



UNIVERSIDAD TÉCNICA
FEDERICO SANTA MARÍA

DEPARTAMENTO DE ELECTROTECNIA E INFORMÁTICA
INGENIERÍA EN INFORMÁTICA

Arquitectura y diseño de un sistema de planificación y asignación logística para Ecovuelta

Diego Javier Muñoz Del Pino

diego.munozde@sansano.usm.cl

Óscar Francisco Carrasco Vera

Profesor guía

Resumen

KEYWORDS: Logística sostenible, arquitectura por capas, microservicios, metodología ágil, optimización logística.

Esta tesina trabaja el diseño y la implementación de Ecovuelta, una plataforma logística sostenible en Chile enfocada en reducir el "síndrome del camión vacío". Existe actualmente una falta de planificación y optimización en el rubro del transporte terrestre que puede ser mejorado implementando sistemas como este.

Utiliza una arquitectura por capas basada en principios SOLID, KISS y YAGNI para garantizar flexibilidad y escalabilidad. El sistema combina Angular (UI), Spring Boot y FastAPI (lógica de negocio), y MongoDB (datos), alojándose en Amazon AWS. La metodología ágil (Scrum y Kanban) facilitó el desarrollo y ajustes iterativos.

Los componentes incluyen diagramas de arquitectura, patrones (MVC, cliente-servidor), y diccionarios de datos. Además, incorpora pruebas de rendimiento, mostrando tiempos de respuesta óptimos en microservicios, mostrando las fortalezas y las áreas de mejora de cada uno de los endpoints.

Las fortalezas del sistema incluyen bajo tiempo de respuesta, manejo eficiente de concurrencia y estabilidad. Áreas de mejora apuntan a pruebas con carga extrema, optimización de código, y almacenamiento preprocesado de datos críticos. El sistema cumple los objetivos de modularidad, escalabilidad y sostenibilidad, garantizando su éxito a largo plazo.

La necesidad de crear una aplicación basada en una arquitectura robusta, segura y eficiente conlleva a concretar la creación de este servicio, fijándose atentamente en los detalles no solamente técnicos, sino que orientados a los clientes finales.

Finalmente, el tiempo invertido en generar una arquitectura bien definida permitió la creación de un sistema robusto, cumpliendo expectativas para que en el futuro sea comprensible, flexible, y fácil de mantener por los desarrolladores que trabajaron en el sistema, o desarrolladores nuevos que se puedan integrar futuramente, reduciendo el esfuerzo del trabajo y maximizando los resultados.

1. Introducción

El transporte de cargas por carretera en Chile es un eje fundamental para la economía del país, siendo crucial para la logística y distribución de productos. Este sector concentra la mayor cantidad de movimientos de carga, lo que resalta su importancia en el funcionamiento económico. La constante necesidad de modernización ha llevado al desarrollo de proyectos que integran tecnología con una conciencia ecológica, buscando mejorar la eficiencia y sostenibilidad del medio ambiente. Este sector es responsable del 11% de las emisiones de gases de efecto invernadero, lo que ha impulsado esfuerzos hacia un giro más limpio, incluyendo prácticas como la conducción eficiente y la adopción de tecnologías más limpias.

El fenómeno conocido como “**síndrome del camión vacío**” representa un desafío significativo, ya que se asocia con complicaciones económicas y logísticas cuando los camiones realizan viajes sin carga, generalmente en retorno a su punto de origen. Esto resulta en traslados ineficientes y sin beneficios económicos o ambientales.

Para abordar esta problemática, surge **Ecovuelta**, una plataforma enfocada en el B2B que permite a las empresas participar en un mercado donde pueden ofrecer y solicitar servicios de transporte para cargas específicas. Este ecosistema fomenta una comunicación efectiva entre las partes interesadas, priorizando un enfoque ecológico mediante modelos inteligentes de optimización que ayudan a las empresas a reducir sus emisiones.

El sistema **Ecovuelta** está constituido por un equipo de cuatro desarrolladores, todos encargados de distintas áreas del sistema. El objetivo principal de esta tesina y el trabajo personal que se llevará a cabo es detallar la **arquitectura y diseño** de la plataforma, entregando las herramientas a los desarrolladores para poder visualizar claramente las funcionalidades que entregará el sistema y así reducir el esfuerzo de ellos al momento de programar. Se trabajará en el sistema desde su concepción hasta su disponibilidad al público. Se abordará la integración eficaz para asegurar un funcionamiento óptimo con mínima pérdida de recursos, creando un sistema robusto y seguro para su uso diario.

1.1. Contexto y antecedentes

La necesidad de una logística eficiente es crítica en el transporte terrestre, donde los viajes vacíos representan un obstáculo significativo tanto económica como ecológicamente. Estos viajes no solo generan costos adicionales por combustible y mantenimiento, sino que también contribuyen al desgaste prematuro de los vehículos. En Chile, se estima que el 40% de los camiones circulan vacíos, lo que subraya la urgencia de implementar soluciones efectivas. Las empresas deben considerar estrategias como la **optimización de rutas**, el establecimiento de acuerdos para cargas de retorno y la consolidación de envíos para minimizar estos viajes vacíos. La colaboración entre empresas también se presenta como una oportunidad para compartir información sobre rutas y necesidades logísticas, mejorando así la eficiencia operativa.

1.2. Definición del problema.

El **síndrome del camión vacío** no solo afecta la rentabilidad de las empresas transportistas, sino que también tiene un impacto ambiental considerable. Cada kilómetro recorrido sin carga representa una oportunidad perdida para generar ingresos y aumenta los costos operativos. La falta de información y trazabilidad en las operaciones logísticas dificulta la planificación eficiente de rutas. Este problema es especialmente relevante en el contexto actual, donde las empresas buscan reducir costos operativos y mejorar su competitividad en un mercado cada vez más exigente. La implementación de tecnologías avanzadas puede ser clave para abordar este desafío.

1.3. Breve descripción sobre la propuesta de solución.

La propuesta **Ecovuelta** busca crear una plataforma informática que conecte a empresas que necesitan mover cargas con aquellas que ofrecen servicios de transporte, optimizando así los recursos disponibles. Esta solución no solo aborda el problema económico asociado con los viajes vacíos, sino que también promueve prácticas sostenibles mediante la reducción de emisiones. Se llevará a cabo una investigación exhaustiva sobre los lenguajes de programación a implementar, junto con diagramas de secuencia y otros elementos necesarios para explicar la funcionalidad del sistema. Además, se definirán conceptos clave como seguridad, modularidad y escalabilidad, analizando cómo estos influyen en la arquitectura del sistema.

Una arquitectura bien definida es fundamental para asegurar el éxito y sostenibilidad del proyecto, facilitando la comunicación entre componentes y garantizando un mantenimiento efectivo a largo plazo. La inversión en diseño arquitectónico se considera una decisión estratégica esencial para cualquier proyecto de software robusto y competitivo.

1.4. Objetivos generales y específicos

El objetivo general de la solución en la visión de la arquitectura de software es definir la plataforma en términos de diagramas, diccionario de datos, casos de uso, funciones y orquestación de todos los ámbitos de la programación, juntando la parte de Front End, Back End y bases de datos, velando por una integración eficaz para un funcionamiento óptimo, con mínima pérdida de recursos, creando un sistema robusto y seguro para su uso diario.

Los objetivos específicos son los siguientes:

- Investigar los lenguajes a implementar y por qué son utilizados.
- Realizar los diagramas de secuencia, diagrama de clases, diccionario de datos, diagramas de arquitecturas y casos de uso.
- Definir cuáles son los actores y cómo éstos interactúan con el sistema.
- Definir conceptos de seguridad, modularidad, escalabilidad, testabilidad, entre otros y cómo estos influyen en la arquitectura de la solución.

1.5. Justificación del proyecto

El desarrollo del sistema **Ecovuelta** es fundamental para abordar el significativo problema del **síndrome del camión vacío**, que afecta la rentabilidad de las empresas de transporte al generar costos adicionales en combustible y mantenimiento. Al optimizar la carga y minimizar los viajes sin

carga, **Ecovuelta** no solo ayudará a reducir estos gastos operativos, sino que también contribuye a la sostenibilidad ambiental al disminuir las emisiones de gases de efecto invernadero. La plataforma facilitará la comunicación entre empresas, promoviendo un ecosistema colaborativo que mejora la eficiencia logística y permite una mejor planificación de rutas.

Además, la implementación de **Ecovuelta** representa una oportunidad para adoptar tecnologías avanzadas que transformen el sector del transporte, donde este enfoque no solo responde a un entorno empresarial competitivo, sino que también tiene un impacto social positivo al mejorar la calidad del aire y la salud pública en las comunidades.

2. Marco teórico

Cuando se habla de arquitectura de software, se debe tomar en cuenta los factores que hacen posible el inicio de la construcción de un sistema, donde los puntos más importantes por analizar son los principios de diseño de software, documentación y comunicación, patrones de arquitectura de sistemas y evaluación continua.

2.1. Principios de diseño de software

Los desarrollos de sistema actuales son considerados iteraciones durante años de mejoras continuas y de esfuerzo por estandarizar las buenas prácticas, es aquí donde se generan los principios de diseños de software. Estos principios son directrices fundamentales que orientan a los desarrolladores a crear códigos y sistemas que resultan ser efectivos y de alta calidad, y buscan mejorar conceptos clave como la mantenibilidad, flexibilidad y eficiencia. Proporcionan un marco que invita a tomar decisiones informadas de la organización y estructuración del código, minimizando la complejidad y facilitando la colaboración entre equipos.

Algunos de los principios comunes son los siguientes:

2.1.1. Principios SOLID

Cuando se habla de principios de desarrollo de software, SOLID es el más común. Está enfocado en hacer que los diseños de software sean más comprensibles, flexibles y mantenibles, que generalmente es aplicado en desarrollo orientado a objetos. Es un acrónimo mnemónico¹ que significa:

- **S (Single Responsibility Principle)**

El principio de SRP establece que un módulo debe tener una y solo una razón para cambiar. Este concepto está relacionado con los conceptos de acoplamiento y cohesión, donde el enfoque es disminuir el acoplamiento y aumentar la cohesión, de lo contrario si existe más de una razón para cambiar una clase, puede que tenga más de una responsabilidad.

Una ventaja es el aislamiento de las clases, lo que hace realizar cambios de una forma fácil, minimizando los efectos colaterales.

- **O (Open/Closed Principle)**

¹ Ocupar la primera letra de cada palabra para hacer otra

El principio OCP establece que los módulos, clases o funciones deben estar abiertos para su extensión, pero cerrados para su modificación, esto significa que el código debe tener la capacidad de agregar nuevas funcionalidades sin modificar las existentes. Esto conlleva al acoplamiento si no está implementado, lo que genera una cascada de cambios al momento de presentar una nueva funcionalidad.

Lo más común para este principio es ocupar interfaces o clases abstractas, que pueden cambiar la implementación de la clase y mantener la interfaz intacta.

- **L (Liskov Substitution Principle)**

El principio LSP propone lo siguiente: *Si a un método “q” le podemos pasar objetos “x” de la clase “T”, y luego tenemos una clase “S” que hereda de la clase “T”, entonces un objeto “y” de la clase “S” debería ser capaz de pasarse a la función “q” y esta funcionará igualmente.*

Esta definición, si bien es confusa, explica que cada clase que hereda a otra puede usarse como padre sin necesidad de conocer las diferencias entre ellas.

- **I (Interface Segregation Principle)**

El principio ISP establece la separación de las interfaces para aquellos clientes que ocupen una o más de sus funcionalidades. La violación de este principio hace que los clientes que dependen de estas interfaces sean afectados por interfaces que no son usadas por completo, por lo que es probable que sufran al realizar modificaciones al momento de realizar cambios.

Generalmente los sistemas que no se adhieren a este principio llevan consigo muy pocas pruebas unitarias o cobertura, por lo que encontrar fallos provocados por otra parte resulta ser más costoso.

- **D (Dependency Inversion Principle)**

El principio DIP establece que las entidades de software deben depender de abstracciones, no de las implementaciones, mientras que los módulos de alto nivel no deberían depender de los de bajo nivel y ambos deberían depender de abstracciones.

El punto fundamental de este principio es la flexibilidad, donde se puede cambiar fácilmente los componentes sin que los clientes que los consumen se vean afectados, ya que dependerían de la abstracción, haciendo a los módulos más fácil de reutilizar.

2.1.2. Principio KISS

Este principio es un acrónimo que proviene del inglés “*keep it simple, stupid*”, el cual se entiende por la necesidad de disminuir los errores realizando las tareas del desarrollo de una forma simple y eficiente complicándose lo menos posible.

La simplicidad es un factor importante al momento de regirse por este principio, es fundamental para no tener complicaciones, tanto en el diseño como en el desarrollo del sistema.

2.1.3. Principio YAGNI

Este principio también es un acrónimo del inglés “*You aren’t gonna need it*” y estipula que no se deben agregar funcionalidades nuevas hasta que estas no se necesiten.

Esto genera relativos retrasos con las planificaciones, en funcionalidades que deben pasar por ciclos de depuración, documentación y mantención, y que a su vez no pueden quedar bien ejecutadas ya que, si no existe el qué, es imposible que esté el cómo.

2.2. Patrones de arquitectura

Existen variadas formas de trabajar en el área del desarrollo de *software*, cada una con sus ventajas y desventajas, a continuación, se presentan las más comunes.

2.2.1. Patrón por capas

Este patrón se ha convertido en el que se basan la mayoría de las soluciones actuales, ya que provee la facilidad de dividir los componentes en grupos que realicen subtarefas, cada una con un grado de abstracción dentro de su propia capa donde cada una proporciona servicios a la capa superior. Esto permite entregar un desacoplamiento entre los componentes, lo que facilita la escalabilidad y la modularidad del sistema, permitiendo reemplazar capas sin afectar el funcionamiento entero del sistema

Los sistemas informáticos comúnmente presentan tres capas, las cuales se presentan a continuación.

- **Capa de presentación o UI**

Esta capa es responsable de gestionar la interfaz del usuario, la cual permite presentar los datos de manera que sean comprensibles y accesibles. Incluye factores como la captura de las acciones del usuario, la representación visual de los datos, la transformación y validación de los datos para ser enviados a la capa inferior, siendo fundamental para la interacción entre el sistema y el usuario final.

- **Capa de lógica de negocios o servicios**

Esta capa se encarga de implementar las reglas y lógicas centrales de nuestro sistema, donde generalmente se procesan datos obtenidos desde la capa de presentación, donde también actúa realizando validaciones complejas que la capa de presentaciones no pueda manejar, como verificaciones de múltiples condiciones, cálculos y transformación que necesite realizar la capa como calcular precios, descuentos. Es un intermediario entre la capa de presentación y el acceso a las capas inferiores de acceso de datos.

- **Capa de acceso a datos o persistencia**

En esta capa ocurren las interacciones con las fuentes de datos, ya sea de bases de datos o servicios externos, donde se hacen las operaciones de escritura y lectura de datos y generalmente proveen métodos para generar CRUD, encargándose de la abstracción de algunos métodos para poder ser reutilizados y simplificados como lo son los *getUserById* por ejemplo, donde traería a un usuario por su id sin necesidad de saber qué hay detrás.

Las ventajas de ocupar este patrón de arquitectura son los factores claves como **mantenibilidad** por la separación de responsabilidades en capas, **modularidad** por la capacidad de agregar, modificar o eliminar capas y **escalabilidad** por su capacidad de aumentar vertical u horizontalmente las capas individualmente, mientras que las desventajas son **complejidad y rendimiento** al momento de tener muchas capas, recordando que entre la comunicación de componentes existe una intermitencia para llegar de una capa a otra.

2.2.2. Patrón cliente-servidor

Este patrón organiza las comunicaciones entre dos entidades primordiales, el cliente siendo la parte que inicia las solicitudes al servicio, donde su función principal es interactuar con el usuario y generar peticiones al servidor, mientras que el servidor gestiona servicios y maneja los recursos a los clientes, y realizan tareas como almacenar datos y mantener la lógica de negocios.

Esta conexión entre ambas entidades se realiza mediante protocolos HTTP, donde el intercambio puede incluir el transporte de datos, la petición a un servicio en particular o entrega de recursos.

Algunas ventajas de este patrón son el mantenimiento centralizado, donde existe toda la lógica relacionada al sistema guardado en el servidor, lo que hace que el mantenimiento y las actualizaciones sean más fáciles y la seguridad ya que podemos securitizar los servidores, y mantener protegida la información y los datos. Mientras que las desventajas caen en la dependencia pura del servidor, por lo que si este servidor llega a fallar los clientes pierden el acceso a los recursos y los costos de arquitectura, los cuales suelen recurrir a necesitar hardware y software más potentes para alimentar a las máquinas clientes con sus necesidades.

2.2.3. Patrón MVC

Este patrón divide una aplicación interactiva en tres partes, conocidas como modelo, vista y controlador, separándolos en base a las preocupaciones internas, permitiendo el mantenimiento, la escalabilidad y organización del código. Estas partes son interconectadas y se comunican entre sí para mantener el flujo de la aplicación.

Es comúnmente encontrado en los principales lenguajes de programación y en *frameworks* como Django y Rails.

A continuación, indagamos en la separación de componentes y explicamos cada una.

- **Modelo**

El modelo se encarga de la gestión de los datos, haciendo representaciones de su estructura y las interacciones con las bases de datos, aquí se definen factores como el tipo de datos, nombre de las tablas, conexiones foráneas si es que estas existieran en los modelos SQL, y en algunos lenguajes provee la creación automática de ellas para su visualización y manejo de datos.

- **Vista**

Las vistas son las interfaces gráficas que presentan los datos y se los visualiza al usuario, esta se encarga de mostrar la información proveniente del modelo y recibir las interacciones del usuario que son capturadas por el controlador del sistema.

- **Controlador**

El controlador es un intermediario entre el modelo y las vistas. Procesa las acciones realizadas por los usuarios (vistas), e interactúa con el modelo actualizando las vistas. Es el responsable de hacer conversar los componentes para su correcto funcionamiento.

Las ventajas de este patrón es la **separación de responsabilidades**, ya que cada una tiene su propia preocupación haciéndolo mantenible. Su **escalabilidad** permite agregar nuevas funcionalidades sin afectar las partes existentes. Respecto a las desventajas, por su **complejidad inicial**, requiere de la creación de variados componentes para su funcionamiento inicial y la **latencia por sobrecarga**, lo que puede traer latencia al momento de la comunicación entre ellos.

2.3. Documentación y comunicación del equipo

La documentación de la arquitectura es un elemento esencial de cualquier proyecto de software, y tiene impacto en los ámbitos de colaboración y mantenibilidad a largo plazo de un proyecto.

El equipo es la pieza fundamental del desarrollo de software, y tener la capacidad de documentar permite alinear los intereses del equipo, mejorar la comprensión del sistema acerca de cómo está construido, entrar en un estado de pensamiento crítico y tomar decisiones informadas que van a ayudar a reducir costos y mal entendidos a largo plazo. Esto beneficia al ingreso de nuevos miembros para que tengan una entrada ágil al proyecto, y mejorar a la cohesión del equipo.

2.4. Evaluación continua

La arquitectura de software itera, como todos los proyectos, razón por la cual tiene que existir el constante cuestionamiento de las mejores formas de llevarlo a cabo, como tratar de identificar problemas en las etapas tempranas del proyecto, la alineación de los requisitos en los sentidos del proyecto y de la empresa, para entender si el enfoque está correcto, y es aquí donde podemos implementar técnicas como **ATAM** (*Architecture Tradeoff Analysis Method*) la cual se enfoca en la evaluación de una arquitectura en relación con sus atributos de calidad y objetivos empresariales.

3. Metodología

Existen variados tipos de enfoques para trabajar en un proyecto de software, que se adaptan dependiendo del porte del proyecto, la complejidad y el tiempo pronosticado. A continuación, se presenta el enfoque que se utilizará para desarrollar la arquitectura del sistema.

3.1. Enfoque Ágil

Para el análisis y diseño de la arquitectura de software, se implementará este enfoque que combina las metodologías Scrum y Kanban, donde podemos aprovechar de las fortalezas ofrecidos por estos marcos de trabajo para proporcionar un entorno flexible que puede responder a cambios en los requisitos y mejorar continuamente el proceso. A continuación, se presentará una definición de ambas.

3.1.1. Metodología Scrum

Esta metodología se basa en un marco de trabajo iterativo, donde se realizan entregas parciales cada cierto tiempo del producto final, que se encuentran priorizadas para mejorar su desarrollo. Esta metodología se divide en 3 elementos claves para su funcionamiento, los cuales incluyen los roles, los eventos y los artefactos.

- **Roles**

Los roles que existen en esta metodología son los de **Product Owner**, el cual representa al cliente y es el responsable de maximizar el valor del producto, por él pasan las comunicaciones con el cliente externo y generan el vínculo entre equipo interno y los usuarios finales.

En esta estructura también participa el **Scrum Master**, el cual está encargado de velar porque el enfoque ágil se cumpla, en especial la metodología Scrum, y los desarrolladores no tengan impedimentos, entregando las herramientas necesarias para que el proyecto avance sin problemas.

Finalmente están los desarrolladores que realizan el trabajo de codificar para entregar los resultados al final de cada incremento.

- **Eventos**

Los eventos son las fases en las que se basa la metodología Scrum, cada una de estas tiene una finalidad distinta que juntas aportan a generar ciclos iterativos de una forma óptima.

El corazón de Scrum son los **Sprint**, esto se define como un bloque de tiempo en el cual se debe hacer una iteración con una parte del producto final hecha, por tanto, se debe hacer una **Planificación de Sprint**, en la que se debe definir cuáles serán las tareas que se van a trabajar. Una vez que se da inicio al Sprint se generan dos eventos nuevos: **Scrum Diario** o **DS**, la cual es una reunión diaria de no más de 15 minutos para verificar el estado del objetivo y si existe algún bloqueo por parte del equipo de desarrollo, y la **Revisión del Sprint**, la cual se encarga de ver el alcance del Sprint y ajustarlo a medida que avanza el tiempo para ver si hay que modificar o arreglar algo. El último evento por realizar se llama **Retrospectiva del Sprint**, la cual es una oportunidad para juntarse con el equipo ágil y hacer una revisión de los roles, del alcance del Sprint y si se logró cumplir con las tareas y plazos.

- **Artefactos**

Los artefactos son herramientas que ayudan a gestionar el trabajo y maximizar la transparencia. En primer lugar, se cuenta con el **Product Backlog**, la cual es una lista que contiene todo lo necesario para empezar a armar o a realizar mejoras del producto, esta se encuentra dividida en múltiples ítems y generalmente están refinados y priorizados.

Por otro lado, el **Sprint Backlog** es un conjunto de elementos seleccionados del **Product Backlog**, donde el equipo de desarrollo se compromete a completar durante un Sprint específico. Incluye las tareas a realizar para cumplir con dichos ítems como con la planificación de entrega de incremento del producto final del Sprint.

Finalmente existe el **Incremento**, la cual es la suma de todos los elementos del **Product Backlog** completados durante un Sprint y de todos los incrementos anteriores, al finalizarlos deben haber cumplido con la definición de “terminado” acordada por el equipo.

3.1.2. Metodología Kanban

Kanban es un marco de trabajo ágil representado por tableros para la rápida visualización del trabajo y esfuerzo presentado por los equipos, entregando valor al cliente de manera continua. Esta metodología se caracteriza por el uso de tarjetas como historias de usuario y cómo van avanzando en distintas etapas del proyecto. Algunos principios claves son:

- **Visualizar el flujo de trabajo**

Al representar las tareas en un tablero, las distintas etapas del desarrollo se representan por columnas, y representa la actualidad de las tareas que están desarrollando los integrantes del equipo.

- **Limitar el trabajo en curso**

Se pueden limitar las tareas que están en cada columna, de esta forma se puede evitar la sobrecarga y mejorar el enfoque de las tareas actuales antes de empezar tareas nuevas.

- **Implementar ciclos de feedback**

Al tener la capacidad de mover las tarjetas de etapas, podemos avanzar o retroceder las acciones que están siendo trabajadas, de esta forma se puede controlar la razón por la cual se deben retroceder, o también agregar comentarios a medida que van avanzando. Este *feedback* se ve más claro al utilizar productos como JIRA.

3.2. Herramientas de modelado empleadas

Para la creación de los modelos que se verán más adelante, se realizarán en base al lenguaje UML, o Lenguaje Unificado de Modelado, el cual consiste en un estándar para el modelado visual de proyectos, estos serán desarrollados en una herramienta gratis para el modelado como lo es **Lucidchart** o **draw.io** como también plantillas utilizando **plantuml**.

Algunos de los diagramas que estarán presente en esta arquitectura son:

- Diagrama de arquitectura del sistema (HLD)
- Diccionario de datos
- Diagrama de secuencia
- Casos de uso
- Diagrama de clases

3.3. Diseño y patrón escogidos

En los puntos descritos anteriormente se explicaron algunos de los principios de diseño y patrones de arquitectura que actualmente están siendo usados en empresas grandes y medianas, por tanto, la idea es mantener el nivel de esas empresas para empezar a realizar la arquitectura de este sistema, llegando a la decisión de tomar los principios descritos (**SOLID – KISS – YAGNI**).

Estos principios son fundamentales para obtener la escalabilidad y flexibilidad del sistema, en conjunto con velar por la reducción del trabajo innecesario y así aumentar la eficiencia del equipo de desarrollo. Por otro lado, el patrón que se utilizará será el **Patrón por capas**, ya que se cuentan con las mismas características que los principios de diseño.

La forma en que será dividido este patrón es utilizando tecnologías que llevan años en el mercado, por tanto, existe una amplia base de información, documentación y ayuda provenientes de foros en internet para poder desarrollar una arquitectura robusta y escalable.

Para la capa de presentación o UI se utilizará **Angular 16+**, que provee la capacidad de realizar páginas web interactivas, sin necesidad de recargar el buscador, y sobre todo escalables.

Para la capa de lógica de negocios, se utilizará **Java 17+** con **Spring Boot**, la cual es muy potente para realizar API REST, y también capaces de manejar configuraciones como *Spring Security*, *Circuit Breaker*, *Multithread* y otras que van a ayudar al funcionamiento y correcto almacenamiento de los datos. Esto acompañado de otra API REST, configurado con **Python 3** con **FastAPI**, para recibir las peticiones de la integración entre ambos servicios de lógica.

Finalmente, para la capa de datos se utilizará **MongoDB**, aprovechando una base de datos *NoSQL*, flexible y rápida al no tener que armar y buscar datos ligados fuertemente, sino que considerar una base semi-estructurada capaz de entregar información oportuna.

4. Análisis de requerimientos

En esta sección se describirán los requerimientos funcionales y no funcionales del sistema.

4.1. Requerimientos funcionales

Los requerimientos funcionales se definen como el qué debería hacer el sistema, los puntos a considerar son los siguientes:

- Permitir el ingreso de nuevas solicitudes de movimientos de carga.
- Procesar las solicitudes para generar planificaciones de rutas.
- Ingresar activos a la plataforma según el perfil designado.
- Asignar planificaciones de rutas a camiones que cumplan con las certificaciones requeridas.
- Visualizar el estado de los camiones en un tablero para su planificación.
- Enviar alertas vía WhatsApp para ofrecer los servicios de transporte a clientes.

4.2. Requerimientos no funcionales

Los requerimientos no funcionales son aquellos que se establecen como niveles de estándar para el sistema. Estos puntos son los siguientes:

- Se manejará una información anual de 50 GB.
- Se contempla menos de 100 usuarios, con posible escalamiento.
- Se contempla una carga de 250 peticiones concurrentes.
- Se manejará un SLA de 48 horas para tickets.
- Se debe manejar una seguridad a la información sensible, como son las contraseñas e información confidencial como número de teléfono y correos de los trabajadores.

5. Diseño de la arquitectura

En esta sección, se muestran los distintos diagramas que explican el funcionamiento de la arquitectura de software.

5.1. Diagramas de la arquitectura

Los diagramas que se presentarán a continuación son el apoyo visual para dimensionar en qué forma estará armado el proyecto Ecovuelta, mostrándolo de un enfoque general a lo más específico.

5.1.1. Diagrama HLD o de alto nivel

Este diagrama pretende explicar de una forma general los componentes y cómo estarán conectados entre sí para hacer fluir la información de una capa a otra.

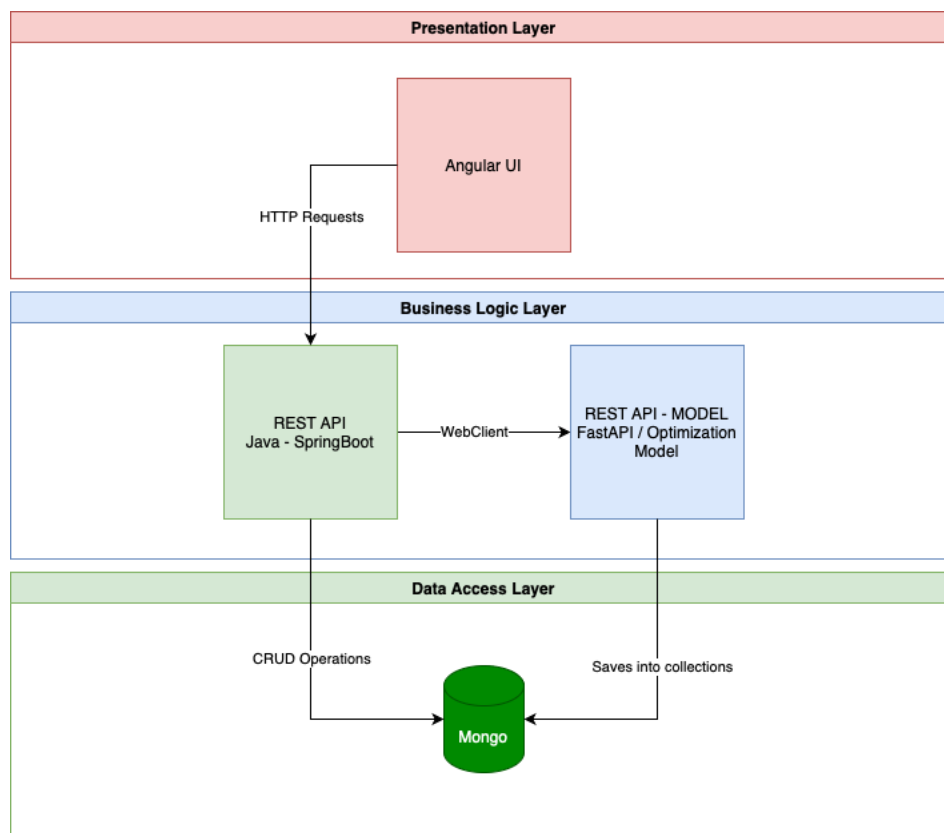


Figura 1. Representación general de la arquitectura

Fuente: Elaboración propia

Se logra visualizar las capas que tendrá el sistema, utilizando el patrón por capas definido previamente. Podemos observar que el flujo que tendrá el sistema, donde la capa de presentación generará peticiones a la capa de lógica de negocios, donde esta capa va a distribuir dependiendo de qué tipo de petición esté realizando para finalmente generar alguna acción en la capa de acceso a los datos.

5.1.2. Diccionario de datos

Los diccionarios ayudan a los desarrolladores a visualizar y entender mejor cuál es la lógica detrás de la información que se almacenan en la capa de acceso a los datos. Aquí se definen conceptos como los campos, el tipo de dato, la descripción del campo y un valor de ejemplo. Para este proyecto se ocuparán documentos y colecciones, ya que están ligados al tipo NoSQL con Mongo DB, y se mostrarán las colecciones a continuación.

Tabla 1. Listado de colecciones de la base de datos

Colección	Descripción
Vehicle Collection	Vehículos que transportan carga
Request Collection	Solicitudes de mover carga
Route Collection	Resumen general de una planificación realizada por el modelo
Route Details Collection	Detalles de una planificación realizada por el modelo
Assigned Plan Collection	Planificaciones de rutas asignadas a un vehículo
User Collection	Usuarios pertenecientes al sistema
Restriction Collection	Restricciones de las cargas con las configuraciones
Location Collection	Ubicación de las bodegas y posibles destinos
Driver Collection	Conductores de los vehículos

Fuente: Elaboración propia

Por cada una de estas colecciones se presenta su diccionario de datos.

- Vehicle Collection

Tabla 2. Diccionario de datos colección de vehículos

Campo	Tipo	Descripción
id	String	identificador único de los vehículos
carMake	String	Marca del vehículo
active	boolean	Si el vehículo está activo o no
equipmentType	String	Tipo de equipo rodante
axle	String	Cantidad de ejes (Ej: 3, 2+1, 1)
maxCapacity	Integer	Capacidad máxima de peso (ton)
model	String	Modelo del vehículo
configuration	String	Configuración del vehículo
type	String	Vehículo CAMION/PLUMA
year	Integer	Año del vehículo
certifications	String	Certificaciones del vehículo
company	String	Empresa a la que pertenece
type	String	Tipo de vehículo

Fuente: Elaboración propia

- Requests Collection

Tabla 3. Diccionario de datos colección de solicitudes

Campo	Tipo	Descripción
id	String	Identificador único de la solicitud
origin	String	Punto de origen de la solicitud
destination	String	Punto de destino de la solicitud
weight	Integer	Peso de la carga
startDate	Date	Fecha en la que la carga se debe mover
endDate	Date	Fecha en la que la carga debe llegar
expirationDate	Date	Fecha en la que la solicitud expira
loadType	String	Tipo de carga que va a ser transportada
certifications	String	Certificación necesaria para llevar la carga
description	String	Descripción de la solicitud (observaciones)
price	Integer	Valor de la carga que se está transportando en pesos chilenos (CLP)
company	String	Empresa que ingresó la solicitud
configType	String	Tipo de configuración que necesita la carga
vehicleType	String	Tipo de vehículo que necesita la carga
maxCapacity	Integer	Capacidad máxima para el vehículo
alive	boolean	Si está disponible la carga

Fuente: Elaboración propia

- Route Collection

Tabla 4. Diccionario de datos colección de planificación de rutas

Campo	Tipo	Descripción
id	String	Id de la ruta planificada
origin	String	Punto de origen de la planificación de ruta
stops	Integer	Cantidad de paradas planificadas en la ruta
destination	String	Punto de destino de la planificación de ruta
configType	String	Configuración necesaria para llevar la carga
vehicleType	String	Tipo necesario de camión para llevar la carga
certifications	String	Certificaciones necesarias para llevar la carga
costs	Integer	Costos asociados a transportar la carga a través de la ruta en pesos chilenos (CLP)
earnings	Integer	Resultado entre los ingresos y costos por transportar las cargas en pesos chilenos (CLP)
startDate	Date	Fecha de inicio de la ruta
endDate	Date	Fecha de término de la ruta
details	List	Lista de detalles si es que tiene paradas

assigned	boolean	Determina si la carga fue asociada o no
----------	---------	---

Fuente: Elaboración propia

- Route Details Collection

Tabla 5. Diccionario de datos colección de detalle planificación de rutas

Campo	Tipo	Descripción
id	String	Id del detalle de una planificación en particular
routeld	String	Id de la planificación de ruta general
origin	String	Punto de partida de la solicitud de transporte de carga
destination	String	Punto de destino de la solicitud de transporte de carga
startDate	Date	Fecha de inicio de la solicitud de transporte de la carga
endDate	Date	Fecha de llegada de la solicitud de la carga
configType	String	Configuración necesaria para llevar la carga
vehicleType	String	Tipo necesario de camión para llevar la carga
weight	Integer	Capacidad de la carga en ton
costs	Integer	Costos asociados a la ruta de transporte de la carga en pesos chilenos (CLP)
price	Integer	Valor de la carga que se está transportando en pesos chilenos (CLP)
earnings	Integer	Resultado entre los ingresos y costos en pesos chilenos (CLP)
order	Integer	Número de orden de la planificación de ruta general

Fuente: Elaboración propia

- Assigned Plan Collection

Tabla 6. Diccionario de datos colección de rutas asignadas

Campo	Tipo	Descripción
id	String	Id de la asignación de un plan de ruta a un vehículo
vehicleId	String	Id del vehículo asignado al plan
routeld	String	Id del plan de ruta asignado
company	String	Empresa asignada al viaje
status	String	Estado actual del viaje
startDate	Date	Fecha de inicio del plan de transporte
endDate	Date	Fecha de término del plan de transporte
earnings	Integer	Resultado entre los ingresos y costos en pesos chilenos (CLP)
driver	String	Nombre del conductor asignado
details	List	Detalles si existen más de una parada
savedCo2	Integer	Emisiones de gas ahorradas escogiendo una ruta recomendada por la plataforma Ecovuelta .

Fuente: Elaboración propia

- Users Collection

Tabla 7. Diccionario de datos colección de usuarios

Campo	Tipo	Descripción
id	String	Id del usuario que representa la empresa
name	String	Nombre de la persona que representa la empresa
lastName	String	Apellido de la persona que representa la empresa
company	String	Empresa a la que la persona representa
email	String	Correo de la persona que representa a la empresa
phone	String	Número de la persona que representa la empresa
comune	String	Comuna de la ubicación de la matriz de la empresa
region	String	Región de la ubicación de la matriz de la empresa
rut	String	Rut del usuario
type	String	Tipo de cargas que maneja la empresa
rating	Integer	Sistema de puntuación de la empresa basado en 5 puntos.

Fuente: Elaboración propia

- Driver Collection

Tabla 8. Diccionario de datos colección de conductores

Campo	Tipo	Descripción
id	String	Id del conductor
name	String	Nombre del conductor
lastName	String	Apellido del conductor
company	String	Empresa del conductor
email	String	Correo del conductor
phone	String	Número telefónico del conductor
comune	String	Comuna del conductor
region	String	Región del conductor
rut	String	Rut del conductor
certifications	String	Certificaciones que tiene el conductor

Fuente: Elaboración propia

- Location Collection

Tabla 9. Diccionario de datos colección de ubicaciones

Campo	Tipo	Descripción
id	String	Id del usuario que representa la empresa
branchName	String	Nombre de la sucursal
agencyName	String	Nombre de la agencia
businessName	String	Nombre de la unidad de negocio
address	String	Dirección asociada
comune	String	Comuna de la ubicación de la matriz de la empresa
region	String	Región de la ubicación de la matriz de la empresa
type	String	Tipo de la empresa (Rubro)

Fuente: Elaboración propia

- Restriction Collection

Tabla 10. Diccionario de datos colección de restricciones

Campo	Tipo	Descripción
id	String	Id del usuario que representa la empresa
loadType	String	Tipo de carga
configType	String	Configuración necesaria para mover la carga
vehicleType	String	Tipo de vehículo para mover la carga
maxCapacity	Integer	Capacidad máxima en ton.

Fuente: Elaboración propia

5.1.3. Diagramas de secuencia

Estos diagramas son esenciales para explicar las funcionalidades del sistema de una forma que los desarrolladores puedan realizar su trabajo con un flujo claro a seguir. Las funcionalidades que se explicarán a través de los diagramas de secuencia son los siguientes.

- Funcionalidad de Login.

El usuario deberá tener la capacidad de autenticarse ante el sistema, esto apelando al principio de seguridad impartida por el proyecto. (Véase [Anexo A](#))

- Funcionalidad de agregar solicitud de movimiento de carga

El actor que desea mover alguna carga tiene que llenar cierta información para ser procesada por el modelo y, por consecuencia, ser ofrecida a los actores que ofrecen los servicios de transporte. (Véase [Anexo B](#))

- Funcionalidad de registrar un nuevo usuario

Los actores que deseen hacer negocios a través de la plataforma deben estar registrados en el sistema, por lo que es necesario tener un formulario de registro. (Véase [Anexo C](#))

- Funcionalidad de enviar WhatsApp

Los actores que mueven carga deben ser capaz de ofrecer su vehículo, si tienen uno disponible, para poder enviar mensajes a los encargados de las empresas que supervisan los vehículos, **no** al conductor directamente. (Véase [Anexo D](#))

- Funcionalidad de asignar vehículos a planes de ruta e ir a buscar los planes asignados

Los actores que mueven carga deben ser capaces de asignar rutas a sus vehículos, y luego poder filtrar por sus asignaciones. (Véase [Anexo E](#))

Los siguientes diagramas de secuencia son basadas en un CRUD para poder realizar las operaciones de buscar, crear, actualizar y borrar.

- Funcionalidades de los usuarios (Véase [Anexo F](#))
- Funcionalidades de los vehículos (Véase [Anexo G](#))
- Funcionalidades de las ubicaciones (Véase [Anexo H](#))
- Funcionalidades de los conductores (Véase [Anexo I](#))
- Funcionalidades de las solicitudes (Véase [Anexo J](#))
- Funcionalidades de las rutas planificadas (Véase [Anexo K](#))

5.1.4. Casos de uso

Los casos de uso explican la funcionalidad del sistema desde la perspectiva de los usuarios. En este caso, tenemos dos actores que son los clientes, o sea los que necesitan llevar una carga, y los operadores transportistas, o sea las empresas que tienen camiones para llevar esas cargas. Al dividir estos actores, podemos identificar las acciones que cada uno puede realizar dentro del sistema. (Véase [Anexo L](#))

5.1.5. Diagrama de clases

Este diagrama se enfoca en canalizar las entidades, o clases que existen en el sistema y una leve representación de cómo funcionan a nivel de relaciones, donde también se consideran aspectos como sus atributos y sus métodos. (Véase [Anexo M](#))

5.2. Justificación de la decisión

Los diagramas y diccionarios realizados en el punto anterior están basados en la construcción del sistema de una forma robusta y rápida. A lo largo del tiempo, se fueron realizando mejoras a los distintos diagramas de secuencia, pasando por iteraciones varias que finalmente resultaron en los presentados. Estos diagramas describen cómo debería funcionar el sistema y sirve como documentación para su posterior uso.

6. Comparación de arquitecturas

En este punto se discutirán distintas arquitecturas con las que se puede armar un sistema informático, aquí veremos dos de las más comunes.

6.1. Arquitectura monolítica

La arquitectura monolítica se refiere a un enfoque tradicional donde las funcionalidades de la aplicación están integradas en un único código base, o sea, está todo centralizando en un solo componente que tiene todas sus funcionalidades en el mismo lugar. Funcionalidades como la interfaz de usuario, la lógica de negocio y el acceso de base de datos se desarrollan y despliegan como una unidad cohesiva.

Algunas características claves de este sistema son:

- Integración completa: Todos los módulos y funcionalidades están interconectados y dependen unos de otros.
- Despliegue único: La aplicación se despliega como un solo artefacto, lo que simplifica el proceso inicial.
- Desarrollo centralizado: Un solo equipo o grupo es responsable de toda la aplicación, lo que puede facilitar la comunicación o en algunos casos, dificultarla.

Esta arquitectura es la más común dentro de aplicaciones de software ya que ofrece un inicio rápido para formar sistemas que se comuniquen de forma rápida, ya que no existe comunicación externa entre protocolos.

Algunos pro y contra de esta arquitectura se presentan a continuación:

- Pros
 - Simplicidad en el desarrollo al utilizar un único código base, lo que facilita la gestión y el despliegue inicial del sistema.
 - Rendimiento por la comunicación interna de componentes.
 - Facilidad de pruebas unitarias y de integración debido a la cohesión del sistema.
- Contras
 - Escalabilidad limitada a medida que la aplicación crece, puede volverse difícil escalar y mantener.
 - Los despliegues después de la primera fase se pueden complicar al tener que recompilar y redespregar toda la aplicación.
 - Se convierte difícil adoptar nuevas tecnologías para integrar nuevas herramientas o lenguajes.

6.2. Arquitectura por capas

La arquitectura por capas, como ya fue mencionada anteriormente, organiza la aplicación en diferentes niveles o capas lógicas, cada una con responsabilidades específicas. Comúnmente se divide en tres capas: presentación, lógica de negocio y acceso a datos. Este enfoque es el que busca promover la modularidad y facilitar el mantenimiento.

Algunas características de la arquitectura por capas son:

- Separación de responsabilidades: Cada capa tiene una función específica y se encarga de un conjunto particular de tareas, lo que delimita las responsabilidades.
- Modularidad: Permite que cada capa sea desarrollada, probada y mantenida de forma independiente, lo que reduce la complejidad del código y facilita la reutilización de componentes.
- Flexibilidad y adaptabilidad: Se adapta a diferentes entornos y requisitos del sistema lo que permite realizar ajustes sin necesidad de reestructurar completamente la aplicación.

Esta arquitectura promueve los factores claves que se necesitan para mantener una arquitectura robusta, y que sea mantenible en el tiempo, sin la necesidad de dificultar el proceso de creación, ni de modificación si es que este requiriese.

Algunos pro y contras de esta arquitectura se presentan a continuación:

- Pros
 - La reutilización de código para ser ocupadas en distintos proyectos, reduciendo el tiempo y costo de desarrollo.
 - La escalabilidad de agregar o quitar capas según las necesidades del sistema, facilitando la adaptación a cambios en la demanda o en los requisitos del negocio.
 - La mantenibilidad simplificada al momento de abordar problemas en capas específicas sin afectar a otras, lo que reduce el riesgo de introducir nuevos errores durante el mantenimiento.
- Contras
 - La complejidad del sistema debido a la necesidad de interfaces y comunicación entre las capas, lo que puede dificultar la comprensión y gestión.
 - Los costos elevados al momento de diseñar y desarrollar varias capas separadas que puede incrementar los costos en términos de desarrollo y potencial infraestructura.
 - El rendimiento se ve un poco afectado debido a que la información debe viajar entre capas, y depende de factores externos si es que no están bien optimizados.

6.3. Rendimiento y mantenibilidad a lo largo del tiempo

En los puntos anteriores se revisaron las dos arquitecturas más famosas y relevantes en el mundo del desarrollo de software actual, sin embargo, los factores como rendimiento y mantenibilidad a lo largo del tiempo hacen que la decisión sea clara por una, que es el enfoque escogido para realizar este sistema.

- **Rendimiento**

La arquitectura monolítica puede tener un rendimiento superior al estar todo en una única ejecución, no es comparación para considerar que los sistemas informáticos crecen con el tiempo, aumentando su nivel de usuarios, funcionalidades y datos, mientras que la arquitectura por capas, si es configurada correctamente, ofrece un desempeño similar, ofreciendo una escalabilidad y flexibilidad que los monolitos simplemente no entregan. La capacidad de ser escalable en la actualidad es sumamente importante, y se pueden hacer por las capas que se necesiten así no afectando el resto del sistema.

- **Mantenimiento**

En términos de mantenimiento, los monolitos requieren un despliegue completo para cada cambio, lo que dificulta las actualizaciones y modificaciones al sistema, ya que es un único código base, esto puede llevar a tiempos de inactividad significativos si no se gestiona adecuadamente. Mientras que la arquitectura por capas permite las actualizaciones independientes por capas, lo que facilita enormemente el mantenimiento y permite realizar cambios sin afectar a toda la aplicación, lo que es especialmente beneficioso para sistemas grandes y complejos.

Por tanto, mientras los monolitos son más simples y rápidos de desarrollar, quedan cortos al momento de mantenibilidad y modularidad a lo largo del tiempo.

7. Implementación

La implementación de este sistema está basada en la arquitectura por capas vista anteriormente, como se había mencionado, esta arquitectura tiene un tiempo más prolongado de desarrollo e integración, sin embargo, es más duradera en el tiempo y hace que las aplicaciones sean más robustas al momento de salir al público.

En esta sección, se justificarán las elecciones realizadas para armar el sistema.

7.1. Diferencias entre “On-Premise” y “en la nube”

La elección de dónde se va a encontrar alojado nuestro sistema es una parte vital de la arquitectura, y tiene factores como escalabilidad, costos y mantenibilidad asociados. A continuación, se presenta una comparativa entre ambas posibilidades.

7.1.1. Infraestructura On-Premise

On-Premise es el nombre con el cual se identifican servicios almacenados físicamente en una ubicación conocida por la empresa, por ejemplo, almacenado en servidores en una oficina física donde se puede ver, modificar e interactuar.

Algunas características de esta infraestructura son:

- Son instalaciones físicas dentro de la empresa.
- Existe un control total sobre hardware y datos.
- Acceso local a los servidores, puede llegar a requerir VPN para el acceso remoto.
- Una escalabilidad difícil y costosa, que requiere la compra de hardware adicional.
- Requiere personal interno para mantenimiento

Este tipo de infraestructura es algo que se ve en empresas que tienen sistemas antiguos, o privados, por lo que prefieren mantenerla cerrada y cerca para posibles modificaciones.

Algunos pros y contras se ven a continuación:

- Pros
 - Control total sobre la infraestructura y datos sensibles
 - Acceso offline para acceder a datos sin necesidad de conexión a internet
 - Costos a largo plazo, donde puede ser más rentable si se considera el costo total de la propiedad.
- Contras
 - Existe una inversión inicial alta, ya que son gastos significativos en la compra de hardware y software.
 - Conlleva una rigidez para escalar rápidamente y adaptarse a cambios en el negocio.
 - Tiene un mantenimiento costoso ya que requiere personal especializado y recursos para mantenimiento continuo.

7.1.2. Infraestructura en la nube

La infraestructura en la nube está siendo la más famosa, donde empresas como Google, Microsoft y Amazon AWS son los principales actores prestando servicios. Los servicios en nube son recursos informáticos que se proporcionan a través de internet, permitiendo a las empresas y usuarios acceder a diversas tecnologías y aplicaciones sin necesidad de gestionar infraestructura física. Estos servicios son escalables, flexibles y suelen seguir un modelo de pago por uso.

Algunas características de los servicios en nube:

- Costo inicial bajo a moderado, dado el modelo pago por uso.
- Una escalabilidad alta ya que los recursos se pueden ajustar fácilmente.
- Un mantenimiento gestionado por el proveedor.

- No requiere espacio para servidores en oficinas.
- Acceso desde cualquier parte con conexión a internet.

Esta infraestructura ha llegado a revolucionar en los últimos años a los desarrolladores y arquitectos, pudiendo subir y desplegar aplicaciones en días, sin la complejidad inicial de comprar recursos informáticos On-Premise.

Algunos pros y contras son:

- Pros
 - Tiene un bajo costo inicial ya que no se necesita gastar en hardware, lo que reduce significativamente el costo.
 - Contiene una escalabilidad rápida, donde permite aumentar o disminuir recursos rápidamente según las necesidades.
 - Se puede acceder desde cualquier lugar facilitando el trabajo remoto.
- Contrás
 - Existe un menor control ya que la seguridad y gestión depende del proveedor.
 - Dependen de internet para poder acceder a los servicios.
 - Riesgo de perder disponibilidad debido a fallo de los proveedores.

7.2. Proceso de implementación

En el punto anterior revisamos las infraestructuras más famosas que existen para implementar sistemas informáticos, y para este proyecto se decidió implementar la arquitectura con los servicios en nube, específicamente en Amazon AWS.

Para poder implementar el sistema completo, primero se trabajó en una maqueta donde se visualizan los distintos servicios que maneja AWS para poder armar una arquitectura robusta, esta se presenta a continuación.

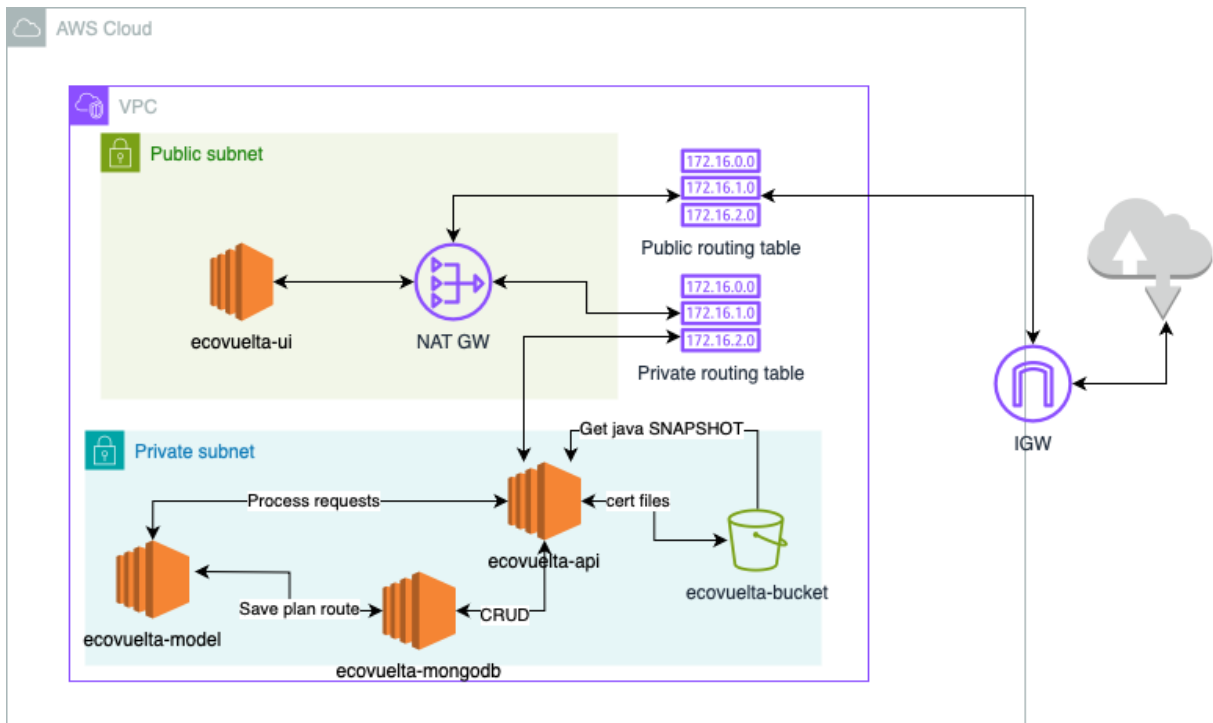


Figura 2. Representación gráfica la arquitectura en AWS
Fuente: Elaboración propia

Una vez teniendo el modelo de la arquitectura, se empezó a realizar una investigación acerca de la capa gratuita de AWS, la que consta en tener algunos servicios por 12 meses sin costos de mantención.

A continuación, se presentan en detalle los pasos y la arquitectura para el levantamiento:

- **Compra de dominio**

Dado que el nombre es Ecovuelta, fue necesario comprar un dominio nacional, o sea, que termine en “.cl”. Para esto, se realizó la compra del dominio en BlueHosting, obteniendo el acceso a la consola de administración donde podemos modificar cosas como el nombre, agregar certificados SSL, modificar los servidores de enrutamiento, etc.

- **Levantamiento de enrutamiento vía Route53**

Route53 es un servicio de Amazon AWS para poder alojar la zona del dominio adquirido en BlueHosting, aquí se puede dar un nombre a nuestras direcciones IP que van a ser presentadas más adelante. Debido a que son 4 direcciones IP que van a estar almacenadas en máquinas virtuales, se vincula el dominio con Route53, recibiendo el nombre de (ecovuelta.cl) en conjunto con subdominios para las otras aplicaciones que conforman la arquitectura (api.ecovuelta.cl, mongo.ecovuelta.cl). Estos registros almacenados son de tipo “A”, lo cual vincula nombre con dirección IP.

- **Levantamiento de grupos de seguridad**

Los grupos de seguridad sirven para filtrar el tráfico de los distintos puertos de las máquinas para prevenir conexiones no deseadas y posibles ataques. Estos grupos tienen reglas de

entrada, significa qué peticiones puedo realizar hacia la máquina, y de salida, que son qué peticiones puede realizar la máquina devuelta. En este punto se realizaron 4 grupos de seguridad que se explican a continuación.

Todas las máquinas exponen el puerto 22 (SSH) para su conexión exitosa con la máquina y realizar ajustes, instalaciones y actualizaciones.

El grupo de seguridad llamado “WEB” permite también conexiones por los puertos 80 y 443, o sea, HTTP y HTTPS respectivamente, lo que permite acceder a la página web desde cualquier buscador.

El grupo de seguridad llamado “API” permite conexiones por el puerto 8080, que es el puerto por defecto por donde salen y entran las peticiones, en conjunto con el puerto 8000, que también es para realizar integraciones entre dos microservicios. Estos son Java SpringBoot y FastAPI.

El grupo de seguridad llamado “MONGO” permite las conexiones por el puerto 27017, que es donde el servicio de base de datos mongo funciona.

- **Levantamiento de instancias EC2**

Las instancias EC2 son máquinas virtuales que viven dentro de Amazon AWS, las cuales se pueden armar con distintas imágenes, procesadores, almacenamiento y conectarlas a redes específicas, ya sean privadas o públicas.

Para la arquitectura, se tienen 4 máquinas virtuales, donde cada una tiene la funcionalidad de instalar y almacenar los códigos fuentes de los sistemas que conforman la arquitectura de EcoVuelta.

En la primera máquina llamada “ecovuelta-ui”, se expone la capa de presentación del sistema, donde la máquina entrega una IPV4 para ser expuesta al público. Esta máquina tiene el grupo de seguridad “WEB” explicado anteriormente.

En la segunda máquina llamada “ecovuelta-api”, se presenta la capa de lógica de negocios, la cual tiene todas las operaciones que se mandan a hacer desde la capa de presentación. Esta máquina tiene el grupo de seguridad “API”.

La tercera máquina llamada “ecovuelta-fastapi” presenta el modelo de optimización para la ayuda de planificaciones logísticas de las empresas de transporte, la cual recibe peticiones desde “ecovuelta-api”. Esta máquina también tiene el grupo de seguridad “API”.

Finalmente, una máquina llamada “ecovuelta-mongo”, la cual es la responsable de manejar las peticiones de base de datos de los servicios. Esta máquina tiene el grupo de seguridad “MONGO”.

En cada máquina se instalaron los lenguajes, servicios y frameworks necesarios para poder ser alojados en sus respectivos lugares, y poder funcionar en conjunto.

7.3. Tecnologías y lenguajes utilizados

En este punto se explicará a detalle los lenguajes y tecnologías utilizadas por cada capa de la arquitectura.

7.3.1. Capa de presentación

En la capa de presentación, se utilizó Angular 18.0, la última versión de angular, instalado desde NodeJS con sus node modules, en conjunto de diseños y estilos CSS, librería de Bootstrap para las dimensiones y uso de TypeScript para sus funcionalidades, se implementaron librerías como RxJS para realizar peticiones HTTP hacia la capa de negocios, y esperar respuestas con promesas.

7.3.2. Capa de lógica de negocio

Para la capa de negocios, existen dos sistemas involucrados, los cuales se presentarán a continuación:

- **Java 17 SpringBoot 3.3.4**

Para el microservicio principal se utilizó Java con el framework SpringBoot, donde se implementó una API REST capaz de manejar las solicitudes CRUD de varios modelos, para así ser almacenados en una base de datos utilizando la librería data mongo, que permite generar repositorios para hacer operaciones.

Lombok es utilizado para implementar anotaciones que reemplazan la necesidad de generar trabajo extra al momento de programar, como, por ejemplo, generar manualmente los getters y setters. Por otro lado, SpringBoot Security se utiliza para asegurar los endpoints que estarán expuestos al público, de esta forma previniendo ataques o solicitudes no deseadas.

Se expone una estructura de paquetes que dividen las funciones de Java para ofrecer un mayor control sobre sus funciones, y se separan en controladores, modelos, servicios, excepciones y configuraciones, donde cada uno cumple una parte de la estructura REST.

- **Python 3.x FastAPI**

El microservicio FastAPI es el encargado de exponer un endpoint que llama al modelo de optimización. Es el más importante al momento de almacenar una solicitud y que este empiece a procesarlas para generar las planificaciones de ruta. Consta con librerías como fastapi, que es la base principal del proyecto, pymongo que sirve para almacenar datos a la base de datos mongo que comparten ambos microservicios.

Mientras que el modelo de optimización está almacenado en un archivo aparte llamado OptimizationModel.py, el cual se ejecuta para realizar cálculos y mediciones, y toma decisiones acerca de qué solicitud es viable agregar a una planificación y cuáles no. Utiliza librerías como Pulp, requests, geopy y networkx, todo trabaja en conjunto para entregar los mejores resultados al cliente.

7.3.3. Capa de acceso a datos

La capa de acceso tiene utiliza una base de datos no relacional, o NoSQL, la cual es MongoDB. Esta base de datos utiliza colecciones para almacenar documentos que representan los registros, los cuales están armados en formato JSON.

Es una base de datos rápida, flexible y da la opción de manejar la información de forma rápida y completa, si bien ocupa un poco la redundancia, para efectos de este sistema la necesitamos para almacenar la información en distintos lados para poder llamar a los microservicios de una forma libre, y obtener todos los recursos que el cliente pide en la capa de presentación.

7.4. Desafíos y experiencias

Esta implementación se realizó pensando en la necesidad de adaptar el sistema de una forma conocida, esto es desarrollándolo con tecnologías conocidas mundialmente y también siendo robustas, ya que llevan años en el mercado. El stack Angular, SpringBoot y Mongo es utilizado en grandes empresas por lo que es perfecto para este sistema.

La necesidad de aprender estos lenguajes suma al desafío de los integrantes a trabajar en distintas áreas que no son su fuerte, sin embargo, la motivación por aprender y las ganas de desarrollar un sistema que tiene altas promesas, les permite realizar código limpio y bien estructurado. Una gran ayuda fueron las asignaturas universitarias con las que se contaron para aprender a realizar código en estos lenguajes, y la elección de cada participante mostró ser la correcta al ser asignados a lo que ellos sabían, y más importante, lo que les gustaba.

Una implementación en un servicio de nube como Amazon AWS no es sencillo, e incluye la necesidad de estudiar y aprender, especialmente desde la mirada de la arquitectura de software, de esto depende la funcionalidad del sistema y la alineación con términos que no son vistos o no se está familiarizado. Por eso, la explicación de cómo funciona la implementación está en detalle ya que existió un esfuerzo detrás para aprender y seguir mejorando en arquitecturas con soluciones en la nube.

El mayor desafío fue tomar todas estas aristas y hacerlas funcionar en un margen ágil, considerando las metodologías como Scrum y Kanban, las cuales fueron cruciales para poder realizar entregables a tiempo, e iteraciones que tengan valor para el cliente final.

8. Resultados y evaluación

Actualmente, el sistema no está bajo ningún estrés. Sin embargo, más adelante se enfrentará a un aumento de usuarios concurrentes realizando peticiones y el sistema deberá ser capaz de manejarlas para su óptimo funcionamiento. Para esto, se procedió a realizar pruebas de tiempo de respuesta de los microservicios, y cuánto demora en realizar algunas funcionalidades. Estas pruebas estarán enfocadas en un CRUD, en endpoints de diversas funcionalidades y en el servicio de procesamiento del modelo de optimización para ver la calidad de los tiempos y su evaluación con respecto a rendimiento, siendo las necesarias para saber con qué tiempos se está trabajando y si cumple con la norma vigente actual, en conjunto con los requerimientos no funcionales que tiene este sistema.

Las pruebas al microservicio de Java SpringBoot fueron realizadas con JMeter, una aplicación disponible para todos los OS, que realiza pruebas de estrés con la aplicación. Esto, además de dar un estimado de

cuánto se demora el servicio en responder, ayuda a entender la escalabilidad que tiene el sistema, ya que, si el volumen de peticiones fuese más alto, lo más probable es que se necesite más potencia ya sea escalando en forma horizontal o vertical.

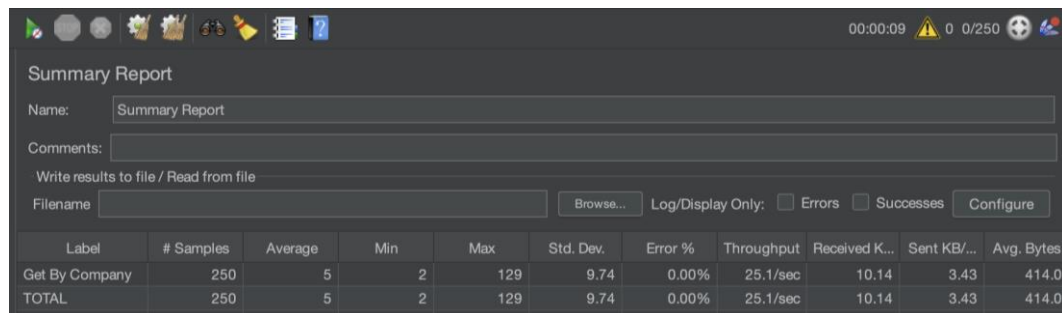
Para los requerimientos no funcionales, se pensó en el primer año de funcionamiento, donde existiría un máximo de 250 peticiones concurrentes entre usuarios, ya sea realizando variadas acciones como agregando solicitudes, yendo a buscar solicitudes, registrándose en la aplicación, realizando login, agregando vehículos, conductores, ubicaciones, etc. Esto, en conjunto con la norma que son aproximadamente 100 ms para devolver una respuesta adecuada por petición, es nuestro estándar por el cual nos guiaremos a realizar estas pruebas. El sistema en el que se corrieron estas pruebas es un Macbook M3 Pro con 18 GB de RAM, OS Sequoia 15.

8.1. Pruebas de estrés sobre microservicios.

8.1.1. Request Controller

Este controlador está encargado de recibir peticiones referentes a las solicitudes de movimiento de carga, lo que lo hace uno de los principales controladores de nuestro sistema.

- **Get requests by company:** Recibe el parámetro company, para entregar las solicitudes por cada empresa. Se promedió 5 ms entre cada solicitud, dentro de un rango de 2 ms y 129 ms.



Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

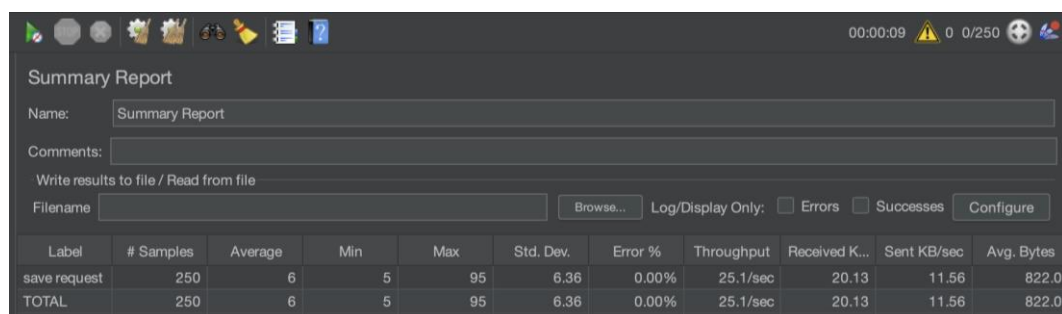
Filename: Browse... Log/Display Only: ☐ Errors ☐ Successes

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received K...	Sent KB/sec	Avg. Bytes
Get By Company	250	5	2	129	9.74	0.00%	25.1/sec	10.14	3.43	414.0
TOTAL	250	5	2	129	9.74	0.00%	25.1/sec	10.14	3.43	414.0

Figura 3. Resultado de prueba de estrés en endpoint getByCompany

Fuente: Apache JMeter (5.6.3)

- **Save request:** Este endpoint recibe un cuerpo que es necesario para guardar las solicitudes y enviarlas al modelo de optimización. Cabe recordar que este endpoint en particular tiene la particularidad de activar un proceso asíncrono que se desencadena en el microservicio FastAPI, por lo tanto, no deja tomado el servicio y es capaz de responder con una media de 6 ms por solicitud, donde el tiempo mínimo fueron 5 ms y el máximo 95 ms.



Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

Filename: Browse... Log/Display Only: ☐ Errors ☐ Successes

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received K...	Sent KB/sec	Avg. Bytes
save request	250	6	5	95	6.36	0.00%	25.1/sec	20.13	11.56	822.0
TOTAL	250	6	5	95	6.36	0.00%	25.1/sec	20.13	11.56	822.0

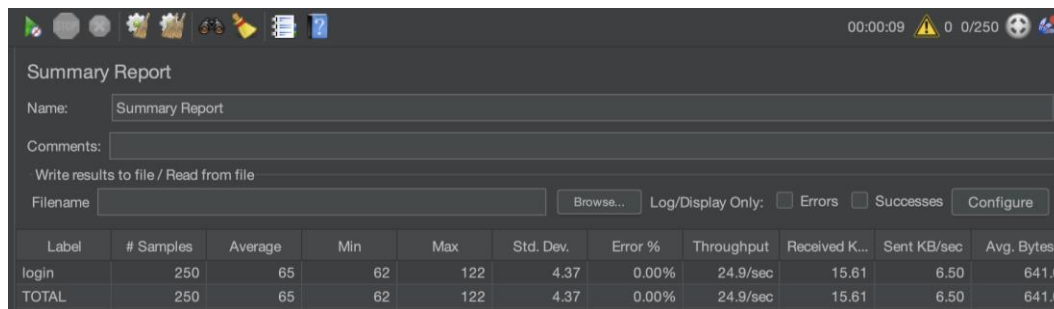
Figura 4. Resultado de prueba de estrés en endpoint saveRequest

Fuente: Apache JMeter (5.6.3)

8.1.2. Auth Controller

Este controlador está encargado de autenticar a los usuarios que ya están registrados e intentan acceder a la página web.

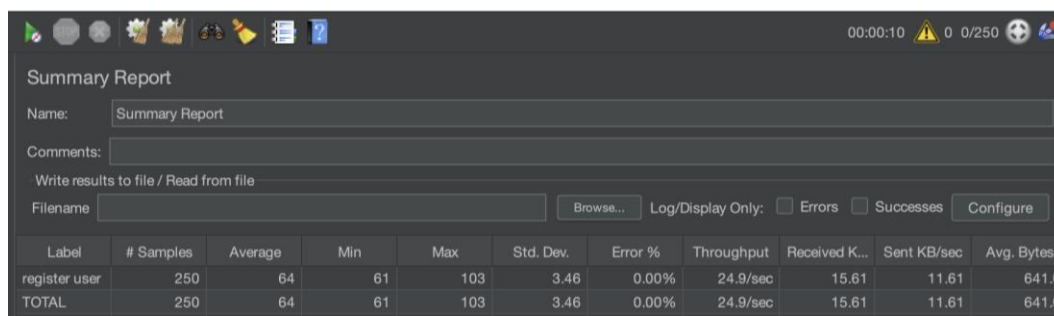
- **Login:** Este endpoint recibe un cuerpo que envía el correo y contraseña en texto plano, para lograr obtener un token JWT en conjunto con datos relevantes a la cuenta. Los resultados tienen una pequeña elevación debido a que se realiza una validación de la contraseña contra la base de datos, y si es correcta, se genera un token JWT para realizar solicitudes al microservicio. Se promedian 65 ms por solicitud, con un mínimo de 62 ms y máximo de 122 ms.



Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received K...	Sent KB/sec	Avg. Bytes
login	250	65	62	122	4.37	0.00%	24.9/sec	15.61	6.50	641.0
TOTAL	250	65	62	122	4.37	0.00%	24.9/sec	15.61	6.50	641.0

Figura 5. Resultado de prueba de estrés en endpoint login
Fuente: Apache JMeter (5.6.3)

- **Register User:** Este endpoint es responsable por la creación de las cuentas de la aplicación, la cual recibe un cuerpo que arma la estructura de un usuario. La razón por la cual está bajo el punto de auth es porque se protege la contraseña utilizando BCrypt para generar un hash. Los resultados se deben a la validación de la contraseña de texto plano a hash. Se promedian 64 ms por solicitud, siendo la mínima 61 ms y máxima 103 ms.



Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received K...	Sent KB/sec	Avg. Bytes
register user	250	64	61	103	3.46	0.00%	24.9/sec	15.61	11.61	641.0
TOTAL	250	64	61	103	3.46	0.00%	24.9/sec	15.61	11.61	641.0

Figura 6. Resultado de prueba de estrés en endpoint register user
Fuente: Apache JMeter (5.6.3)

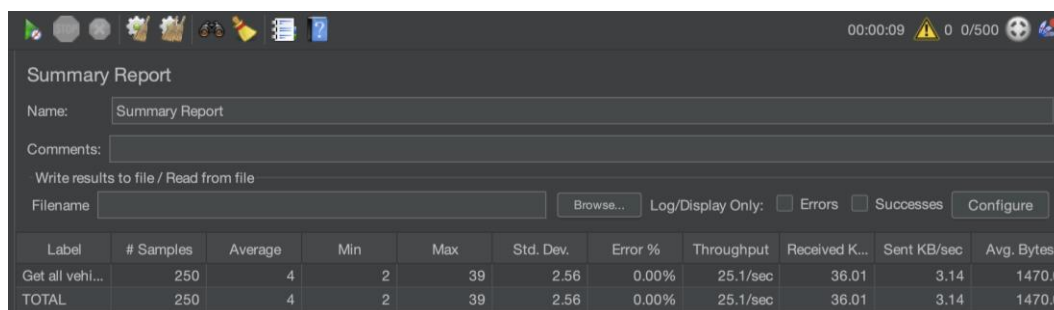
8.1.3. Vehicle Controller

El controlador vehicle controller apunta a la colección VehicleCollection de la base de datos, esta es responsable de actuar como un CRUD para las operaciones de almacenar, leer, editar y borrar vehículos. Debido que existen distintos CRUD, se tomarán este de referencia ya que no existen mayores cambios entre ellos, además del nombre de su colección.

Cabe destacar que la acción de borrar de los distintos CRUD se reemplazó por un *soft delete* las cuales no borran el registro, sino que cambian un campo llamado *active* para diferenciar los borrados con los activos, por tanto, se mantienen los registros y se almacena su historial.

A continuación, se muestran las pruebas de estrés del controlador de vehículos.

- **Get all vehicles:** Este endpoint trae todos los vehículos de la aplicación, para efectos de tener un control de los vehículos más comunes o configuraciones más frecuentes. Dado que este es una lista general de los vehículos, su respuesta es en promedio 4 ms, con una mínima de 2 ms y máxima de 39 ms.



Summary Report

Name: Summary Report

Comments:

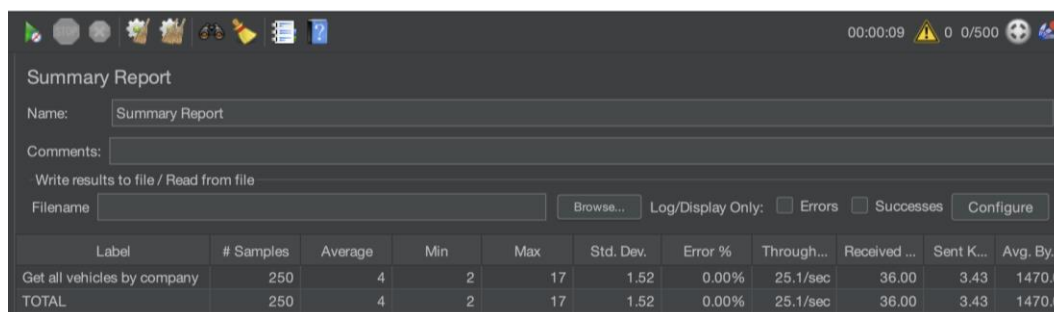
Write results to file / Read from file

Filename: Browse... Log/Display Only: ☐ Errors ☐ Successes

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received K...	Sent KB/sec	Avg. Bytes
Get all vehi...	250	4	2	39	2.56	0.00%	25.1/sec	36.01	3.14	1470.0
TOTAL	250	4	2	39	2.56	0.00%	25.1/sec	36.01	3.14	1470.0

Figura 7. Resultado de prueba de estrés en endpoint getAllVehicles
Fuente: Apache JMeter (5.6.3)

- **Get vehicles by company:** Este endpoint trae todos los vehículos por empresa para mostrar a los usuarios que sean trabajadores de aquella. Los resultados son un promedio de 4 ms, con una mínima de 2 ms y un máximo de 17 ms.



Summary Report

Name: Summary Report

Comments:

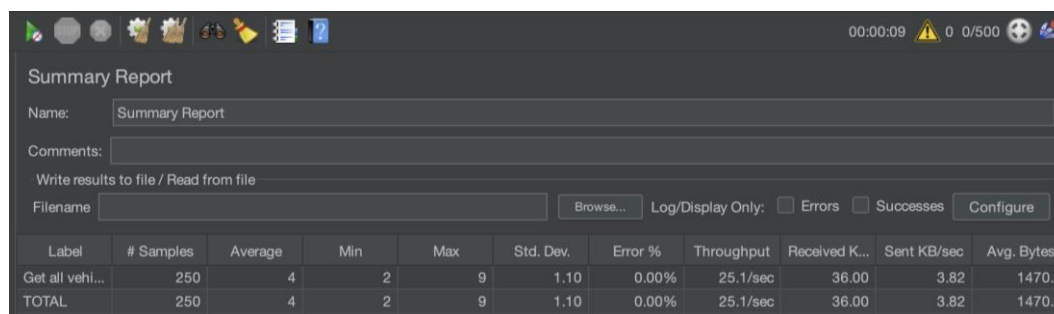
Write results to file / Read from file

Filename: Browse... Log/Display Only: ☐ Errors ☐ Successes

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Through...	Received ...	Sent K...	Avg. By...
Get all vehicles by company	250	4	2	17	1.52	0.00%	25.1/sec	36.00	3.43	1470.0
TOTAL	250	4	2	17	1.52	0.00%	25.1/sec	36.00	3.43	1470.0

Figura 8. Resultado de prueba de estrés en endpoint getAllVehiclesByCompany
Fuente: Apache JMeter (5.6.3)

- **Get vehicle by id:** Este endpoint obtiene un vehículo con su id, y es necesario para asignar una planificación a un vehículo. Los resultados promedian 4ms por solicitud, teniendo una mínima de 2 ms y máxima de 9 ms.



Summary Report

Name: Summary Report

Comments:

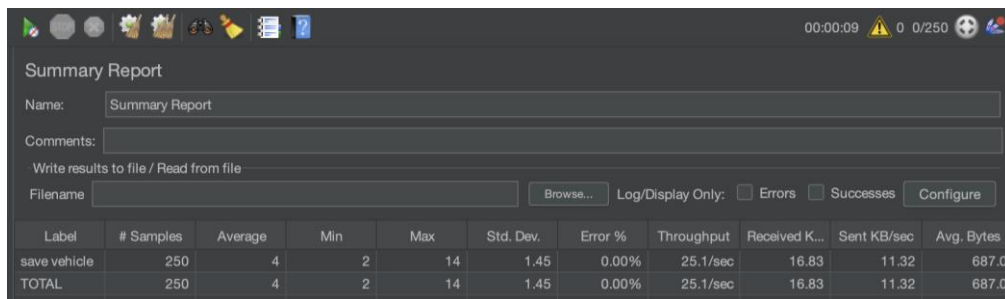
Write results to file / Read from file

Filename: Browse... Log/Display Only: ☐ Errors ☐ Successes

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received K...	Sent KB/sec	Avg. Bytes
Get all vehi...	250	4	2	9	1.10	0.00%	25.1/sec	36.00	3.82	1470.0
TOTAL	250	4	2	9	1.10	0.00%	25.1/sec	36.00	3.82	1470.0

Figura 9. Resultado de prueba de estrés en endpoint getAllVehiclesById
Fuente: Apache JMeter (5.6.3)

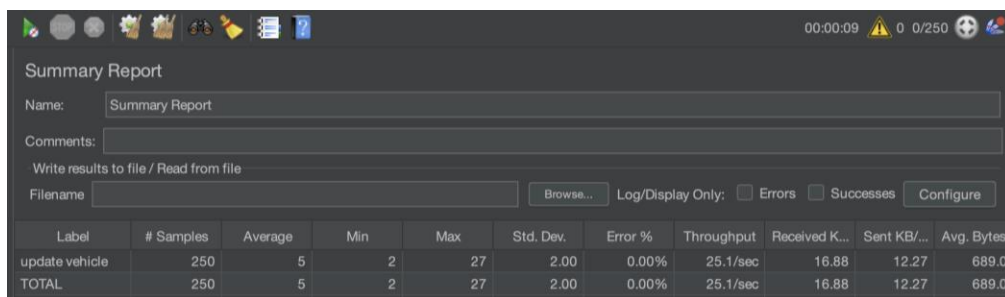
- **Save vehicle:** Este endpoint es el responsable de almacenar a los vehículos que se registran en el formulario que aparece en la capa de presentaciones. El promedio de las solicitudes tuvo una respuesta de 4 ms, con un mínimo de 2 ms y un máximo de 14 ms.



Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received K...	Sent KB/sec	Avg. Bytes
save vehicle	250	4	2	14	1.45	0.00%	25.1/sec	16.83	11.32	687.0
TOTAL	250	4	2	14	1.45	0.00%	25.1/sec	16.83	11.32	687.0

Figura 10. Resultado de prueba de estrés en endpoint saveVehicle
Fuente: Apache JMeter (5.6.3)

- **Update vehicle:** Este endpoint está encargado de actualizar la información de un vehículo dado su id. Los resultados en promedio de actualización fueron de 5 ms, con una mínima de 2 ms y una máxima de 27 ms.



Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received K...	Sent KB/sec	Avg. Bytes
update vehicle	250	5	2	27	2.00	0.00%	25.1/sec	16.88	12.27	689.0
TOTAL	250	5	2	27	2.00	0.00%	25.1/sec	16.88	12.27	689.0

Figura 11. Resultado de prueba de estrés en endpoint updateVehicle
Fuente: Apache JMeter (5.6.3)

8.1.4. FastAPI Controller

El modelo de optimización está almacenado en un repositorio, donde su archivo main.py es el levantamiento del microservicio FastAPI. Este servicio tiene la particularidad de ser una función asíncrona que proviene desde el microservicio Springboot, y llama a un subproceso con los campos que se le entrega al endpoint. Este endpoint es el responsable de hacer correr al modelo, para calcular la optimización de las rutas, y tiene distintos pasos que requieren procesamiento. Los dos pasos más importantes son procesar las distancias entre los puntos de borde, y el guardado a la base de datos, y es aquí donde se genera el mayor tiempo de demora del modelo.

En el caso del cálculo de distancias, no es el único valor que se está procesando, sino que también el tiempo y el costo. Esto tiene una variación entre 26 y 27 segundos, los cuales no son problema ya que el modelo es asíncrono. Se muestra una figura a continuación.

```
(venv) diegomunoz@MAC ecovuelta-model % python calculos.py
Calculando costos y distancias...
El tiempo de ejecución de calculos.py fue: 27.835250 segundos
{('Arica', 'Iquique'): 309.0, ('Iquique', 'Pozo Almonte'): 56.0, ('Iquique', 'Antofagasta'): 416, ('Antofagasta', 'Copiapo'): 537.0, ('Copiapo', 'Coquimbo'): 470.0, ('Coquimbo', 'Valparaiso'): 368.0, ('Valparaiso', 'San Antonio'): 223.0, ('Valparaiso', 'RM'): 179.0, ('San Antonio', 'RM'): 118.0, ('RM', 'Rancagua'): 79, ('San Antonio', 'Rancagua'): 139.0, ('Rancagua', 'Talca'): 172.0, ('Talca', 'Concepcion'): 248.0, ('Talca', 'Talcahuano'): 252.0, ('Talcahuano', 'Concepcion'): 18.0, ('Concepcion', 'Temuco'): 292.0, ('Temuco', 'Valdivia'): 168.0, ('Valdivia', 'Puerto Montt'): 214.0, ('Puerto Montt', 'Punta Arenas'): 2174.0}
(venv) diegomunoz@MAC ecovuelta-model %
```

Figura 12. Tiempo de procesamiento de distancia, costo y tiempo entre bordes
Fuente: Python bash

Por el otro lado, tenemos el almacenamiento de las rutas en las bases de datos, esto corresponde a 5 segundos en los que el proceso asíncrono **si** espera ciertos datos que necesitan un id almacenado. Se logra apreciar que el tiempo de ejecución fue de 32 segundos, restando los 27 segundos anteriores, obtenemos el tiempo mencionado anteriormente. La diferencia de tiempo se muestra a continuación.

```
El tiempo de ejecución fue: 31.597479 segundos
::[{refId: '6729a29cb978c187fa8f1e0d', 'origin': 'RM', 'stops': 2, 'destination': 'Coquimbo', 'certifications': 'VIALIDAD', 'weight': 1, 'configType': '6x4', 'vehicleType': 'CAMION', 'maxCapacity': 50, 'alive': 'True', 'costs': 156058.0, 'price': 240000, 'earnings': 83942.0}, {refId: '6729a29cb978c187fa8f1e0d', 'origin': 'RM', 'stops': 1, 'destination': 'Valparaiso', 'loadType': 'Granilla', 'certifications': 'VIALIDAD', 'weight': 1, 'configType': '6x4', 'vehicleType': 'CAMION', 'maxCapacity': 50, 'alive': 'True', 'costs': 51068.0, 'price': 90000, 'earnings': 38932.0}, {refId: '6729a29cb978c187fa8f1e0d', 'origin': 'Concepcion', 'stops': 1, 'destination': 'Temuco', 'loadType': 'Productos Forestales', 'certifications': 'VIALIDAD', 'weight': 2, 'configType': '6x4', 'vehicleType': 'CAMION', 'maxCapacity': 50, 'alive': 'True', 'costs': 83307.0, 'price': 150000, 'earnings': 66693.0}, {refId: '673ac88d565ba81db81a770a', 'origin': 'Valparaiso', 'stops': 1, 'destination': 'Coquimbo', 'loadType': 'GRANEL', 'certifications': 'ISO22000', 'weight': 10, 'configType': '6x4', 'vehicleType': 'CAMION', 'maxCapacity': 50, 'alive': 'True', 'costs': 104990.0, 'price': 150000, 'earnings': 45010.0}]]
::[{refId: '6729a29cb978c187fa8f1e0d', 'origin': 'RM', 'destination': 'Valparaiso', 'certifications': 'VIALIDAD', 'loadType': 'Granilla', 'weight': 1, 'configType': '6x4', 'vehicleType': 'CAMION', 'maxCapacity': 50, 'alive': 'True', 'costs': 51068, 'price': 90000, 'earnings': 38932, 'startDate': '2024-10-14T07:00:00+00:00', 'endDate': '2024-10-14T15:00:00+00:00', 'order': 1}, {refId: '673ac88d565ba81db81a770a', 'origin': 'Valparaiso', 'destination': 'Coquimbo', 'certifications': 'ISO22000', 'loadType': 'GRANEL', 'weight': 10, 'configType': '6x4', 'vehicleType': 'CAMION', 'maxCapacity': 50, 'alive': 'True', 'costs': 104990, 'price': 150000, 'earnings': 45010, 'startDate': '2024-11-21T08:00:00+00:00', 'endDate': '2024-11-24T12:00:00+00:00', 'order': 2}]]
```

Figura 13. Tiempo de procesamiento y guardado en base de datos
Fuente: Python bash

8.2. Evaluación de rendimiento

Con estas pruebas, podemos ver que, dado las normas descritas anteriormente, el microservicio está realizando y contestando todas las peticiones con una velocidad considerada excelente. Sin embargo, el controlador de FastAPI que es aplicación del modelo de optimización, está teniendo problemas para procesar rápidamente las rutas, llegándose a demorar aproximadamente 30 segundos en completar la selección de las solicitudes aptas, esto sería preocupante ya que no está dentro de lo que se planificó en las normas, pero están siendo controladas con un proceso asíncrono.

El proceso asíncrono hace que se ingrese una ruta, devuelva la petición al cliente y deje disponible la aplicación para tomar otra petición, y así sucesivamente, mientras que la aplicación por detrás está realizando código en un hilo aparte, sin importarle a cómo se comporte el modelo (si es que falla o no).

De esta forma, estamos validando que el sistema no se congele al momento de hacer una solicitud, y se puede seguir navegando libremente.

Esto puede ser resuelto de forma fácil, que se explicará en la sección de áreas de mejora para su mayor entendimiento.

8.3. Identificación de fortalezas

La aplicación actualmente está respondiendo con valores muy positivos, a un punto donde se considera óptima, es por eso por lo que cabe recalcar las siguientes fortalezas del sistema:

- **Bajo tiempo de respuesta:** Un promedio aproximado de 20 ms (tomando en consideración todos los resultados promedios de los endpoints probados) es un número importante pensando en el número mínimo de espera de los usuarios actuales (100 ms)
- **Manejo eficiente de concurrencia:** Tener la capacidad de manejar 250 solicitudes de forma concurrente sin problemas sugiere que la aplicación es robusta y que la arquitectura está bien diseñada.
- **Estabilidad:** No existieron errores realizando las pruebas de estrés.

- **Asíncrono:** Los servicios funcionan con hilos separados para los subprocesos, lo que permite generar una experiencia grata sin necesidad de estar esperando que termine de procesar y vuelva a estar disponible el microservicio.

8.4. Áreas de mejora

Si bien se ha buscado optimizar la aplicación, hay algunas áreas que considerar que puede ayudar a seguir probando el rendimiento de la arquitectura, y así prevenir caídas o imprevistos cuando la aplicación alcance niveles más altos de usuarios y funcionalidades. Algunas de estas áreas son:

- **Pruebas con carga extrema:** Realizar pruebas de estrés con un mayor número de usuarios concurrentes (500 – 1000 usuarios) para identificar el punto donde el rendimiento comienza a degradarse.
- **Monitoreo y logging:** Implementar herramientas y técnicas avanzadas de monitoreo y logs para capturar métricas de tiempo real durante las pruebas y en producción. Esto permite atacar los problemas antes que lleguen al usuario final.
- **Optimización de código:** Revisar el código donde se puedan implementar mejoras, reducir llamadas innecesarias o mejorar la latencia entre componentes.
- **Escalabilidad horizontal:** Agregar más máquinas para soportar la llegada de más usuarios si se llegase a necesitar.
- **Pruebas de resiliencia:** Realizar pruebas que tengan que ver con la recuperación del fallo de algún componente.
- **Manejo de datos:** En el modelo de optimización, el proceso que más se demora es el cálculo de distancia, costos y tiempo, por lo que sería recomendable habilitar una colección en la base de datos que almacene estos valores para reducir el tiempo de procesamiento.

9. Conclusiones y recomendaciones

La responsabilidad de un arquitecto de software es importante ya que conlleva distintos aspectos del funcionamiento, desde la declaración del problema, pensar en la solución con el equipo, y finalmente los aspectos de rendimiento, escalabilidad y flexibilidad del sistema. La experiencia ha sido la mayor ayuda al momento de diseñar este sistema, con tal de armar una solución óptima que entregue la mejor experiencia de usabilidad a los usuarios finales.

Al momento de terminar el MVP, se utilizaron tecnologías de alta calidad, tales como Angular, Springboot, FastAPI y MongoDB, los cuales tienen una proyección de mantenimiento, documentación y mejora continua por un largo tiempo. Esto significa que las decisiones tomadas en el transcurso del desarrollo fueron correctas, pudiendo ser validadas con clientes finales y también realizando pruebas funcionales contra los servicios.

La arquitectura de Ecovuelta tiene como foco principal la relevancia e importancia de un diseño bien estructurado y fundamentado en principios sólidos. La aplicación de patrones arquitectónicos adecuados, como lo es el patrón por capas y enfocado en microservicios, permite crear un sistema que no solo es eficiente en términos de procesamiento y respuesta, sino también escalable y mantenible a largo plazo.

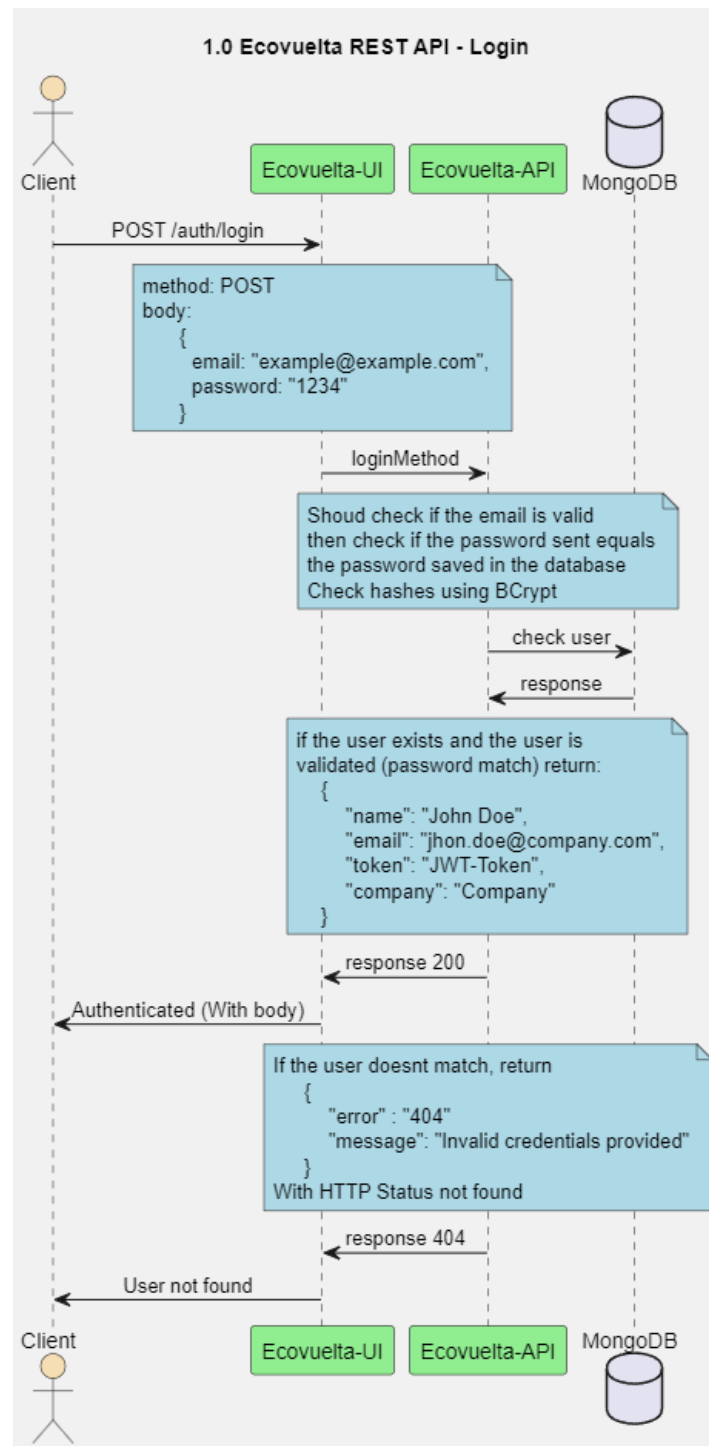
El trabajo ágil realizado con metodologías como Scrum y Kanban facilita una colaboración efectiva entre el equipo de desarrollo, promoviendo una comunicación fluida y una evaluación continua en lo que fue el progreso del proyecto, contribuyendo a una mejor calidad del software, y dando la oportunidad de dar ajustes rápidos ante imprevistos o cambios en el negocio.

En un contexto donde se busca garantizar una buena arquitectura, es importante velar por cumplir los aspectos no funcionales del sistema, tales como la concurrencia de usuarios, el soporte y la disponibilidad. Esto debido a que se debe pensar y entender al cliente, logrando de esta forma un sistema que beneficie a todas las partes.

Finalmente, se comprueba que invertir tiempo y recursos en una arquitectura bien definida no solo es una decisión estratégica inteligente, sino que también es fundamental para garantizar el éxito y la sostenibilidad del sistema a largo plazo. Habiendo cumplido con los puntos más importantes de este proyecto, como el rendimiento, flexibilidad, testabilidad, modularidad, escalabilidad y otros, se puede decir que el sistema es robusto y seguro para su futuro uso ante los clientes finales.

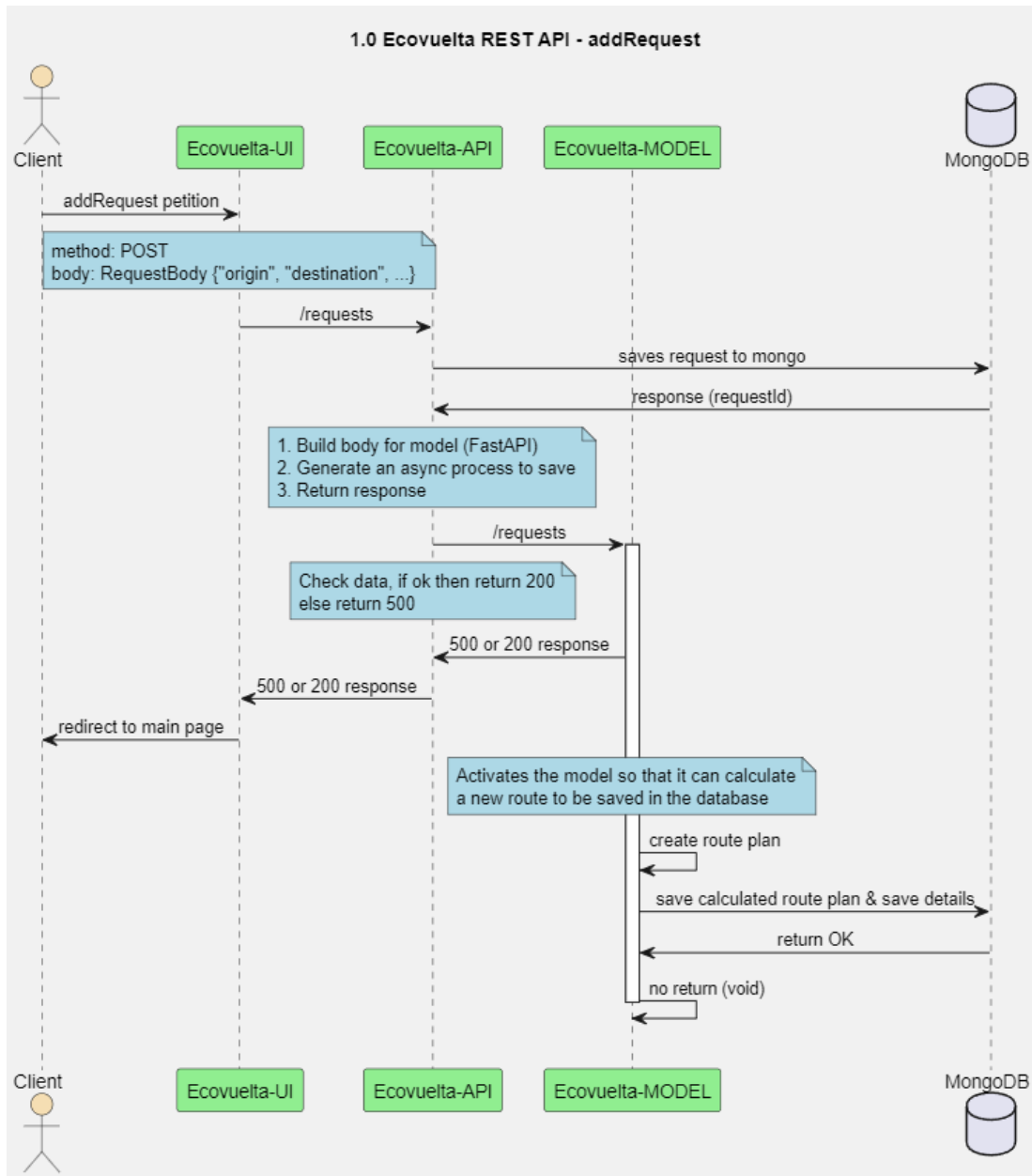
10. Anexos

10.1. Anexo A



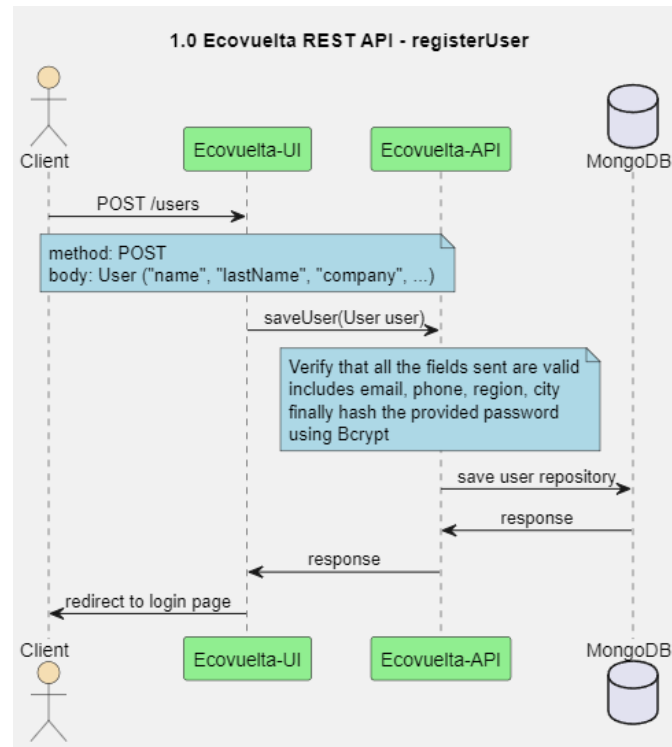
Anexo A. Diagrama de secuencia para funcionalidad de login
Fuente: Elaboración propia

10.2. Anexo B



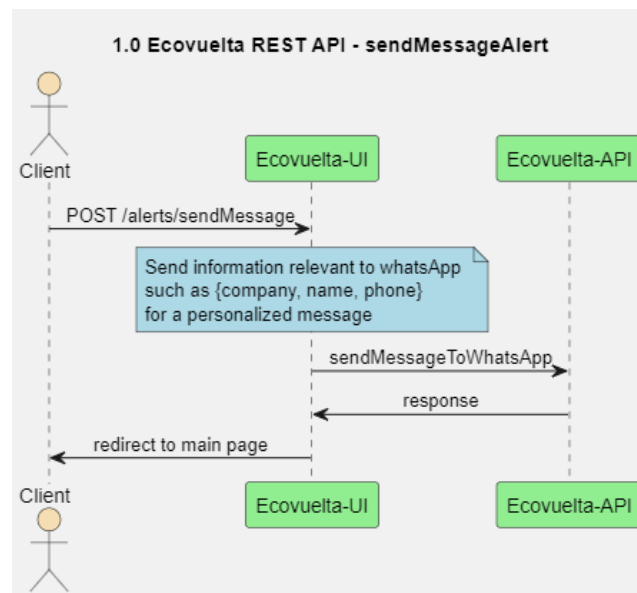
Anexo B. Diagrama de secuencia de funcionalidad agregar solicitud
Fuente: Elaboración propia

10.3. Anexo C



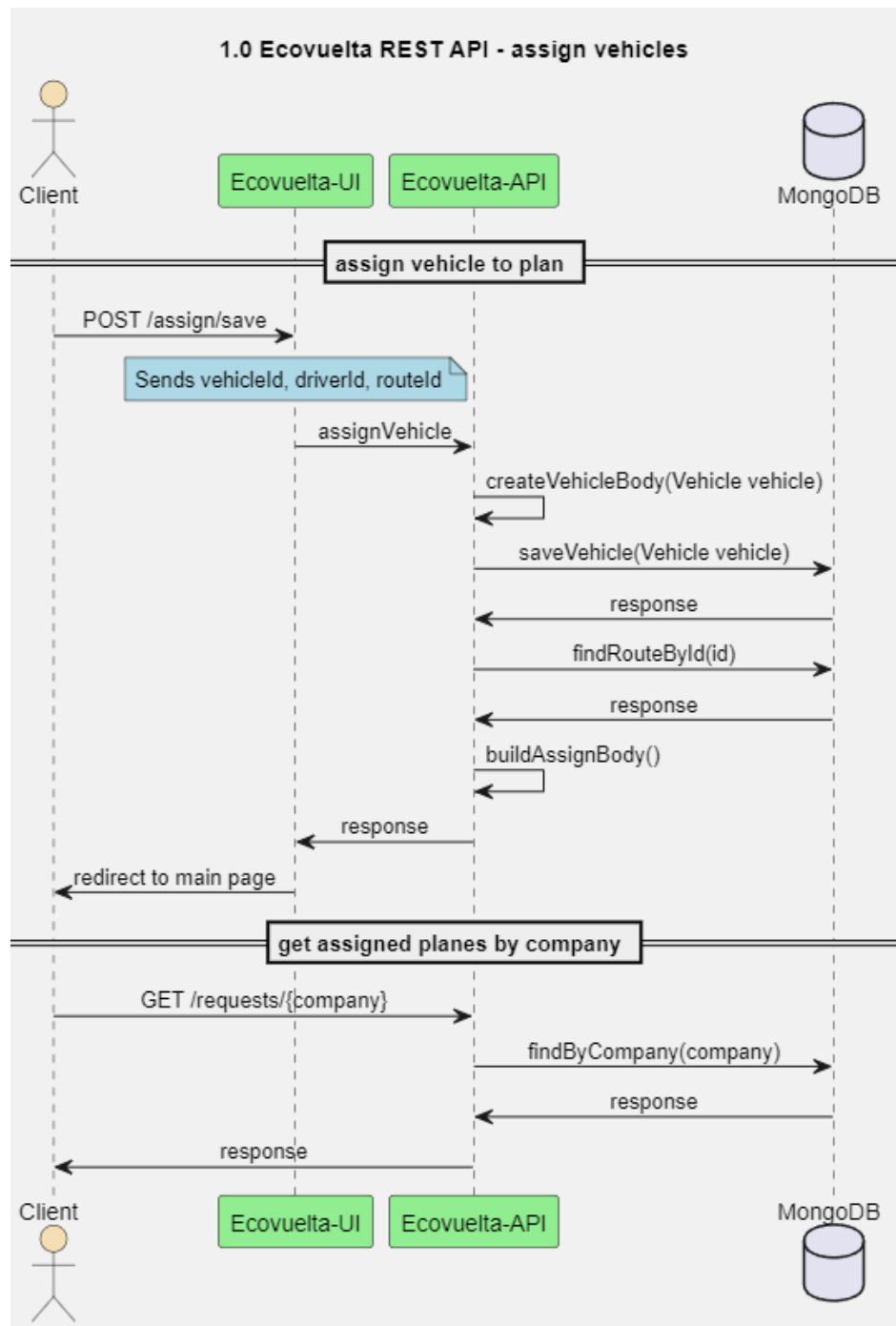
Anexo C. Diagrama de secuencia de la funcionalidad registrar usuario
Fuente: Elaboración propia

10.4. Anexo D



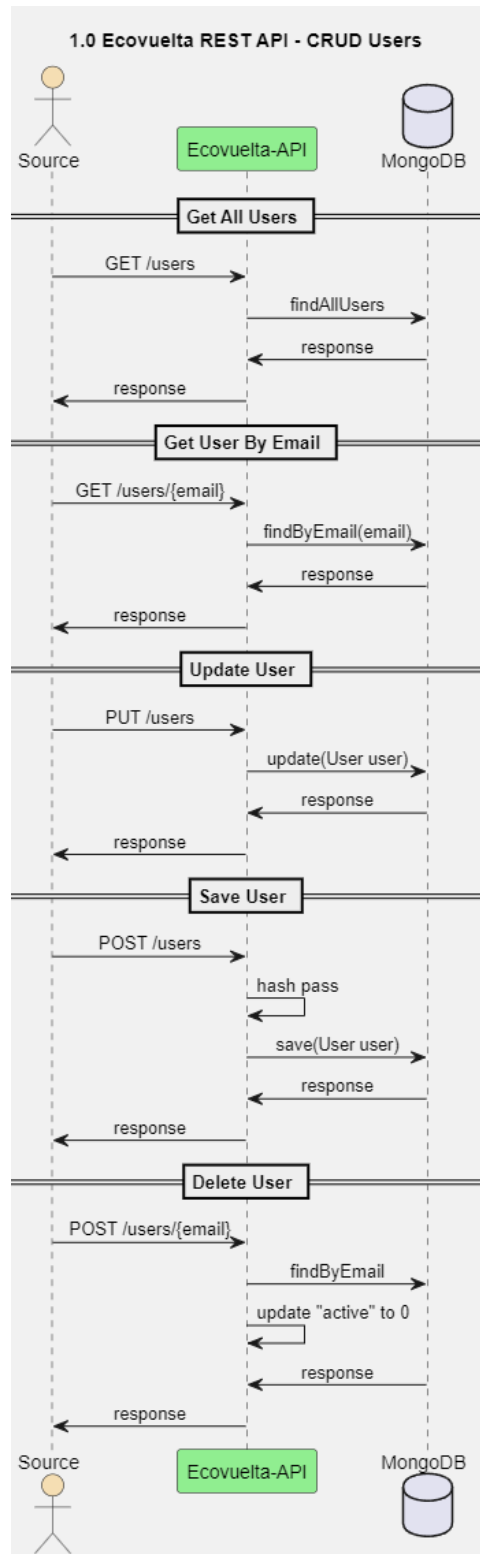
Anexo D. Diagrama de secuencia enviar mensaje por WhatsApp
Fuente: Elaboración propia

10.5. Anexo E



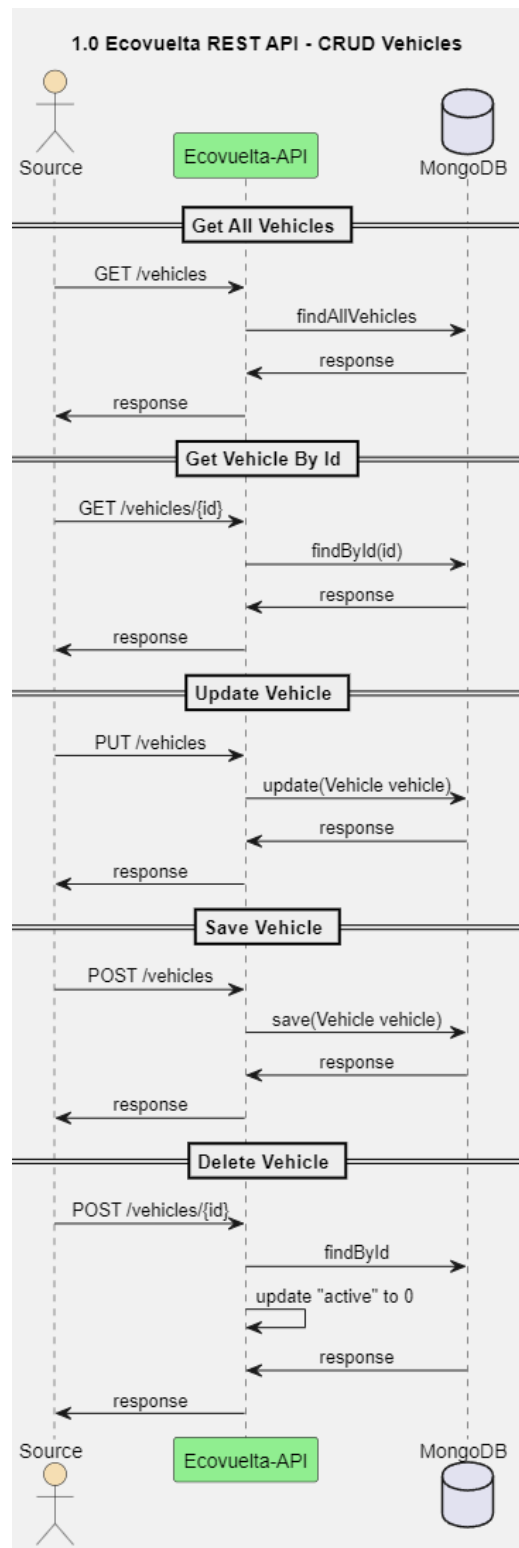
Anexo E. Diagrama de secuencia enviar mensaje por WhatsApp
Fuente: Elaboración propia

10.6. Anexo F



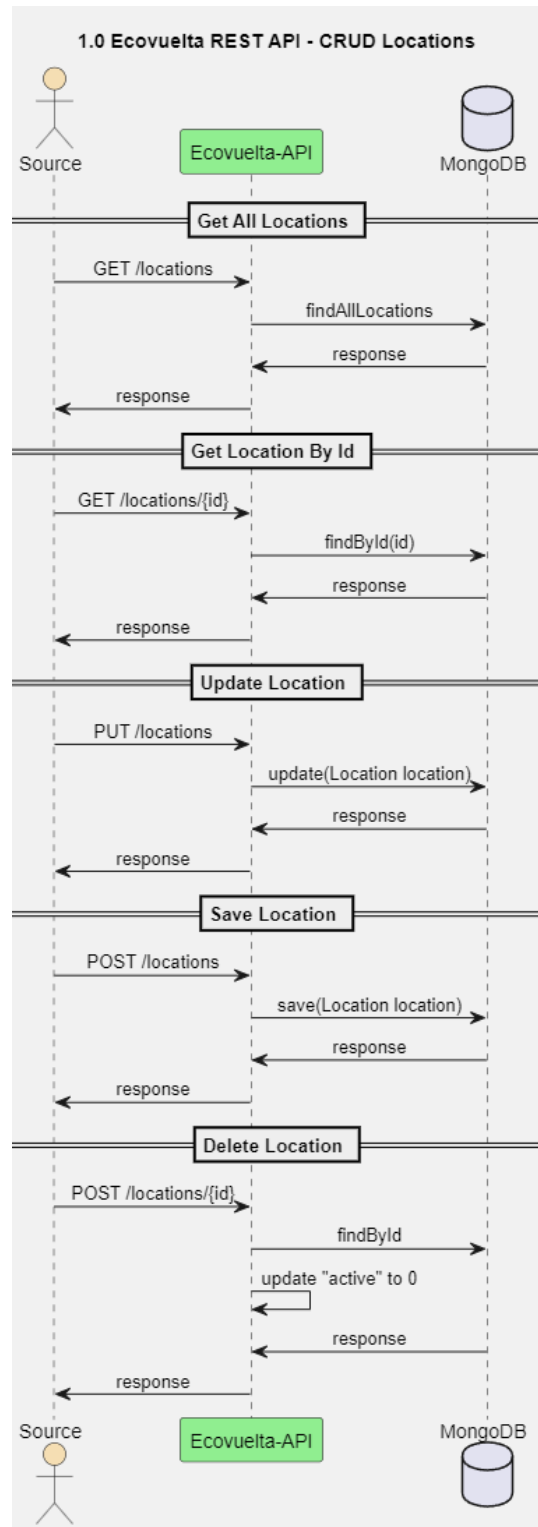
Anexo F. Diagrama de secuencia CRUD Usuarios
Fuente: Elaboración propia

10.7. Anexo G



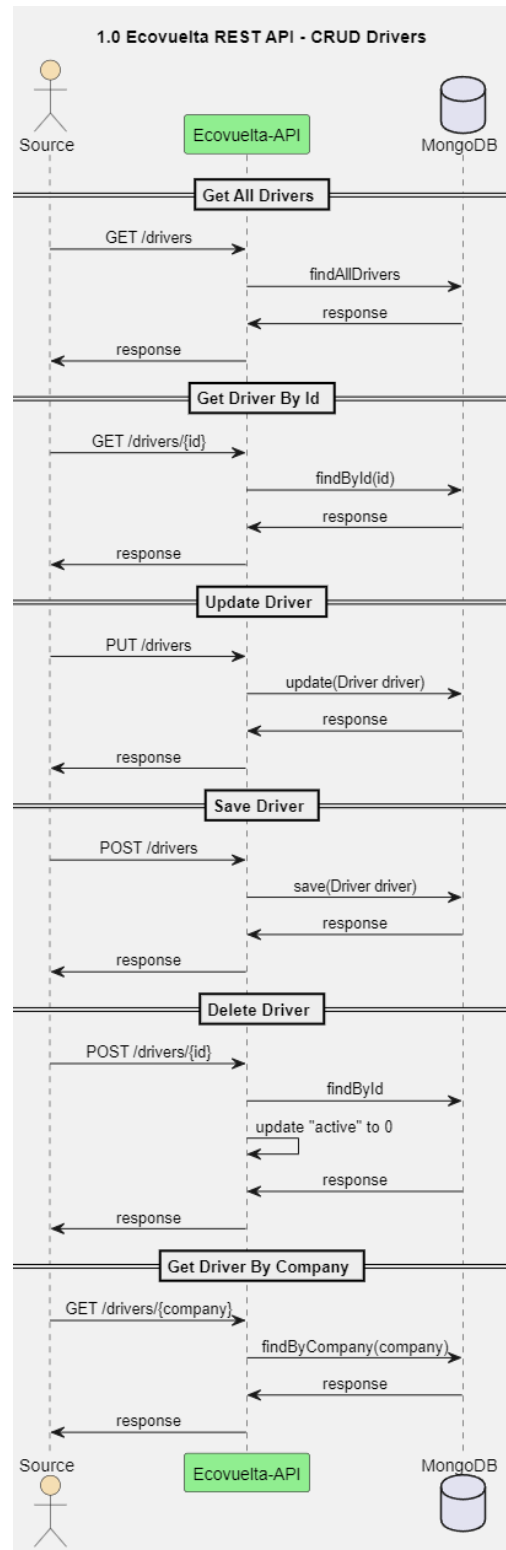
Anexo G. Diagrama de secuencia CRUD Vehículos
Fuente: Elaboración propia

10.8. Anexo H



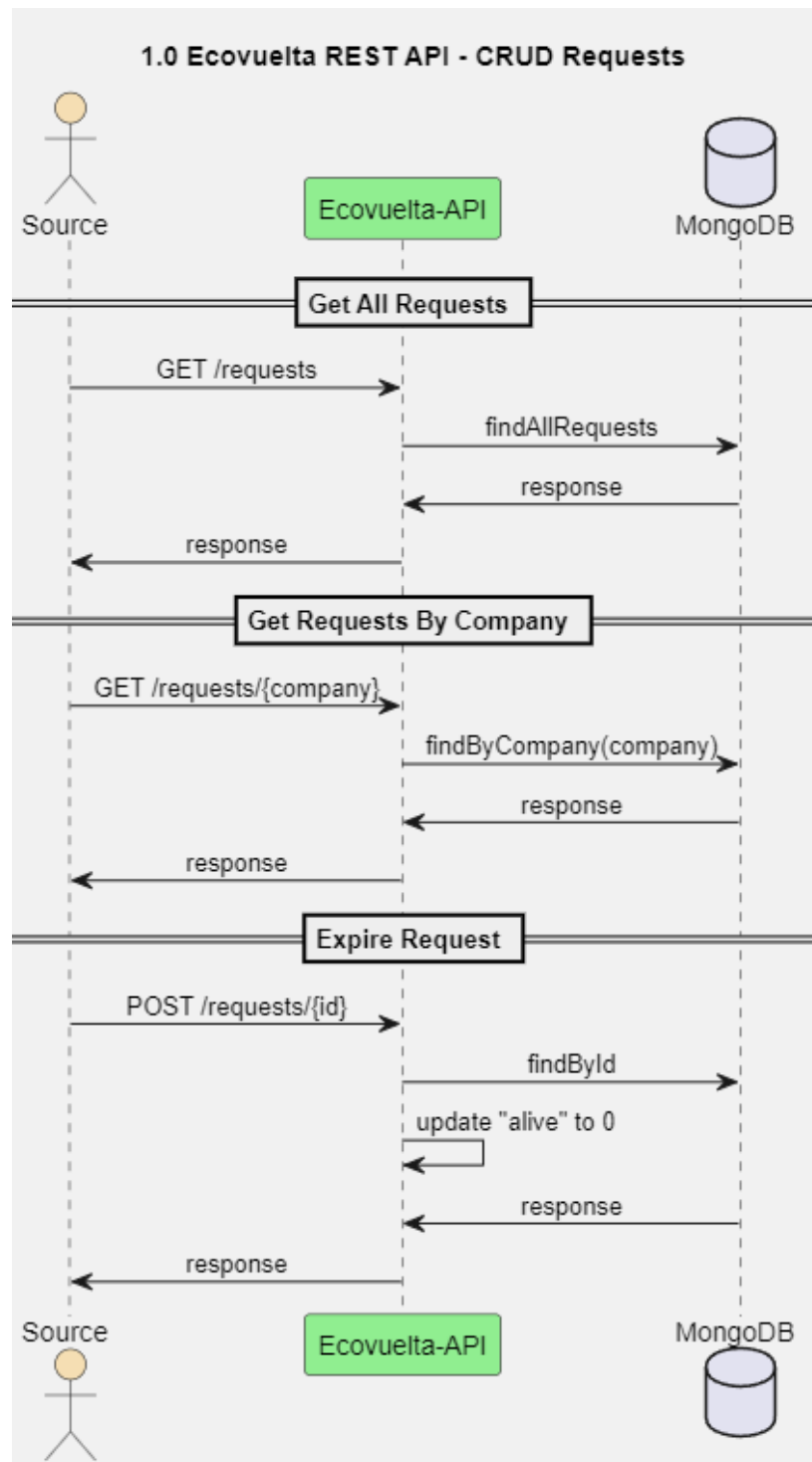
Anexo H. Diagrama de secuencia CRUD Ubicaciones
Fuente: Elaboración propia

10.9. Anexo I



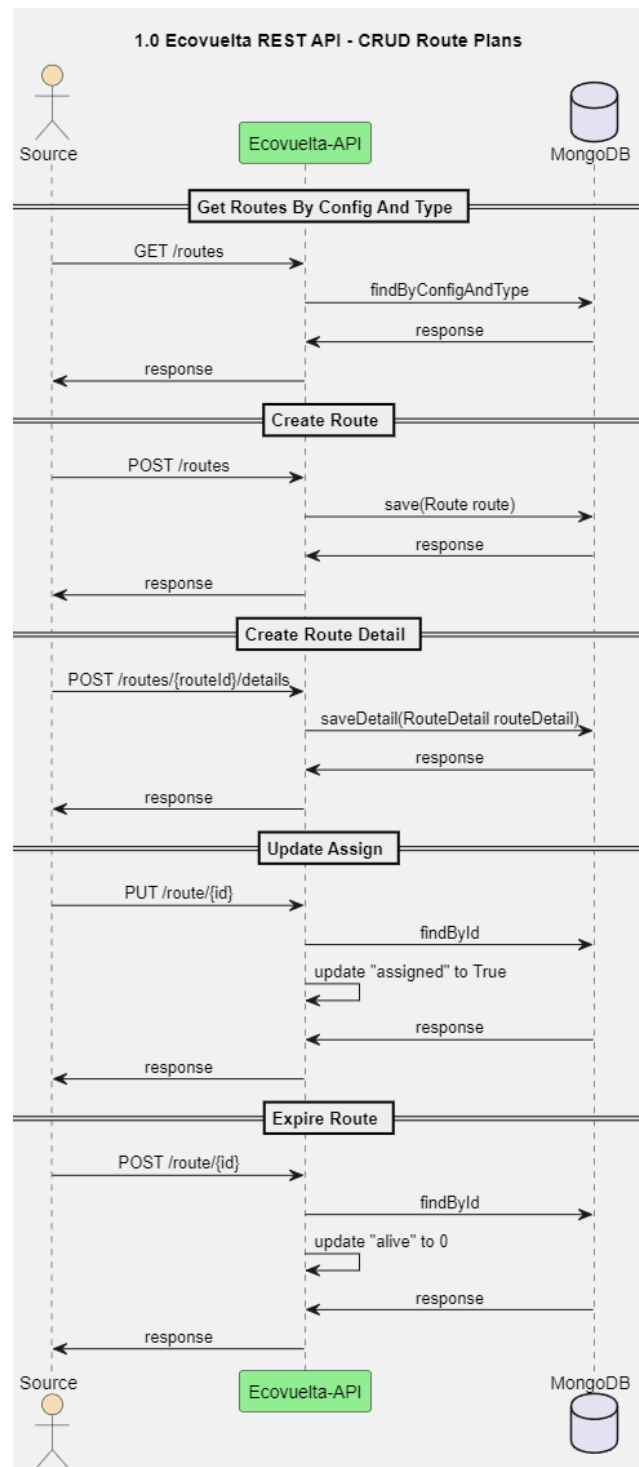
Anexo I. Diagrama de secuencia CRUD Ubicaciones
Fuente: Elaboración propia

10.10. Anexo J



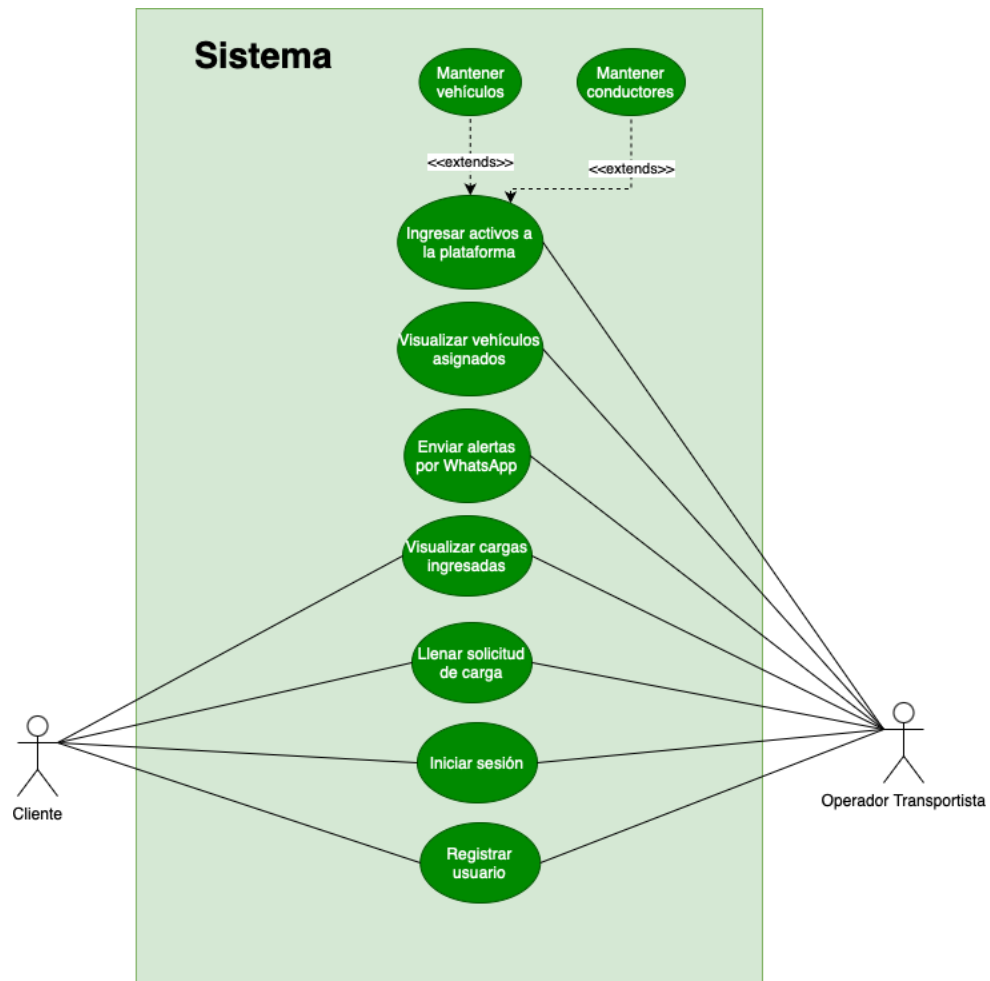
Anexo I. Diagrama de secuencia CRUD Solicitudes
Fuente: Elaboración propia

10.11. Anexo K



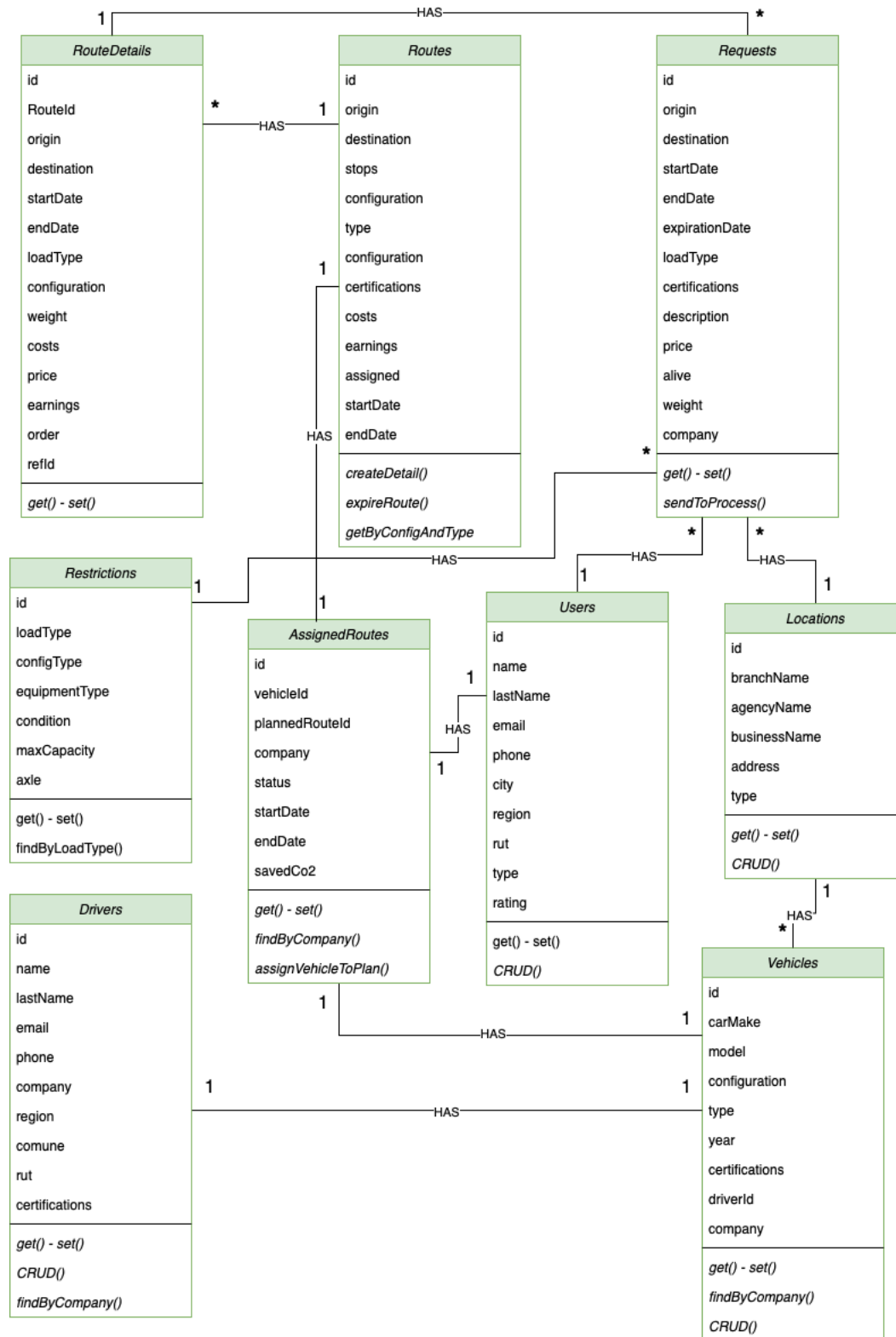
Anexo K. Diagrama de secuencia CRUD Rutas
Fuente: Elaboración propia

10.12. Anexo L



Anexo M. Casos de uso para Ecovuelta
Fuente: Elaboración propia

10.13. Anexo M



Anexo L. Diagrama de clases Ecovuelta
Fuente: Elaboración propia.