de software

La arquitectura del backend de la plataforma EcoVuelta se basa en un enfoque modular y orientado a microservicios, diseñado para maximizar la escalabilidad, flexibilidad y mantenimiento del sistema. Este enfoque permite dividir el sistema en componentes independientes, cada uno con responsabilidades bien definidas. La arquitectura se estructura en múltiples microservicios que gestionan dominios específicos, como autenticación, solicitudes, planificación de rutas y gestión de camiones y conductores. Además, estos microservicios interactúan entre sí mediante APIs RESTful, facilitando la comunicación de manera eficiente y segura.

El flujo general del sistema puede observarse en el diagrama presentado, donde se destacan los principales componentes y sus interacciones. Cada microservicio tiene su propia base de datos MongoDB, diseñada para garantizar la independencia de datos, optimizar el rendimiento y evitar bloqueos al manejar grandes volúmenes de información.

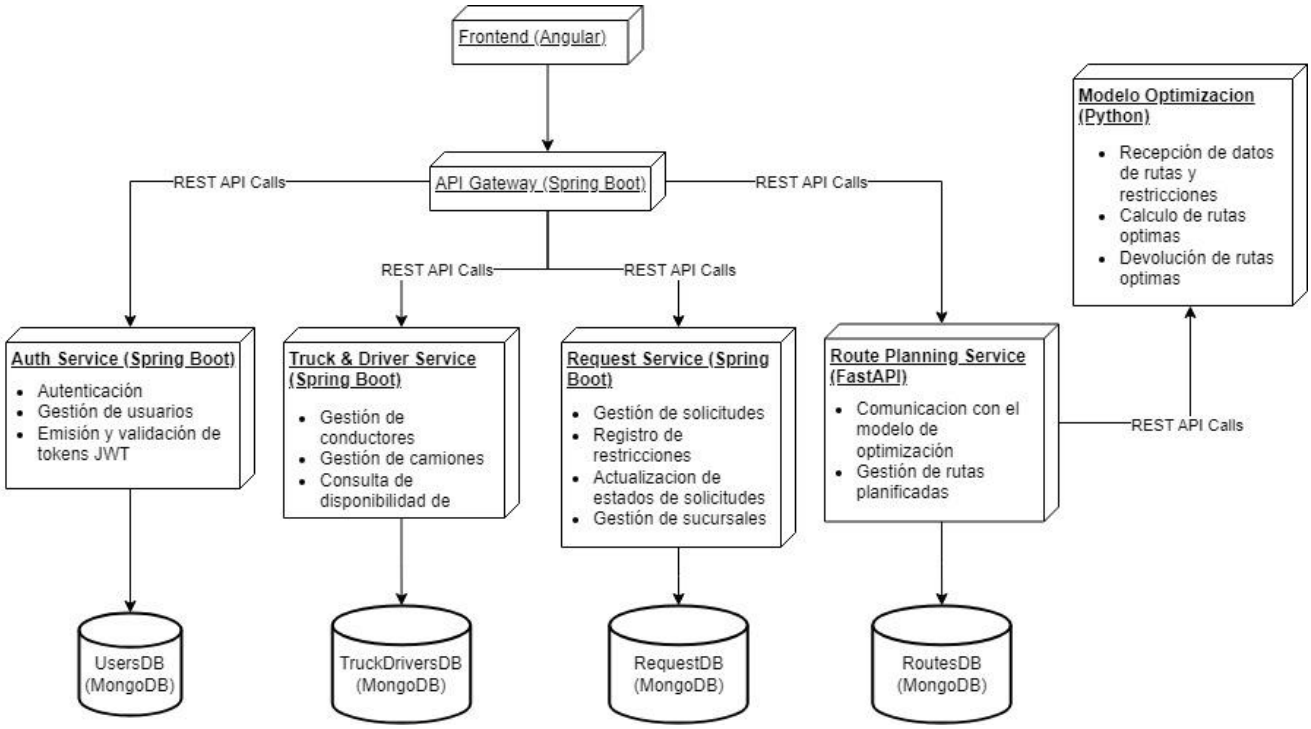


**Ilustración 2.1 Diagrama de la arquitectura del backend de EcoVuelta**

Los elementos principales de la arquitectura son:

- El Frontend actúa como la interfaz del usuario, desarrollada en Angular, que interactúa directamente con el API Gateway.

- El API Gateway, implementado en Spring Boot, centraliza todas las solicitudes entrantes, valida los tokens de autenticación mediante JWT y enruta las peticiones hacia los microservicios correspondientes.

  Los microservicios gestionan funcionalidades específicas:

  - **Auth Service**: Gestiona la autenticación y autorización de usuarios.
  - **Request Service**: Administra las solicitudes de transporte y la información de ubicaciones asociadas.
  - **Route Planning Service**: Se encarga de la planificación de rutas optimizadas mediante la integración con un modelo de optimización en Python.
  - **Truck y Driver Service**: Centraliza la gestión de camiones y conductores, incluyendo la consulta de disponibilidad.

En el diseño del backend se han aplicado varios patrones de diseño, seleccionados para garantizar que el sistema sea modular, flexible y fácil de mantener. Estos patrones también aseguran la escalabilidad y la capacidad de respuesta frente a cambios futuros.

1. **Patrón Singleton**: Este patrón garantiza que ciertos componentes críticos del sistema tengan una única instancia durante toda la ejecución. Se utiliza para gestionar configuraciones globales, puntos de acceso compartidos y recursos como conexiones a la base de datos, asegurando la consistencia y optimización de recursos.

2. **Patrón Circuit Breaker**: Implementado para garantizar la tolerancia a fallos en las comunicaciones entre microservicios. Este patrón supervisa las solicitudes y abre el circuito cuando detecta fallos repetidos o un tiempo de respuesta excesivo, devolviendo respuestas predeterminadas para mantener la estabilidad del sistema.

3. **Patrón Observer**: Este patrón, aunque no está implementado en la etapa inicial, se considera para futuras expansiones del sistema. Permitirá que los componentes reaccionen automáticamente a los cambios en los datos, facilitando la sincronización sin necesidad de acoplamiento fuerte.

4. **Patrón Strategy**: Utilizado para manejar diferentes estrategias de procesamiento, como la asignación de cargas o el cálculo de rutas. Este patrón permite que el sistema aplique distintas estrategias según los parámetros específicos de una solicitud, ofreciendo flexibilidad y modularidad en la lógica de negocio.

La arquitectura seleccionada para EcoVuelta ofrece múltiples ventajas que refuerzan su capacidad para abordar las necesidades actuales y futuras del sistema. Cada microservicio está diseñado para operar de manera independiente, lo que no solo facilita su desarrollo y despliegue, sino que también garantiza que puedan escalarse de forma individual según la demanda. Esta independencia reduce los riesgos asociados a fallos en un componente, evitando que impacten al sistema en su totalidad.

Además, la arquitectura proporciona una flexibilidad significativa al permitir la incorporación o modificación de microservicios sin afectar el funcionamiento de los demás. Esto asegura que el sistema pueda evolucionar fácilmente para adaptarse a nuevos requerimientos o integrar funcionalidades adicionales. Por otro lado, el desempeño del sistema se optimiza mediante el uso de bases de datos independientes para cada microservicio y una comunicación eficiente a través de APIs RESTful, lo que permite manejar grandes volúmenes de datos sin comprometer la velocidad ni la estabilidad.

Finalmente, la arquitectura también está preparada para enfrentar mayores demandas operativas en el futuro. La implementación de patrones de diseño como Circuit Breaker y Observer refuerza la resiliencia del sistema, permitiendo una recuperación rápida ante fallos y mejorando la capacidad de sincronización entre componentes. Este enfoque asegura que EcoVuelta no solo cumpla con los requisitos actuales, sino que también esté lista para adaptarse a las necesidades de un entorno logístico en constante evolución.

Esta combinación de una arquitectura basada en microservicios y patrones de diseño probados asegura que el backend de EcoVuelta no solo cumpla con los requisitos técnicos y operativos actuales, sino que también esté preparado para adaptarse y escalar según las necesidades futuras

### 2.3 Tecnologías y frameworks utilizados

El desarrollo del backend de la plataforma EcoVuelta emplea una variedad de tecnologías, frameworks y herramientas seleccionadas estratégicamente para garantizar un sistema robusto, escalable y seguro. La arquitectura del backend está basada en microservicios, lo que permite dividir el sistema en componentes independientes con responsabilidades bien definidas, facilitando el mantenimiento y la evolución futura.

El backend está compuesto por cuatro microservicios principales. Tres de ellos están implementados en Spring Boot, un framework basado en Java ampliamente utilizado en aplicaciones empresariales por su capacidad para manejar sistemas complejos de manera eficiente. Spring Boot se utiliza en los microservicios de autenticación, gestión de solicitudes y gestión de camiones y conductores, ya que integra herramientas avanzadas como Spring Security para garantizar un control de acceso robusto y gestionar roles y permisos de usuarios.

El cuarto microservicio, encargado de la planificación de rutas, está desarrollado en FastAPI, un framework ligero de Python que destaca por su rendimiento y facilidad de uso. FastAPI es ideal para interactuar con el modelo de optimización, permitiendo una integración eficiente y simplificada con el motor de cálculo de rutas. La base de datos seleccionada para el proyecto es MongoDB, un sistema de base de datos NoSQL conocido por su capacidad para manejar grandes volúmenes de datos no estructurados y dinámicos. Cada microservicio tiene su propia base de datos, lo que asegura independencia y escalabilidad. En particular, el microservicio de gestión de camiones y conductores utiliza una base de datos compartida con colecciones separadas, optimizando la gestión de estos recursos logísticos.

La seguridad es un aspecto fundamental en el diseño del backend. Se implementa un sistema de autenticación basado en JWT (JSON Web Tokens), que permite la verificación segura de las credenciales en cada solicitud. Spring Security refuerza esta funcionalidad al gestionar los roles y permisos de los usuarios en los microservicios basados en Spring Boot. Además, se emplean técnicas estándar de cifrado para proteger los datos en tránsito y prevenir vulnerabilidades comunes como ataques de inyección SQL (aunque no aplica directamente a MongoDB) o ataques XSS.

La comunicación entre los microservicios y el frontend, así como entre los propios microservicios, se realiza mediante APIs RESTful, lo que asegura una interacción eficiente y escalable. En el caso de los microservicios desarrollados en Spring Boot, se utiliza RestTemplate o WebClient para gestionar las solicitudes HTTP, mientras que FastAPI ofrece soporte nativo para el manejo de APIs RESTful.

Para garantizar la calidad del sistema, se utilizan herramientas como Postman y JMeter para realizar pruebas de endpoints. Postman se emplea para verificar manualmente la funcionalidad de las APIs, mientras que JMeter permite realizar pruebas de carga y evaluar el rendimiento del sistema bajo diferentes escenarios de uso. Aunque no se implementan pruebas unitarias tradicionales, estas herramientas aseguran que las interacciones entre los componentes del sistema sean fiables y cumplan con los requisitos funcionales.
El backend está desplegado en contenedores Docker, lo que garantiza portabilidad y consistencia entre los entornos de desarrollo, prueba y producción. Esta decisión técnica permite que cada microservicio opere de manera aislada, simplificando la gestión de dependencias y recursos. Aunque no se utiliza actualmente un

orquestador como Kubernetes, este podría considerarse en futuras iteraciones para manejar la escalabilidad dinámica.

La arquitectura basada en microservicios, junto con la implementación de APIs RESTful y herramientas modernas como Docker, destaca como una de las decisiones más importantes en este proyecto. Estas tecnologías permiten que los diferentes componentes del sistema evolucionen de manera independiente, se adapten a las necesidades cambiantes del negocio y aseguren un rendimiento óptimo. Además, el uso de herramientas de seguridad avanzadas y mecanismos de pruebas garantiza que el sistema cumpla con altos estándares de confiabilidad y robustez, preparando la plataforma EcoVuelta para futuras demandas y expansiones.

# 3 Diseño e Implementación

El presente capítulo aborda el diseño e implementación del sistema EcoVuelta, detallando las decisiones clave en la arquitectura del software y la estructura de sus componentes. El enfoque propuesto utiliza una arquitectura basada en microservicios, lo que permite dividir el sistema en módulos independientes y especializados. Esta metodología no solo facilita el desarrollo y mantenimiento, sino que también proporciona flexibilidad para escalar cada componente de acuerdo con las necesidades específicas de la plataforma.

## 3.1 Diseño de Componentes

El diseño de los componentes del sistema EcoVuelta está orientado a garantizar la modularidad, independencia y escalabilidad de cada funcionalidad clave de la plataforma. Siguiendo los principios de una arquitectura basada en microservicios, cada componente representa un dominio específico del negocio, lo que permite que se desarrollen, despliegen y mantengan de forma independiente. Estos componentes, aunque autónomos, trabajan en conjunto mediante APIs RESTful para garantizar una experiencia fluida e integrada para los usuarios. Los microservicios definidos abarcan áreas críticas como autenticación, gestión de solicitudes, planificación de rutas, y administración de camiones y conductores, y son orquestados mediante un API Gateway que centraliza las peticiones y asegura la comunicación eficiente entre el frontend y los microservicios backend. Además, el uso de herramientas modernas como Spring Boot, FastAPI y bases de datos NoSQL asegura que cada componente esté optimizado para cumplir su propósito con alto rendimiento y fiabilidad.

### 3.1.1 Arquitectura y estructura de los componentes

La arquitectura seleccionada para el sistema EcoVuelta es de microservicios, una solución que permite abordar la complejidad del proyecto mediante la división en servicios independientes. Cada microservicio representa un dominio específico y maneja una funcionalidad bien definida del sistema, lo que facilita su desarrollo, escalabilidad y mantenimiento. La implementación se basa en una combinación de tecnologías como Spring Boot para los microservicios de backend principales, FastAPI para procesos de cálculo más ligeros y eficientes, y MongoDB como base de datos NoSQL para garantizar flexibilidad y rendimiento. A continuación, se presenta la estructura de los componentes principales del sistema.

1. **API Gateway**: Este componente actúa como punto de entrada único para todas las solicitudes del frontend y es responsable de enrutar las peticiones hacia los microservicios correspondientes. También se encarga de gestionar la autenticación inicial mediante el token JWT generado por el microservicio de autenticación, asegurando que todas las solicitudes estén protegidas y validadas.
2. **Microservicio de autenticación**: Implementado en Spring Boot, este servicio gestiona el inicio de sesión y la generación de tokens JWT. Este token se utiliza para autenticar a los usuarios en cada interacción con otros microservicios, permitiendo validar sus permisos y roles. Además, este microservicio se conecta a una base de datos MongoDB donde se almacena la información de usuarios y credenciales de forma segura.
3. **Microservicio de gestión de solicitudes**: También desarrollado en Spring Boot, este componente centraliza la gestión de las solicitudes de transporte generadas por los usuarios, también, este servicio almacena información de ubicaciones (locations), como centros de abastecimiento o bodegas, asociadas a los trayectos. Esto permite una gestión más eficiente de los puntos de origen y destino en las solicitudes. Incluye funcionalidades para crear, almacenar y consultar solicitudes, así como para preparar los datos que serán enviados al servicio de planificación de rutas. Este microservicio interactúa directamente con el API Gateway y almacena las solicitudes en una base de datos MongoDB.

4. **Microservicio de planificación de rutas** Este componente está desarrollado en FastAPI, elegido por su alto rendimiento y facilidad de uso para tareas computacionales específicas. Su objetivo principal es procesar las solicitudes enviadas desde el microservicio de gestión de solicitudes y calcular rutas óptimas considerando factores como distancia, capacidad, costos y restricciones de certificaciones. También está diseñado para integrarse con APIs externas (por ejemplo, servicios de mapas o tráfico en tiempo real) para enriquecer sus resultados.
5. **Microservicio de camiones y conductores**: Utilizando Spring Boot, este servicio gestiona la información de los vehículos y conductores disponibles, incluyendo sus capacidades, certificaciones y disponibilidad. Proporciona datos clave al microservicio de planificación de rutas para optimizar la asignación de recursos. El microservicio utiliza una base de datos MongoDB compartida, donde los datos de camiones y conductores se organizan en colecciones separadas. Esto permite centralizar la gestión de los recursos logísticos dentro de un único servicio.

Esta arquitectura distribuida se refleja en la ilustración presentada, donde se puede observar la relación entre los microservicios, el API Gateway y las bases de datos. La ilustración muestra cómo los diferentes componentes del sistema interactúan para garantizar un flujo de trabajo eficiente y seguro.

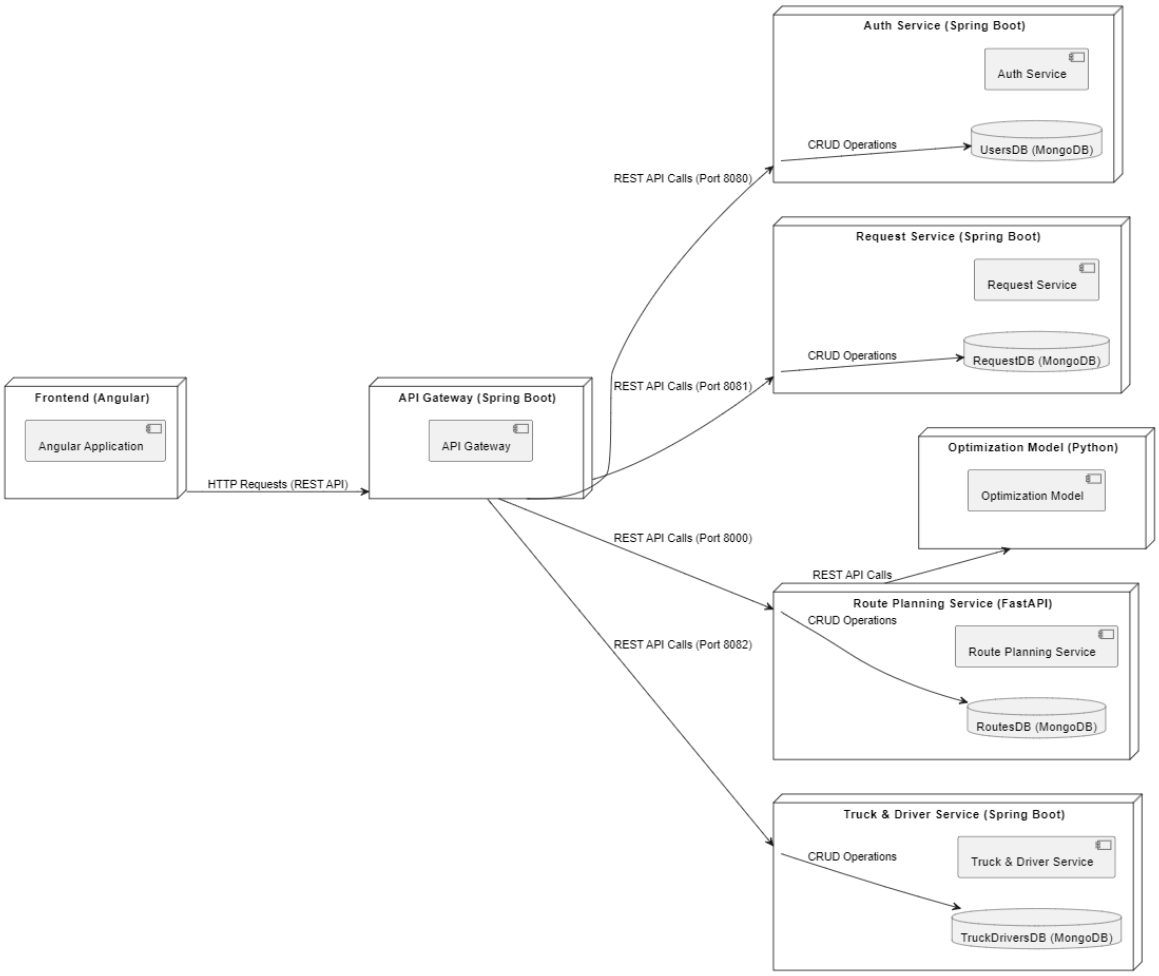


**Ilustración 3.1 Diagrama de despliegue de la arquitectura del sistema EcoVuelta**

La elección de la arquitectura basada en microservicios para EcoVuelta no solo responde a la necesidad de manejar la complejidad del sistema, sino que también ofrece ventajas significativas que aseguran la adaptabilidad del proyecto en el tiempo. Esta decisión permite que cada microservicio, al estar desacoplado, evolucione de forma independiente, facilitando su mantenimiento y actualización sin afectar la funcionalidad

general. Además, esta modularidad fomenta una mejor organización del equipo de desarrollo, ya que los microservicios pueden ser trabajados por equipos distintos, lo que reduce la dependencia entre tareas y mejora la eficiencia.

Otro aspecto clave es la capacidad de escalar los microservicios de manera independiente. En un sistema como EcoVuelta, donde ciertos servicios (como la planificación de rutas) pueden experimentar picos de demanda, esta arquitectura permite asignar más recursos únicamente a los componentes críticos, sin necesidad de escalar el sistema completo. Esto optimiza el uso de la infraestructura y reduce costos operativos, asegurando un rendimiento estable incluso en momentos de alta carga.

La elección de tecnologías también juega un papel fundamental en esta arquitectura. Spring Boot y FastAPI fueron seleccionados estratégicamente para aprovechar sus respectivas fortalezas. Spring Boot proporciona una base sólida y escalable para los microservicios principales, mientras que FastAPI, con su enfoque en la rapidez y eficiencia, resulta ideal para las operaciones computacionales intensivas, como el cálculo de rutas óptimas. Asimismo, el uso de MongoDB como base de datos NoSQL ofrece la flexibilidad necesaria para manejar datos no estructurados y permite un acceso eficiente a la información en tiempo real.

La resiliencia del sistema es otro de los beneficios importantes de esta arquitectura. En caso de que un microservicio falle, el resto del sistema puede continuar operando sin interrupciones significativas, garantizando una alta disponibilidad. Esto es posible gracias a la independencia de los servicios y a la centralización de la comunicación a través del API Gateway, que actúa como un intermediario robusto y confiable.

En términos de comunicación, los microservicios se comunican mediante llamadas REST API, con el API Gateway como único punto de entrada. Este diseño no solo simplifica la integración del frontend, sino que también asegura que las interacciones entre servicios sean seguras y estandarizadas. La autenticación y autorización se gestionan mediante tokens JWT, que no solo protegen el acceso al sistema, sino que también simplifican la validación entre servicios internos.

Finalmente, esta arquitectura proporciona una base sólida para la expansión futura del sistema. Si en algún momento es necesario integrar nuevas funcionalidades, como la incorporación de algoritmos avanzados para optimizar rutas o la conexión con proveedores externos de datos logísticos, estos pueden añadirse como nuevos microservicios sin alterar significativamente la estructura actual.
El despliegue de la arquitectura de EcoVuelta se realizará utilizando un enfoque modular y escalable basado en contenedores Docker. Cada microservicio será empaquetado de forma independiente, lo que garantiza portabilidad entre entornos de desarrollo, prueba y producción. Estos contenedores permitirán que los servicios operen de manera aislada, reduciendo conflictos entre dependencias y simplificando la administración del sistema.

La arquitectura está diseñada para maximizar la independencia de los componentes, asegurando que cada microservicio pueda escalar de manera autónoma. Por ejemplo, el microservicio de planificación de rutas, que tiene una carga computacional significativa, puede ejecutarse en múltiples instancias para manejar picos de demanda, mientras que servicios menos intensivos, como el de autenticación, pueden operar con menos recursos, optimizando así el uso de la infraestructura.

En cuanto a la comunicación, los microservicios interactuarán mediante llamadas REST API, con el API Gateway como intermediario central. Este componente actúa como un punto de entrada único para todas las solicitudes externas, validando cada una de ellas mediante autenticación con tokens JWT antes de enrutarla al microservicio correspondiente. Esto asegura que todas las interacciones sean seguras y que los servicios estén protegidos contra accesos no autorizados.

Los tokens JWT también son esenciales para la autenticación y autorización en las comunicaciones internas entre microservicios. Por ejemplo, cuando el microservicio de gestión de solicitudes envía datos al de

planificación de rutas, el token JWT acompaña la solicitud para validar que esta proviene de un servicio autorizado. Este mecanismo de seguridad garantiza la integridad de los datos y protege el sistema de posibles amenazas.

Además, el sistema implementará estrategias para manejar fallos en la comunicación entre microservicios. Por ejemplo, el patrón Circuit Breaker permitirá detectar y gestionar problemas en las conexiones, evitando que los fallos de un servicio afecten al resto del sistema. En caso de detectar fallos repetidos en la comunicación con un microservicio, el patrón Circuit Breaker devuelve respuestas predefinidas, garantizando la estabilidad del sistema mientras se resuelve el fallo. Esto se complementará con un monitoreo constante de las interacciones, asegurando que las conexiones sean estables y el sistema esté preparado para responder a escenarios inesperados.

En cuanto a la comunicación, los microservicios interactuarán mediante llamadas REST API, con el API Gateway como intermediario central. Este componente actúa como un punto de entrada único para todas las solicitudes externas, validando cada una de ellas mediante autenticación con tokens JWT antes de enrutarla al microservicio correspondiente. Esto asegura que todas las interacciones sean seguras y que los servicios estén protegidos contra accesos no autorizados.

Por ejemplo, cuando un usuario realiza una solicitud de creación de transporte, esta es enviada desde el frontend al API Gateway, que valida el token JWT y luego dirige la solicitud al Request Service, donde se almacenan los datos y se actualiza el estado de la solicitud. Posteriormente, el Request Service interactúa con el Route Planning Service, que calcula las rutas óptimas y devuelve los resultados. Una vez que la ruta está optimizada, el sistema asigna automáticamente los recursos necesarios mediante el Truck y Driver Service, conectando los datos de camiones y conductores para cumplir con la solicitud del usuario. Este flujo eficiente y seguro asegura que todos los componentes trabajen de manera coordinada, maximizando la funcionalidad del sistema

Finalmente, cada microservicio contará con su propia base de datos MongoDB, diseñada para optimizar el acceso a los datos según las necesidades de cada servicio. Por ejemplo, el microservicio de gestión de solicitudes manejará tanto las solicitudes como las ubicaciones en colecciones separadas dentro de su base de datos, mientras que el microservicio de camiones y conductores utilizará una base de datos compartida con colecciones independientes para cada entidad. Esta arquitectura de datos asegura que la información esté bien estructurada y que cada servicio tenga acceso eficiente a los recursos que necesita.

El diseño descrito se refleja en el diagrama presentado, donde se destacan las interacciones entre los componentes principales del sistema, el API Gateway y las bases de datos asociadas. Este enfoque asegura que el sistema sea eficiente, seguro y capaz de adaptarse a los requerimientos operativos de EcoVuelta.

### 3.1.2 Uso de buenas prácticas y patrones de diseño

En el diseño del backend de la plataforma EcoVuelta, se han implementado diversas buenas prácticas de desarrollo y patrones de diseño que aseguran un sistema eficiente, modular y sostenible. Estas decisiones no solo buscan cumplir con los requisitos actuales del proyecto, sino también garantizar la flexibilidad y escalabilidad necesarias para adaptarse a futuras demandas en el sector logístico.

Uno de los principios fundamentales aplicados es la separación de responsabilidades, donde cada microservicio está diseñado para cumplir un rol específico y bien definido dentro del sistema. Por ejemplo, el Auth Service se encarga exclusivamente de la autenticación y autorización de usuarios mediante la emisión y validación de tokens JWT, mientras que el Request Service centraliza la gestión de solicitudes de transporte y el almacenamiento de ubicaciones. Esta separación no solo facilita el mantenimiento y desarrollo del sistema, sino que también minimiza los errores al reducir el acoplamiento entre los diferentes microservicios.

La arquitectura del backend también se apoya en el uso de APIs RESTful, que siguen estándares ampliamente aceptados para la comunicación entre sistemas. Los endpoints están diseñados para ser claros y consistentes, utilizando métodos HTTP como GET, POST, PUT y DELETE según sea necesario. Este enfoque asegura que el frontend pueda interactuar con el backend de manera eficiente y que los microservicios se comuniquen de forma fluida y segura.

En términos de manejo de errores, se ha implementado un enfoque centralizado mediante clases específicas para gestionar excepciones. Esto permite que los errores sean capturados y procesados de manera uniforme en toda la aplicación, proporcionando mensajes claros y útiles tanto para los desarrolladores como para los usuarios finales. Este manejo uniforme mejora la experiencia del cliente al garantizar que las respuestas del sistema sean coherentes y predecibles.

Además, se han adoptado patrones de diseño esenciales que fortalecen la arquitectura del backend:

- **Patrón Singleton**: Este patrón, aunque implícito gracias al manejo de dependencias del framework Spring Boot, asegura que ciertos componentes críticos como servicios y repositorios tengan una única instancia operativa durante toda la ejecución de la aplicación. Esto optimiza el uso de recursos y garantiza la consistencia en las operaciones.
- **Patrón Repository**: Utilizado para gestionar el acceso a la base de datos, este patrón abstrae las operaciones de persistencia y permite que la lógica de negocio permanezca separada de las tareas de almacenamiento y consulta de datos. Esta separación facilita el mantenimiento del código y permite realizar cambios en la estructura de la base de datos sin afectar a otros componentes del sistema.
- **Patrón Circuit Breaker**: Este patrón asegura la resiliencia del sistema frente a fallos en la comunicación entre microservicios. Por ejemplo, en el caso de que el Route Planning Service no responda debido a una alta carga, el Circuit Breaker se activa, devolviendo respuestas predefinidas o interrumpiendo temporalmente las solicitudes para evitar una sobrecarga general.
- **Patrón Observer**: Aunque no está implementado actualmente, se considera para futuras expansiones. Este patrón permitiría que los componentes del sistema reaccionen automáticamente a los cambios en otros módulos. Por ejemplo, una actualización en la base de datos podría notificar a otros servicios relacionados, mejorando la sincronización y escalabilidad del sistema.

El diseño del backend también sigue buenas prácticas de desarrollo reconocidas, como:

- **Control de versiones**: El código fuente se gestiona mediante Git, lo que facilita la colaboración en equipo y el seguimiento de cambios. Cada iteración se valida y documenta para garantizar la trazabilidad.
- **Modularidad del código**: Se siguen principios como la separación de lógica de negocio, acceso a datos y controladores para asegurar que cada parte del sistema sea fácil de mantener y extender.

Estas prácticas y patrones de diseño no solo garantizan que el backend cumpla con los estándares actuales de calidad en desarrollo de software, sino que también preparan el sistema para escalar y adaptarse a futuros retos. A través de un diseño cuidadosamente planificado y decisiones técnicas alineadas con los objetivos del sistema, el backend de EcoVuelta está preparado para responder a las demandas dinámicas del sector logístico con eficacia y sostenibilidad.

### 3.1.3 Integración con el sistema general

El backend de la plataforma EcoVuelta se integra con otros componentes del sistema mediante un enfoque modular y estandarizado, utilizando APIs RESTful para la comunicación. Este método asegura un intercambio eficiente de información entre el frontend, los microservicios, las bases de datos y el modelo de optimización. Las llamadas y respuestas se estructuran en formato JSON, garantizando la interoperabilidad entre los componentes.

La arquitectura está diseñada para que cada microservicio cumpla un rol específico dentro del sistema, manteniendo el desacoplamiento y facilitando el mantenimiento. Los flujos de datos están coordinados para asegurar que las solicitudes de los usuarios sean procesadas de manera eficiente y segura. A continuación, se

describe el flujo de datos típico relacionado con el registro y la optimización de una solicitud de transporte, ilustrado con un diagrama de secuencia que detalla la interacción entre el frontend, el API Gateway, el Request Service y el Route Planning Service.

**Flujo de datos: Registro de una solicitud de transporte**

1. **Envió de la solicitud desde el frontend**: El usuario completa un formulario con los detalles de la solicitud (origen, destino, tipo de carga, capacidad requerida y fecha de inicio) y lo envía mediante una solicitud HTTP POST al endpoint /api/requests gestionado por el API Gateway.
2. **Procesamiento inicial en el Request Service**:
   - El API Gateway enruta la solicitud al Request Service, que valida los datos recibidos para asegurarse de que cumplan con las reglas de negocio (por ejemplo, que el origen y el destino sean válidos).
   - Una vez validados, los datos se almacenan en MongoDB utilizando el patrón Repository, asegurando la separación entre la lógica de negocio y las operaciones de persistencia.
3. **Interacción con el Route Planning Service**:
   - El Request Service envía los datos de la solicitud al Route Planning Service a través de una llamada RESTful, donde se procesan y se transmiten al modelo de optimización en Python.
   - El modelo genera una ruta óptima considerando factores como la distancia, la capacidad del camión y los costos operativos.
4. **Respuesta al Request Service y al usuario**:
   - Los resultados generados son devueltos al Request Service, que actualiza los datos de la solicitud en MongoDB.
   - Finalmente, el Request Service envía una respuesta al frontend confirmando el registro exitoso y proporcionando el estado actualizado de la solicitud.

A continuación, se presenta un diagrama de secuencia que ilustra este flujo de datos. Muestra cómo se integra el backend con el resto del sistema y cómo se comunican dos microservicios: el Request Service y el Route Planning Service.

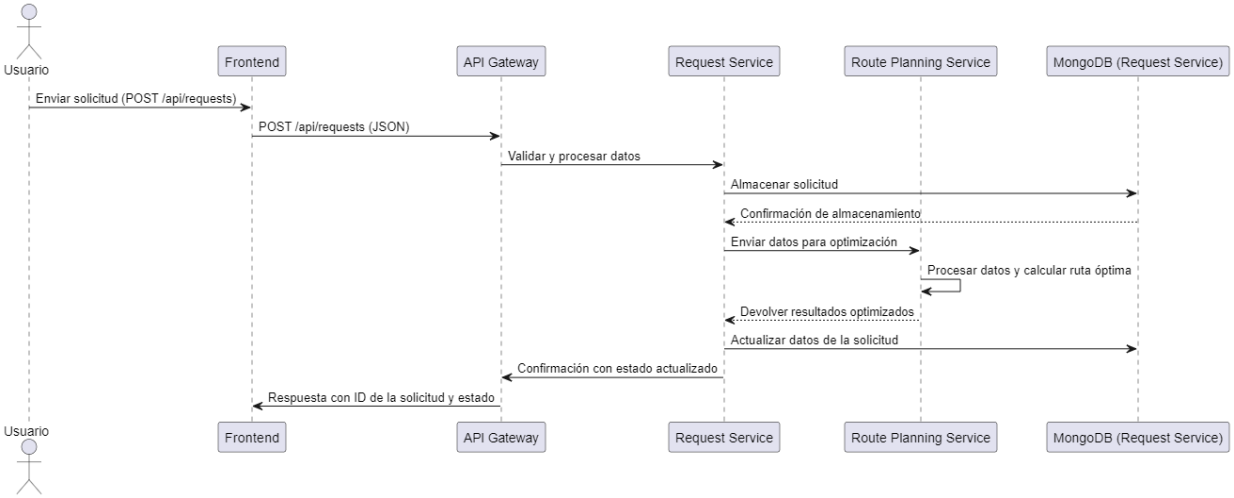


**Ilustración 3.2 Diagrama de Secuencia: Flujo de Creación y Optimización de Solicitudes de Transporte**

El sistema está diseñado para manejar errores comunes en cada etapa del flujo. Por ejemplo, si los datos enviados por el frontend son incompletos o inválidos, el API Gateway devuelve un código de error 400 (Bad Request) al usuario con un mensaje descriptivo.

En caso de un fallo en la comunicación entre el Request Service y el Route Planning Service, el patrón Circuit Breaker evita que se acumulen solicitudes y devuelve una respuesta predeterminada, asegurando la estabilidad del sistema.

Este flujo de datos, junto con las estrategias de validación y manejo de errores, demuestra cómo los diferentes componentes del sistema trabajan de manera coordinada para cumplir los objetivos de EcoVuelta. La integración modular asegura que los microservicios sean independientes y que las interacciones sean fluidas, escalables y seguras.

### 3.2 Detalles de la codificación y desarrollo

El desarrollo del backend de EcoVuelta fue llevado a cabo por un equipo de cuatro integrantes, cada uno con responsabilidades específicas: backend, frontend, modelo de optimización, y diseño arquitectónico. En este contexto, el backend participó en aproximadamente un tercio de las historias de usuario, contribuyendo tanto de manera completa como parcial. A continuación, se presenta un desglose detallado del proceso de desarrollo, las prácticas utilizadas y la intervención del backend en cada sprint.

El desarrollo se dividió en tres sprints, cada uno con una duración de dos semanas:

- **Sprint 1:** Establecer los cimientos del backend.
- **Sprint 2:** Desarrollar las funcionalidades principales.
- **Sprint 3:** Integración, validación y optimización del sistema.

Para cada sprint, el backend asumió tareas específicas, documentadas a continuación:

**Sprint 1: Configuración inicial y diseño básico del backend**
Fechas: 7 de octubre de 2024 - 20 de octubre de 2024
**Tareas del backend:**

- Configuración del proyecto en Spring Boot y FastAPI.
- Implementación del endpoint de autenticación inicial /auth/login.
- Configuración de MongoDB y creación de colecciones para usuarios y solicitudes.
- Validación de flujos básicos mediante Postman.

**Sprint 2: Desarrollo de funcionalidades principales**
Fechas: 21 de octubre de 2024 - 3 de noviembre de 2024
**Tareas del backend:**

- Desarrollo de CRUDs para usuarios, solicitudes, conductores, vehículos y ubicaciones.
- Implementación de filtros avanzados, como /drivers/getByCompany/{company}.
- Validación funcional mediante pruebas con Postman.
- Establecimiento de comunicación básica con el modelo de optimización a través del Route Planning Service.

**Sprint 3: Integración, validación y optimización**
Fechas: 4 de noviembre de 2024 - 15 de noviembre de 2024
**Tareas del backend:**

- Consolidación de la integración entre microservicios mediante REST API.
- Resolución de errores detectados en pruebas de carga con JMeter.
- Ajustes en endpoints para optimizar la comunicación entre sistemas.
- Validación final de las funcionalidades críticas.

El equipo utilizó Git como herramienta principal para el control de versiones, siguiendo una estrategia basada en GitFlow, alojando el repositorio en Bitbucket. Este enfoque estructurado permitió un manejo eficiente de ramas y flujos de trabajo:

- **Ramas de desarrollo:**
    - feature/{nombre-tarea}: Ramas para el desarrollo de cada funcionalidad o tarea específica.
    - develop: Rama principal para integrar los cambios en desarrollo.
    - main: Rama estable que almacena versiones funcionales listas para despliegue.

- **Prácticas adicionales en GitFlow:**
    - **Pull Requests:** Antes de fusionar una rama a develop, se requería un Pull Request revisado por al menos otro integrante.
    - **Tags para versiones:** Se utilizaron etiquetas con formato **vX.Y.Z**:
        - **X:** Cambios mayores que afectan significativamente el sistema.
        - **Y:** Nuevas funcionalidades que no rompen la compatibilidad.
        - **Z:** Correcciones de errores menores o ajustes específicos.

Esta estructura facilitó la colaboración en equipo, garantizando que cada funcionalidad estuviera bien documentada y que el sistema mantuviera estabilidad durante el desarrollo.

El backlog incluyó 18 historias de usuario, de las cuales 6 fueron implementadas completamente por el backend y 4 requirieron intervención parcial. Estas historias se documentaron en Jira y fueron organizadas por prioridades, como se detalla a continuación:

**Historias de usuario con intervención del backend**

1. **HU08:** *"Como usuario, quiero autenticarme para acceder al sistema."*
    - **Intervención:** Completa. Desarrollo del endpoint /auth/login y emisión de tokens JWT.
2. **HU14:** *"Como administrador, quiero asignar manualmente conductores y vehículos."*
    - **Intervención:** Completa. Endpoint /assignments/manual para asignaciones específicas.
3. **HU03:** *"Como usuario, quiero crear una solicitud de transporte."*
    - **Intervención:** Completa. Endpoint /requests con validación de reglas de negocio.
4. **HU17:** *"Como usuario, quiero cancelar una solicitud."*
    - **Intervención:** Parcial. Validación del estado y actualización en la base de datos.
5. **HU05:** *"Como administrador, quiero consultar la disponibilidad de recursos."*
    - **Intervención:** Parcial. Integración del endpoint /drivers/availability.
6. **HU12:** *"Como usuario, quiero ver un historial de mis solicitudes."*
    - **Intervención:** Parcial. Endpoint /requests/history.

El sistema EcoVuelta está diseñado para abordar la problemática de la logística y optimización de transporte mediante una solución modular y orientada a microservicios. Los actores principales que interactúan con el sistema representan roles específicos dentro del flujo de trabajo logístico. Estos actores, y las funciones que desempeñan en el sistema, se describen a continuación para proporcionar un contexto claro sobre cómo se organizan las interacciones:

- **Usuario Cliente**: Este tipo de usuario representa a las personas o empresas que generan solicitudes de transporte de carga. Estas solicitudes incluyen información detallada sobre las características de la carga, el origen, el destino y otras restricciones relevantes. Además, este usuario puede consultar el estado de sus solicitudes, cancelar transportes pendientes y gestionar su perfil de usuario.
- **Usuario Transportista**: Corresponde al dueño o administrador de vehículos que pueden ser asignados a las solicitudes de transporte. Este actor es responsable de revisar las solicitudes asignadas, gestionar la disponibilidad de recursos y elegir a cuál de sus camiones desea asignar una ruta evaluada previamente por el sistema.
- **Administrador**: Representa el rol de supervisión dentro del sistema. Este actor tiene permisos para gestionar usuarios, configurar roles, administrar las ubicaciones permitidas en el modelo de optimización.
- **Sistema de Optimización**: Si bien no es un usuario humano, este componente desempeña un rol crítico en el cálculo de rutas óptimas. Se comunica con el backend para recibir información sobre solicitudes y restricciones y devuelve resultados procesados que ayudan a optimizar el transporte.

A través de estos actores, el sistema integra diferentes funcionalidades que permiten una interacción fluida y eficiente. El diagrama de casos de uso que se presenta a continuación resume las interacciones más relevantes entre estos actores y los casos de uso cubiertos por el backend del sistema. Este diagrama no solo ilustra la relación entre los actores y las funcionalidades principales, sino que también destaca cómo cada microservicio participa en el cumplimiento de los objetivos del sistema.

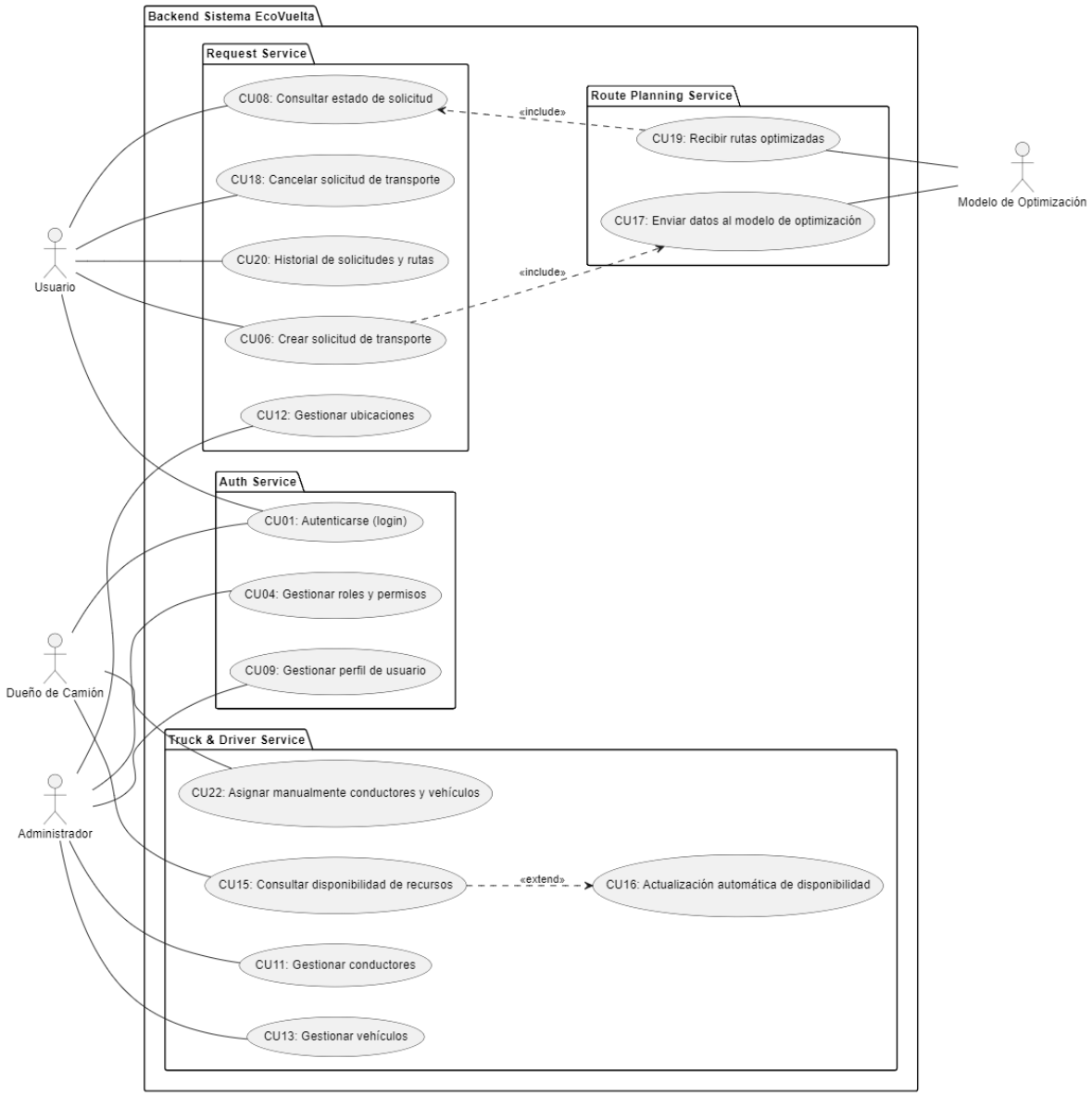


**Ilustración 3.3 Diagrama de casos de uso general**

## 3.3 Pruebas y Validación

Para garantizar que el backend de la plataforma EcoVuelta cumpla con los requerimientos planteados, se adoptó un enfoque de validación basado en pruebas manuales, utilizando herramientas específicas para evaluar tanto la funcionalidad de los endpoints como el rendimiento general del sistema distribuido. Este proceso se llevó a cabo considerando la arquitectura de microservicios implementada, donde cada servicio se probó de forma independiente y luego como parte del sistema integrado.

En primer lugar, se realizaron pruebas funcionales para verificar que los endpoints de cada microservicio cumplieran con los requisitos definidos en las historias de usuario. Estas pruebas se ejecutaron utilizando Postman, permitiendo simular solicitudes HTTP hacia servicios como el Auth Service, Request Service, Truck y Driver Service y Route Planning Service. Las respuestas se validaron asegurando que los datos enviados y

recibidos fueran consistentes con los formatos y las reglas de negocio definidos en el diseño de las APIs RESTful.

Adicionalmente, se llevaron a cabo pruebas de rendimiento utilizando JMeter, enfocadas en evaluar cómo cada microservicio responde bajo diferentes niveles de carga. Estas pruebas incluyeron escenarios críticos, como la creación de solicitudes de transporte, la asignación de recursos y la optimización de rutas. En particular, se midió la capacidad del sistema para manejar solicitudes concurrentes y se analizaron las latencias en las comunicaciones entre microservicios a través del API Gateway. Este enfoque permitió identificar cuellos de botella y ajustar configuraciones en el sistema, como los límites de concurrencia y tiempos de espera.

El manejo de errores también fue un componente esencial en las pruebas. Cada microservicio fue evaluado para asegurar que respondiera con mensajes claros y consistentes en escenarios como solicitudes malformadas, fallos de validación o problemas de comunicación entre servicios. Por ejemplo, se validó que, en caso de que el Route Planning Service no pudiera devolver una ruta optimizada, el sistema devolviera un mensaje adecuado al usuario sin afectar el funcionamiento general del sistema.

Aunque en esta fase del proyecto no se implementaron pruebas unitarias automatizadas debido a restricciones de tiempo y alcance, se considera su inclusión en futuras iteraciones del desarrollo. Estas pruebas permitirían validar individualmente las funciones internas de cada microservicio, facilitando la detección temprana de errores y reduciendo la dependencia de pruebas manuales extensas.

Finalmente, cada nueva funcionalidad fue revisada en un entorno de desarrollo aislado antes de su integración al sistema principal. Este proceso incluyó pruebas de comunicación entre los microservicios a través del API Gateway, garantizando que los cambios en un servicio no interfirieran con otros. Este enfoque modular permitió mantener la integridad del sistema y la calidad en cada entrega.

El enfoque combinado de pruebas funcionales, validación del manejo de errores y pruebas de rendimiento asegura que el backend de EcoVuelta no solo cumpla con los requerimientos actuales, sino que también esté preparado para evolucionar de manera robusta y escalable en el futuro.

### 3.3.1 Estrategias de testing aplicadas a los componentes

Para garantizar la calidad y el correcto funcionamiento del backend de la plataforma EcoVuelta, se implementaron diversas estrategias de testing, adaptadas a la arquitectura de microservicios y orientadas a cubrir aspectos funcionales, de integración y de rendimiento. Estas estrategias permitieron validar tanto el comportamiento individual de cada microservicio como la interacción entre ellos, asegurando que el sistema respondiera de manera eficiente y confiable bajo diferentes escenarios operativos.

Las estrategias implementadas incluyen:
Se llevaron a cabo pruebas funcionales manuales utilizando Postman para validar que los endpoints de cada microservicio cumplieran con los requisitos establecidos. Estas pruebas incluyeron la verificación de las respuestas HTTP (códigos de estado como 200, 400, 401, 404, etc.) y la estructura de las respuestas en formato JSON, asegurando la consistencia con los contratos definidos en el diseño de las APIs RESTful. Se probaron escenarios positivos y negativos, como el envío de datos correctos y casos con datos faltantes o en formatos inválidos. Por ejemplo, en el Request Service, se validó que las solicitudes de transporte con datos incompletos devolvieran mensajes de error claros y específicos.

Mediante JMeter, se evaluó el rendimiento del sistema bajo diferentes niveles de carga, analizando el tiempo de respuesta promedio de los endpoints y la capacidad para manejar solicitudes concurrentes. Estas pruebas incluyeron el procesamiento de múltiples solicitudes de transporte en el Request Service, la validación de rutas optimizadas en el Route Planning Service, y la gestión de recursos en el Truck y Driver Service. Los

resultados permitieron identificar y optimizar puntos críticos, como operaciones con la base de datos y latencias en las comunicaciones entre microservicios.

Se probaron diversos escenarios para asegurar que cada microservicio pudiera manejar errores de manera adecuada. Esto incluyó la validación de respuestas claras y consistentes en casos de solicitudes malformadas, fallos en la conexión con la base de datos o problemas de comunicación entre microservicios. Por ejemplo, si el Route Planning Service no podía devolver una ruta optimizada, se verificó que el sistema informara al usuario de manera comprensible y segura, sin exponer detalles técnicos internos.

Dada la naturaleza distribuida del sistema, se realizaron pruebas de integración para validar la comunicación entre los microservicios a través del API Gateway. Estas pruebas incluyeron la validación del flujo de datos entre el Request Service y el Route Planning Service, asegurando que los datos enviados para optimización fueran correctamente procesados y los resultados almacenados en la base de datos. Además, se probó la interacción entre el Truck y Driver Service y el Request Service para garantizar la correcta asignación de recursos a las solicitudes.

Cada prueba realizada fue documentada detalladamente, registrando los datos de entrada, los resultados esperados y los comportamientos observados. Este enfoque permitió identificar patrones de errores y realizar ajustes específicos en los microservicios afectados. Los resultados también se utilizaron como base para futuras mejoras y la planificación de pruebas automatizadas.

Si bien en esta fase no se implementaron pruebas unitarias automatizadas, estas estrategias manuales, combinadas con herramientas robustas como Postman y JMeter, aseguraron que los componentes clave del backend cumplieran con los estándares de calidad requeridos para operar de manera eficiente y satisfacer las necesidades de los usuarios de EcoVuelta.

### 3.3.2 Resolución de errores y optimización

Durante el desarrollo del backend de la plataforma EcoVuelta, se implementaron estrategias estructuradas para identificar y resolver errores (bugs) y optimizar el rendimiento del sistema. Estas acciones fueron esenciales para garantizar que el sistema cumpliera con los requerimientos establecidos y pudiera operar de manera eficiente bajo diferentes condiciones.

La resolución de bugs se realizó de manera iterativa, combinando pruebas funcionales y de rendimiento con un monitoreo constante de la interacción entre los microservicios. Los errores más comunes se presentaron durante la comunicación entre microservicios, la validación de datos, y las operaciones relacionadas con la base de datos. Una vez identificado un bug, se reproducía el escenario en un entorno de desarrollo controlado, lo que permitió analizar las condiciones específicas que originaron el problema. Este análisis se complementó con herramientas como Postman para verificar los datos enviados y recibidos, y JMeter para medir el impacto en el rendimiento.

Tras identificar la causa raíz de un bug, el equipo implementaba las correcciones necesarias en el microservicio correspondiente. Estas correcciones se documentaban en el repositorio de control de versiones, asegurando que cada cambio quedara registrado con una descripción detallada del problema resuelto. Posteriormente, se ejecutaban pruebas adicionales para validar la solución, garantizando que el bug había sido eliminado sin introducir nuevos problemas. Este proceso incluyó volver a ejecutar los casos de prueba que originalmente detectaron el error y realizar pruebas de integración para validar que la comunicación entre los microservicios seguía funcionando correctamente.

En paralelo, se realizaron ajustes para optimizar el rendimiento del sistema, enfocados en tres áreas principales:

1. **Optimización de consultas a la base de datos:** Se revisaron y optimizaron las operaciones más frecuentes en MongoDB, aplicando índices en campos clave y ajustando la estructura de las

colecciones para mejorar los tiempos de respuesta. Por ejemplo, en el Request Service, se optimizaron las consultas relacionadas con el historial de solicitudes y la búsqueda de ubicaciones.

2. **Reducción de latencia en la comunicación entre microservicios:** Se analizaron los tiempos de respuesta en las solicitudes enviadas entre el Request Service, el Route Planning Service y el Truck y Driver Service. Esto llevó a la optimización del formato de datos transmitidos y la implementación de tiempos de espera adecuados para minimizar retrasos.
3. **Mejoras en el manejo de excepciones:** Cada microservicio implementó controladores de excepciones específicos para capturar errores de manera precisa y devolver mensajes claros y consistentes. Este ajuste no solo mejoró la experiencia del usuario, sino que también facilitó la depuración de problemas al proporcionar información detallada sobre los errores.

Todos los cambios realizados en el backend, tanto en la resolución de bugs como en la optimización, fueron registrados en el repositorio de control de versiones, utilizando un esquema de ramas estructurado para garantizar que las modificaciones fueran validadas antes de integrarse al sistema principal. Este enfoque aseguraba la estabilidad del sistema y mantenía un historial claro y accesible de las mejoras realizadas.

Gracias a este proceso, el backend de EcoVuelta tiene excelentes niveles de estabilidad, rendimiento y confiabilidad, asegurando su capacidad para manejar las operaciones críticas de la plataforma de manera eficiente.

### 3.4 Integración con otros componentes

El backend de la plataforma EcoVuelta está diseñado para facilitar una integración eficiente con otros sistemas y componentes externos, garantizando un ecosistema flexible y conectado que responde a las necesidades operativas y funcionales del proyecto. Gracias a la arquitectura basada en microservicios y al uso de APIs RESTful, el sistema permite una comunicación fluida entre sus módulos internos y con elementos externos, asegurando la interoperabilidad y escalabilidad del sistema.

Una de las principales integraciones implementadas es con un modelo de optimización desarrollado en Python. Este modelo interactúa directamente con el Route Planning Service para recibir datos relacionados con solicitudes de transporte, realizar cálculos avanzados y devolver rutas optimizadas. Los resultados obtenidos se almacenan en la base de datos asociada al Request Service, lo que permite que estos estén disponibles para los usuarios finales a través de consultas realizadas en tiempo real. Este flujo es esencial para optimizar recursos logísticos y garantizar que cada solicitud sea gestionada de manera eficiente.

Adicionalmente, aunque el sistema no incluye un módulo de notificaciones en su versión actual, se ha considerado la integración futura con servicios externos como Twilio para enviar alertas y actualizaciones a los usuarios y transportistas mediante WhatsApp o mensajes SMS. Estas funcionalidades podrían mejorar la experiencia del usuario y fortalecer la comunicación dentro del ecosistema logístico.

En el ámbito de la gestión de pagos, esta funcionalidad no forma parte del alcance actual del proyecto. Sin embargo, el diseño modular y desacoplado del backend permite que futuros desarrollos incluyan un sistema de pagos sin afectar las funcionalidades existentes. Por ejemplo, una posible integración con servicios como WebPay podría implementarse como un módulo independiente que interactúe con las APIs principales, asegurando la compatibilidad y la expansión futura de la plataforma.

El diseño del sistema también se apoya en principios como la separación de responsabilidades y el uso de patrones de diseño como Repository, que aseguran que cada componente funcione de manera independiente pero compatible. Esto facilita la incorporación de nuevas funcionalidades o servicios externos y minimiza el impacto de los cambios en el resto del sistema.

A continuación, se presentan diagramas de secuencia detallados que ilustran los principales flujos de comunicación entre los microservicios y los componentes externos, como el modelo de optimización. Estos

diagramas destacan cómo el sistema maneja procesos críticos, como la creación de solicitudes de transporte, la asignación de recursos y la optimización de rutas.
El siguiente diagrama de secuencia muestra el flujo completo de la creación y procesamiento de una solicitud de transporte en la plataforma EcoVuelta. Este proceso involucra al usuario, el frontend, el API Gateway y los microservicios responsables de la gestión de solicitudes y la planificación de rutas. El objetivo principal es registrar la solicitud, calcular una ruta optimizada utilizando el modelo de optimización, y devolver al usuario los detalles procesados de la operación. Este flujo destaca la interacción eficiente entre los componentes del sistema y la base de datos.

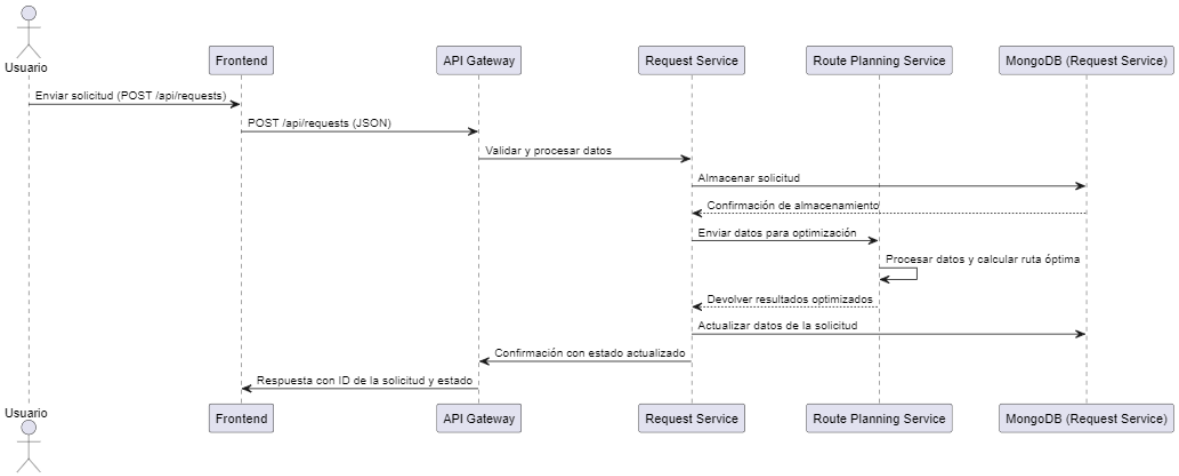


**Ilustración 3.4 Diagrama de Secuencia: Proceso Completo de Creación y Optimización de Solicitudes**

El siguiente diagrama de secuencia describe el flujo de asignación manual de un conductor y un vehículo a una solicitud de transporte. Este proceso es iniciado por el usuario que posee camiones disponibles (Dueño de Camión), quien selecciona un conductor y un vehículo específicos para cumplir con la solicitud previamente creada. El flujo asegura que la información se actualice correctamente en la base de datos y que el sistema mantenga la integridad de los datos entre los microservicios involucrados.

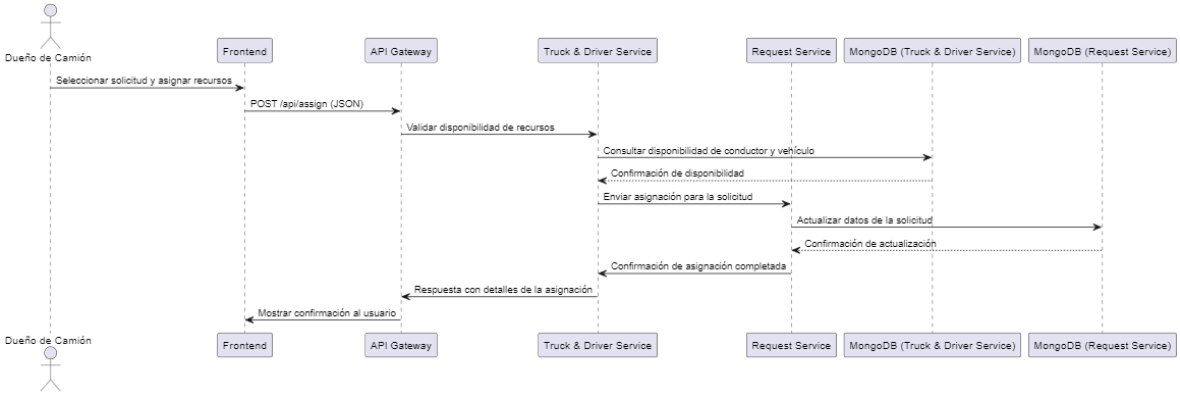


**Ilustración 3.5 Diagrama de Secuencia: Asignación Manual de Conductores y Vehículos**

El siguiente diagrama de secuencia ilustra el proceso mediante el cual un usuario consulta el historial de solicitudes de transporte y las rutas asociadas. Este flujo permite a los usuarios obtener información detallada de las operaciones realizadas, consultando datos almacenados en los microservicios responsables de las solicitudes y la planificación de rutas. El proceso garantiza una respuesta rápida y precisa, asegurando que los datos relevantes se presenten al usuario de manera estructurada.

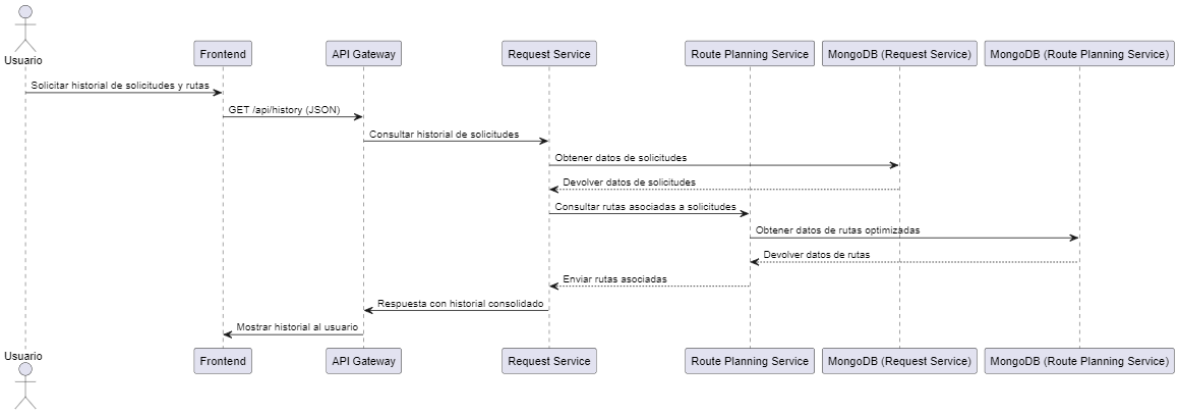


**Ilustración 3.6 Diagrama de Secuencia: Consulta del Historial de Solicitudes y Rutas**

# 4 Conclusiones

El desarrollo del backend para la plataforma EcoVuelta representó un desafío técnico y organizativo significativo, que dejó lecciones valiosas tanto en la disciplina informática como en el sector logístico. A lo largo de este proyecto, no solo se adquirieron conocimientos técnicos avanzados, sino también habilidades clave en la gestión de equipos y toma de decisiones en contextos complejos.

Uno de los aprendizajes más importantes fue la correcta implementación y gestión de una arquitectura basada en microservicios. Inicialmente, el equipo debatió entre mantener una arquitectura monolítica o adoptar microservicios. La decisión de optar por microservicios demostró ser acertada, ya que permitió una mayor flexibilidad, escalabilidad y la posibilidad de trabajar de manera independiente en diferentes módulos. Sin embargo, si se volviera a abordar este proyecto, se implementarían desde el inicio herramientas de automatización como Docker y CI/CD para facilitar el despliegue y la integración, algo que se consideró más adelante pero que habría ahorrado tiempo y esfuerzo.

Entre las decisiones que ahora se tomarían de forma distinta, destaca la estructura de las bases de datos. Originalmente, se planteó una base de datos única con colecciones compartidas entre microservicios. Sin embargo, en la práctica, esto generó desafíos relacionados con la sincronización y la separación de responsabilidades. Dividir las bases de datos por microservicio mejoró significativamente el rendimiento y la claridad del sistema, pero esta decisión podría haberse implementado desde el inicio para evitar reprocesos. Los aspectos más desafiantes del proyecto fueron la integración de los diferentes componentes y la gestión de la comunicación entre microservicios a través del API Gateway. Garantizar que los datos fluyeran correctamente y que las solicitudes fueran procesadas sin latencias significativas requirió ajustes constantes y pruebas rigurosas. También, la interacción con el modelo de optimización representó un reto técnico, ya que implicaba trabajar con un sistema ajeno al backend, asegurando que las respuestas fueran precisas y oportunas.

A pesar de estos desafíos, los logros del proyecto son significativos. Se desarrolló un sistema robusto que aborda un problema real en la logística: la reducción de los viajes vacíos en camiones. Esto no solo mejora la rentabilidad de las operaciones, sino que también tiene un impacto positivo en el medio ambiente al reducir las emisiones de carbono asociadas con el transporte. La integración exitosa de todas las herramientas usadas para implementar el proyecto demostró la capacidad del equipo para trabajar con tecnologías diversas y alcanzar un objetivo común.

El proyecto EcoVuelta no solo aporta soluciones a problemas logísticos, sino que también contribuye a la disciplina informática al demostrar cómo las arquitecturas modernas y las buenas prácticas de desarrollo pueden aplicarse para resolver problemas del mundo real. El uso de patrones de diseño, la separación de responsabilidades y el manejo adecuado de las bases de datos son ejemplos claros de cómo este proyecto incorpora principios fundamentales de la ingeniería de software.

En términos de impacto, EcoVuelta tiene el potencial de transformar la forma en que las empresas de transporte gestionan sus operaciones. Los usuarios y clientes directos se benefician de un sistema más eficiente, que reduce costos y mejora la transparencia en las operaciones. Además, la flexibilidad del diseño permite futuras integraciones, como sistemas de notificaciones en tiempo real y módulos de pago, que aumentarían aún más el valor de la plataforma.

Mirando hacia el futuro, existen múltiples aspectos que podrían mejorarse o expandirse. Por ejemplo, la implementación de notificaciones en tiempo real mediante servicios como Twilio sería un gran avance para la comunicación con los usuarios. También, la integración de un sistema de pagos permitiría a EcoVuelta manejar transacciones de manera segura y eficiente. Desde una perspectiva técnica, la incorporación de pruebas automatizadas y un pipeline de CI/CD optimizarían el desarrollo y el mantenimiento del sistema a largo plazo. En resumen, EcoVuelta es un proyecto que combina innovación tecnológica con un enfoque práctico en la resolución de problemas reales. Las lecciones aprendidas y los logros alcanzados en este proceso no solo

enriquecen la experiencia del equipo, sino que también dejan una base sólida para futuras iteraciones y aplicaciones de la plataforma. El impacto positivo en la sociedad y el medio ambiente refuerza la importancia de proyectos tecnológicos diseñados con un propósito claro y un compromiso con la mejora continua.

# 5 Agradecimientos

Quiero agradecer a mi familia por su apoyo durante mi formación académica, especialmente a mi mamá y mis hermanas, quienes han sido un pilar fundamental en mi vida.
Agradezco al equipo que conformó EcoVuelta, Diego, Martín y Pamela, cuyo esfuerzo y dedicación fueron clave para llevar adelante este proyecto. Gracias a todos por su apoyo y confianza en este camino.
A todos quienes me ayudaron alcanzar este importante logro.

# 6 Referencias

[1] McKinsey & Company, «Sustainability in Freight: Reducing Empty Miles,» McKinsey & Company, 2020.

[2] European Environment Agency (EEA), «The European Environment: State and Outlook 2020,» European Environment Agency, 2020.

[3] International Transport Forum (ITF), «The Future of Road Freight,» OECD Publishing, París, 2018.

[4] World Economic Forum, «How to Decarbonize Freight Transportation: A Framework for Action,» World Economic Forum, Ginebra, 2020.

[5] Cámara Chilena de la Construcción (CChC), «Desafíos de la Logística en Chile,» Cámara Chilena de la Construcción, Santiago, Chile, 2018.

# 7 Anexos

**Tabla documentación de Endpoints**

| Ruta | Método | Descripción | Datos Enviados | Respuesta |
|---|---|---|---|---|
| **/auth/login** | POST | Autentica a un usuario. | JSON con username y password. | JSON con detalles del token de sesión. |
| **/drivers** | GET | Obtiene todos los conductores registrados. | Ninguno. | Lista de conductores en formato JSON. |
| **/drivers** | POST | Crea un nuevo conductor. | JSON con los datos del conductor. | JSON con los detalles del conductor creado. |
| **/drivers/{id}** | GET | Obtiene los detalles de un conductor específico. | Parámetro id del conductor. | JSON con los detalles del conductor. |
| **/drivers/{id}** | PUT | Actualiza los detalles de un conductor. | JSON con los datos del conductor. | JSON con los detalles del conductor actualizado. |
| **/drivers/{id}** | DELETE | Elimina un conductor por su ID. | Parámetro id del conductor. | Mensaje indicando que el conductor fue eliminado. |
| **/drivers/truck** | GET | Obtiene todos los camiones registrados. | Ninguno. | Lista de camiones en formato JSON. |
| **/drivers/truck** | POST | Crea un nuevo camión. | JSON con los datos del camión. | JSON con los detalles del camión creado. |
| **/drivers/truck/{id}** | GET | Obtiene los detalles de un camión específico. | Parámetro id del camión. | JSON con los detalles del camión. |
| **/drivers/truck/{id}** | PUT | Actualiza los detalles de un camión. | JSON con los datos del camión. | JSON con los detalles del camión actualizado. |
| **/drivers/truck/{id}** | DELETE | Elimina un camión por su ID. | Parámetro id del camión. | Mensaje indicando que el camión fue eliminado. |
| **/requests** | POST | Crea una nueva solicitud de transporte. | JSON con los datos de la solicitud. | JSON con los detalles de la solicitud creada. |
| **/requests/{company}** | GET | Obtiene solicitudes de transporte filtradas por empresa. | Parámetro company como query string. | Lista de solicitudes en formato JSON. |

| | | | | |
|---|---|---|---|---|
| **/requests/{id}** | GET | Obtiene los detalles de una solicitud específica. | Parámetro id de la solicitud. | JSON con los detalles de la solicitud. |
| **/requests/{id}** | PUT | Actualiza los datos de una solicitud. | JSON con los datos actualizados. | JSON con los detalles de la solicitud actualizada. |
| **/requests/{id}** | DELETE | Cancela una solicitud de transporte. | Parámetro id de la solicitud. | Mensaje indicando que la solicitud fue cancelada. |
| **/requests/history** | GET | Obtiene el historial de solicitudes y rutas. | Ninguno. | Lista de solicitudes y rutas en formato JSON. |
| **/requests/locations** | GET | Obtiene todas las ubicaciones registradas. | Ninguno. | Lista de ubicaciones en formato JSON. |
| **/requests/locations** | POST | Crea una nueva ubicación. | JSON con los datos de la ubicación. | JSON con los detalles de la ubicación creada. |
| **/requests/locations/{id}** | GET | Obtiene los detalles de una ubicación específica. | Parámetro id de la ubicación. | JSON con los detalles de la ubicación. |
| **/requests/locations/{id}** | PUT | Actualiza los datos de una ubicación. | JSON con los datos actualizados. | JSON con los detalles de la ubicación actualizada. |
| **/requests/locations/{id}** | DELETE | Elimina una ubicación registrada por su ID. | Parámetro id de la ubicación. | Mensaje indicando que la ubicación fue eliminada. |
| **/routes** | POST | Crea un nuevo plan de ruta. | JSON con los datos del plan. | JSON con los detalles del plan creado. |
| **/routes** | GET | Busca planes de ruta según configuración. | Parámetros de búsqueda opcionales. | Lista de planes de ruta en formato JSON. |
| **/routes/{id}** | GET | Obtiene los detalles de un plan de ruta. | Parámetro id del plan. | JSON con los detalles del plan de ruta. |
| **/routes/optimization** | POST | Envía datos al modelo de optimización. | JSON con datos de la solicitud. | JSON con resultados optimizados. |
| **/routes/optimization/{id}** | GET | Consulta los resultados optimizados. | Parámetro id de la optimización. | JSON con los detalles del resultado optimizado. |