UNIVERSITAT OBERTA
DE
CATALUNYA

RideRush: Plataforma integral de alquiler y gestión de flotas turísticas

Trabajo de Fin de Grado presentado en cumplimiento
parcial de los requisitos para el grado de
INGENIERÍA INFORMÁTICA
por
Moisés Pernas Concepción
enero de 2024

El TFG de Moisés Pernas Concepción
es aprobado por:

_____
Prof. Antoni Oller Arcas

_____
Prof. Santiago Caballe Llobet

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RideRush: Plataforma integral de alquiler y gestión de flotas turísticas

Moisés Pernas Concepción

ABSTRACT

This dissertation focuses on the transition from a monolithic application to a microservices architecture, emphasizing a practical approach to modern software development. The main goal is to enhance the scalability and adaptability of a car rental service, addressing key challenges such as managing communication between microservices and ensuring data consistency.

The implementation starts from scratch, evaluating communication options and using retry mechanisms to obtain specific instances from Eureka, the discovery service. A configuration server is incorporated to consolidate application properties. The agile methodology based on Scrum guides the development process, detailing user stories in planned iterations.

The thesis also addresses critical aspects of monitoring and error management, implementing exceptions and leveraging the non-blocking capabilities of Mono within the Spring framework. Eureka is used to obtain health indicators of the microservices.

In fulfilment of the undergraduate studies in Computer Engineering, this dissertation provides a comprehensive view of the process of adopting microservices, highlighting practical and modern aspects of software development with Java EE.

Agradecimientos

En primer lugar, quiero expresar mi profundo agradecimiento a mi familia por las enseñanzas que me brindaron y los valores que me inculcaron. Estoy eternamente agradecido por todo lo que me han ofrecido a lo largo de los años y por haberme mostrado que con esfuerzo, sacrificio y dedicación se pueden alcanzar metas extraordinarias.

Adicionalmente, deseo agradecer a mi tutor, Antoni Oller Arcas, por su invaluable ayuda y comprensión durante la elaboración de este trabajo. Sus consejos, comentarios y orientación fueron esenciales para el desarrollo de esta TFG.

También, quiero expresar mi agradecimiento a mis amigos y seres queridos por su comprensión en este período y por estar siempre allí para apoyarme y alentarme a alcanzar mis metas. Su papel fue crucial para mantener el equilibrio entre mi dedicación a los estudios y mi vida personal.

Finalmente, debo reconocer y agradecer a mis profesores y compañeros de estudio en la Universitat Oberta Catalunya, así como a todos los profesionales que he conocido a lo largo de mi carrera. Su influencia y apoyo fueron fundamentales en mi desarrollo como profesional.

1 Introducción

En un contexto de constante avance tecnológico, RideRush no solo aspira a mantenerse a la vanguardia, sino a liderar la transformación. Este proyecto estratégico busca redefinir la experiencia de alquiler de vehículos, enfrentándose a desafíos únicos en el sector turístico. Aunque, la demanda intensiva, en temporada alta, satura el servicios, RideRush ve esta situación como una oportunidad para innovar y mejorar su servicio.

Reconocida por su proceso de reserva simplificado y atención al cliente excepcional, RideRush afronta el reto de mantener altos estándares durante períodos de alta demanda. El proyecto se propone implementar una solución tecnológica que optimice costos, mejore la eficiencia y consolide a RideRush como líder innovador en el mercado del alquiler de vehículos. Además, con la expansión internacional de RideRush, el proyecto respaldará este crecimiento, garantizando la calidad del servicio en diversas regiones.

Consciente de la complejidad del camino hacia la innovación, RideRush ha solicitado un exhaustivo proceso de análisis y diseño antes de adoptar una nueva arquitectura. Este enfoque asegura decisiones fundamentadas y alineadas con las necesidades y objetivos de la empresa. Así, RideRush no solo está preparada para superar los desafíos actuales, sino que también está dando forma activamente al futuro del alquiler de vehículos, destacando su compromiso con la satisfacción del cliente y la innovación constante.

1.1 Contexto

RideRush es una empresa que se especializa en el alquiler de coches turísticos, ofreciendo opciones desde económicas hasta deportivas. Su distinción radica en un enfoque centrado en el cliente, ofreciendo un proceso de reservas intuitivo y cómodo a

través de su sitio web, con soporte en distintas zonas geográficas. La necesidad que RideRush nos plantea se centra en una mejora específica: evitar la saturación del servicio en momentos de alta demanda, que genera costos significativos en el proceso de escalado vertical. Además, busca fidelizar a los clientes mediante la optimización de reservas, la gestión eficiente del inventario y la automatización de procesos administrativos.

Nuestra propuesta de proyecto no solo se limita a abordar estos desafíos particulares, sino que también busca transformar digitalmente el panorama del alquiler de coches turísticos, estableciendo un estándar de eficiencia y calidad en la industria.

1.2 Justificación

El sector de alquiler de vehículos está sujeto a fluctuaciones de la demanda en función de diversos factores, tales como la ubicación geográfica, el día de la semana o la estación del año. Estas variaciones exigen un sistema que sea capaz de adaptarse a ellas, escalando su capacidad para atender la demanda en los momentos de mayor actividad, y reduciéndola para optimizar el uso de los recursos en los períodos de menor afluencia.

Un servicio de alquiler de vehículos implica una serie de pasos complejos, como la verificación de la disponibilidad del vehículo, la reserva, la recogida y la devolución del mismo. Cada uno de estos pasos puede implicar la interacción con distintos sistemas y bases de datos. Esta interacción puede ocasionar operaciones lentas y erróneas si no se gestionan adecuadamente. Por ello, resulta imprescindible contar con un sistema que pueda realizar estas operaciones de forma eficaz y precisa.

En este contexto, las aplicaciones monolíticas pueden agravar estos problemas al impedir el escalado independiente de cada servicio. No obstante, los clientes demandan una experiencia de usuario ágil y sin interrupciones. Por consiguiente, el sistema debe

ofrecer respuestas rápidas y exactas a las peticiones de los clientes, con independencia de la carga del sistema o de la posible existencia de fallos en alguno de los servicios.

De lo expuesto se desprende que *RIDERUSH*, empresa ficticia objeto de estudio, presenta un problema de gestión de la demanda, la eficiencia operativa y la satisfacción del cliente, que requiere un sistema que sea capaz de adaptarse a las fluctuaciones del mercado, realizar operaciones complejas de forma eficaz y precisa, ofreciendo una experiencia de usuario ágil y sin interrupciones.

Sin embargo, la arquitectura monolítica que utiliza *RIDERUSH* no es capaz de cumplir con estos requisitos, ya que presenta limitaciones en cuanto a la escalabilidad, la resiliencia, la seguridad y la usabilidad. Por lo tanto, se hace necesario explorar una solución alternativa que supere estas limitaciones y que se adapte mejor a las características y necesidades del mercado.

1.3 Objetivos

El objetivo general de este trabajo es diseñar e implementar un sistema de alquiler de vehículos basado en microservicios para la empresa *RIDERUSH*. Con este sistema se pretende mejorar la gestión de la demanda, la eficiencia operativa y la satisfacción del cliente. Para ello, se plantean los siguientes objetivos específicos:

1. Analizar el estado del arte de las tecnologías Java EE y microservicios, así como sus ventajas e inconvenientes frente a la arquitectura monolítica.
2. Identificar los requisitos funcionales y no funcionales del sistema, así como los casos de uso y los actores involucrados.
3. Diseñar la arquitectura del sistema, definiendo los microservicios, las interfaces, las comunicaciones, los protocolos, los patrones y las herramientas a utilizar.
4. Implementar el sistema, aplicando las buenas prácticas de desarrollo, despliegue y monitorización de microservicios.

5. Evaluar el rendimiento, la escalabilidad, la resiliencia, la seguridad y la usabilidad del sistema.

Asimismo, este objetivo contribuirá al desarrollo de software moderno, basado en la modularidad, la flexibilidad y la interoperabilidad de los microservicios. Para lograr este objetivo, se aplicará y evaluará un enfoque práctico para el desarrollo de software basado en microservicios, que incluirá el análisis del estado del arte, la identificación de los requisitos, el diseño de la arquitectura, la implementación del sistema y la evaluación del rendimiento, la escalabilidad, la resiliencia, la seguridad y la usabilidad.

1.4 Estructura del documento

- Introducción: presenta el contexto, la justificación, los objetivos y la planificación temporal del proyecto.
- Estado del arte: contiene el análisis del marco tecnológico en el alquiler de vehículos turísticos y las arquitecturas basadas en microservicios.
- Análisis detallado de requerimientos: xxpone los requisitos de la aplicación, los casos de uso, las historias de usuario y los pilares ágiles del proyecto.
- Diseño del sistema: muestra el prototipado de las interfaces de usuario, el esquema relacional de la base de datos, el diagrama de clases del dominio y la arquitectura del sistema.
- Implementación: describe las elecciones tecnológicas y los *Sprints* realizados para desarrollar el producto mínimo viable.
- Resultados y evaluación: presenta los resultados obtenidos y evalúa el grado de cumplimiento de los objetivos del proyecto.
- Conclusiones: resumen las principales aportaciones y aprendizajes del proyecto.

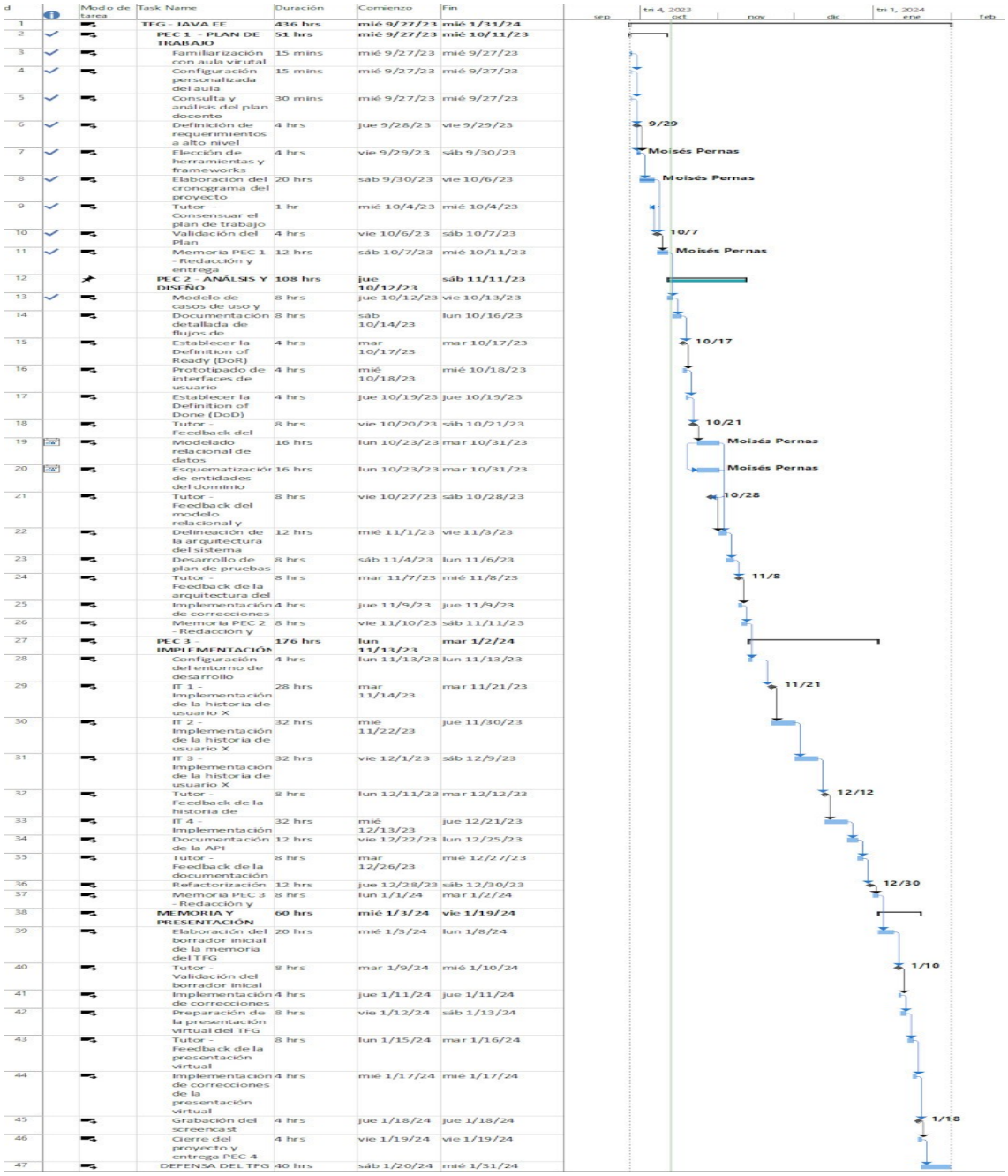
- Trabajo futuro: se proponen posibles mejoras y ampliaciones para el proyecto.

2 Planificación temporal

La organización eficaz del trabajo es un factor clave para el éxito de un proyecto. Por esta razón, hemos elaborado un plan de trabajo bien definido que nos facilitará la gestión de tareas, la asignación de prioridades y el cumplimiento de los objetivos del proyecto.

Asimismo, hemos adoptado la metodología ágil, ver [2], para las fases de desarrollo. Los sprints, ciclos breves de desarrollo, nos permiten entregar un incremento periódico del producto, lo que nos posibilita adaptarnos al cambio y a las necesidades variables del proyecto de manera flexible y eficiente.

En Figure 1, mostramos el cronograma general del proyecto.



d		Modo de tarea	Task Name	Duración	Comienzo	Fin
1			TFG - JAVA EE	436 hrs	mié 9/27/23	mié 1/31/24
2	✓		PEC 1 - PLAN DE TRABAJO	51 hrs	mié 9/27/23	mié 10/11/23
3	✓		Familiarización con aula virutal	15 mins	mié 9/27/23	mié 9/27/23
4	✓		Configuración personalizada del aula	15 mins	mié 9/27/23	mié 9/27/23
5	✓		Consulta y análisis del plan docente	30 mins	mié 9/27/23	mié 9/27/23
6	✓		Definición de requerimientos a alto nivel	4 hrs	jue 9/28/23	vie 9/29/23
7	✓		Elección de herramientas y frameworks	4 hrs	vie 9/29/23	sáb 9/30/23
8	✓		Elaboración del cronograma del proyecto	20 hrs	sáb 9/30/23	vie 10/6/23
9	✓		Tutor - Consensuar el plan de trabajo	1 hr	mié 10/4/23	mié 10/4/23
10	✓		Validación del Plan	4 hrs	vie 10/6/23	sáb 10/7/23
11	✓		Memoria PEC 1 - Redacción y entrega	12 hrs	sáb 10/7/23	mié 10/11/23
12			PEC 2 - ANÁLSIS Y DISEÑO	108 hrs	jue 10/12/23	sáb 11/11/23
13	✓		Modelo de casos de uso y	8 hrs	jue 10/12/23	vie 10/13/23
14			Documentación detallada de flujos de	8 hrs	sáb 10/14/23	lun 10/16/23
15			Establecer la Definition of Ready (DoR)	4 hrs	mar 10/17/23	mar 10/17/23
16			Prototipado de interfaces de usuario	4 hrs	mié 10/18/23	mié 10/18/23
17			Establecer la Definition of Done (DoD)	4 hrs	jue 10/19/23	jue 10/19/23
18			Tutor - Feedback del	8 hrs	vie 10/20/23	sáb 10/21/23
19			Modelado relacional de datos	16 hrs	lun 10/23/23	mar 10/31/23
20			Esquematización de entidades del dominio	16 hrs	lun 10/23/23	mar 10/31/23
21			Tutor - Feedback del modelo relacional y	8 hrs	vie 10/27/23	sáb 10/28/23
22			Delineación de la arquitectura del sistema	12 hrs	mié 11/1/23	vie 11/3/23
23			Desarrollo de plan de pruebas	8 hrs	sáb 11/4/23	lun 11/6/23
24			Tutor - Feedback de la arquitectura del	8 hrs	mar 11/7/23	mié 11/8/23
25			Implementación de correcciones	4 hrs	jue 11/9/23	jue 11/9/23
26			Memoria PEC 2 - Redacción y	8 hrs	vie 11/10/23	sáb 11/11/23
27			PEC 3 - IMPLEMENTACIÓN	176 hrs	lun 11/13/23	mar 1/2/24
28			Configuración del entorno de desarrollo	4 hrs	lun 11/13/23	lun 11/13/23
29			IT 1 - Implementación de la historia de usuario X	28 hrs	mar 11/14/23	mar 11/21/23
30			IT 2 - Implementación de la historia de usuario X	32 hrs	mié 11/22/23	jue 11/30/23
31			IT 3 - Implementación de la historia de usuario X	32 hrs	vie 12/1/23	sáb 12/9/23
32			Tutor - Feedback de la historia de	8 hrs	lun 12/11/23	mar 12/12/23
33			IT 4 - Implementación	32 hrs	mié 12/13/23	jue 12/21/23
34			Documentación de la API	12 hrs	vie 12/22/23	lun 12/25/23
35			Tutor - Feedback de la documentación	8 hrs	mar 12/26/23	mié 12/27/23
36			Refactorización	12 hrs	jue 12/28/23	sáb 12/30/23
37			Memoria PEC 3 - Redacción y	8 hrs	lun 1/1/24	mar 1/2/24
38			MEMORIA Y PRESENTACIÓN	60 hrs	mié 1/3/24	vie 1/19/24
39			Elaboración del borrador inicial de la memoria del TFG	20 hrs	mié 1/3/24	lun 1/8/24
40			Tutor - Validación del borrador inical	8 hrs	mar 1/9/24	mié 1/10/24
41			Implementación de correcciones	4 hrs	jue 1/11/24	jue 1/11/24
42			Preparación de la presentación virtual del TFG	8 hrs	vie 1/12/24	sáb 1/13/24
43			Tutor - Feedback de la presentación virtual	8 hrs	lun 1/15/24	mar 1/16/24
44			Implementación de correcciones de la presentación virtual	4 hrs	mié 1/17/24	mié 1/17/24
45			Grabación del screencast	4 hrs	jue 1/18/24	jue 1/18/24
46			Cierre del proyecto y entrega PEC 4	4 hrs	vie 1/19/24	vie 1/19/24
47			DEFENSA DEL TFG	40 hrs	sáb 1/20/24	mié 1/31/24

Figure 1: Cronograma del proyecto

Para facilitar la visualización, se desglosan cada una de las fases clave del proyecto, señalando los hitos, las actividades y los entregables que se realizarán en cada una de ellas.

1. En esta primera fase Figure 2, el objetivo principal es realizar un *kick-off* con el tutor de la asignatura, validar la idea a desarrollar e implementar en el TFG y elaborar un plan de trabajo general.

Id		Modo de tarea	Task Name	Duración	Comienzo	Fin
2	✓		**PEC 1 - PLAN DE TRABAJO**	**51 hrs**	**mié 9/27/23**	**mié 10/11/23**
3	✓		Familiarización con aula virutal	15 mins	mié 9/27/23	mié 9/27/23
4	✓		Configuración personalizada del aula	15 mins	mié 9/27/23	mié 9/27/23
5	✓		Consulta y análisis del plan docente	30 mins	mié 9/27/23	mié 9/27/23
6	✓		Definición de requerimientos a alto nivel	4 hrs	jue 9/28/23	vie 9/29/23
7	✓		Elección de herramientas y frameworks	4 hrs	vie 9/29/23	sáb 9/30/23
8	✓		Elaboración del cronograma del proyecto	20 hrs	sáb 9/30/23	vie 10/6/23
9	✓		Tutor - Consensuar el plan de trabajo	1 hr	mié 10/4/23	mié 10/4/23
10	✓		Validación del Plan	4 hrs	vie 10/6/23	sáb 10/7/23
11	✓		Memoria PEC 1 - Redacción y entrega	12 hrs	sáb 10/7/23	mié 10/11/23

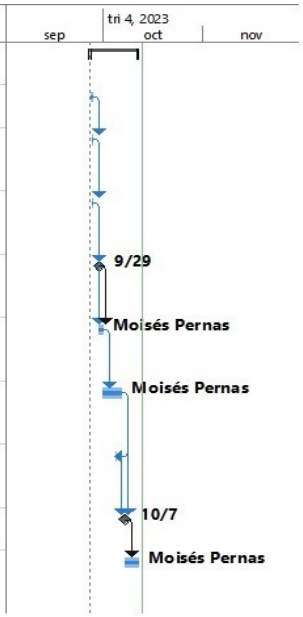


Figure 2: Plan de trabajo

2. En esta fase Figure 3, el objetivo principal es analizar y diseñar la solución que se quiere implementar para RideRush. Para ello, identificaremos y validaremos las historias de usuario, que describen las funcionalidades y requisitos que el cliente espera de la plataforma. Además, desarrollaremos el modelo de datos, que representa la estructura y las relaciones que se manejarán en la plataforma. Con estas actividades, definiremos todos los componentes que integrarán el sistema final, tanto a nivel lógico como físico.

Id	Modo de tarea	Task Name	Duración	Comienzo	Fin
12		**PEC 2 - ANÁLSIS Y DISEÑO**	**108 hrs**	**jue 10/12/23**	**sáb 11/11/23**
13		Modelo de casos de uso y	8 hrs	jue 10/12/23	vie 10/13/23
14		Documentación detallada de flujos de	8 hrs	sáb 10/14/23	lun 10/16/23
15		Establecer la Definition of Ready (DoR)	4 hrs	mar 10/17/23	mar 10/17/23
16		Prototipado de interfaces de usuario	4 hrs	mié 10/18/23	mié 10/18/23
17		Establecer la Definition of Done (DoD)	4 hrs	jue 10/19/23	jue 10/19/23
18		Tutor - Feedback del	8 hrs	vie 10/20/23	sáb 10/21/23
19		Modelado relacional de datos	16 hrs	lun 10/23/23	mar 10/31/23
20		Esquematización de entidades del dominio	16 hrs	lun 10/23/23	mar 10/31/23
21		Tutor - Feedback del modelo relacional y	8 hrs	vie 10/27/23	sáb 10/28/23
22		Delineación de la arquitectura del sistema	12 hrs	mié 11/1/23	vie 11/3/23
23		Desarrollo de plan de pruebas	8 hrs	sáb 11/4/23	lun 11/6/23
24		Tutor - Feedback de la arquitectura del	8 hrs	mar 11/7/23	mié 11/8/23
25		Implementación de correcciones	4 hrs	jue 11/9/23	jue 11/9/23
26		Memoria PEC 2 - Redacción y	8 hrs	vie 11/10/23	sáb 11/11/23

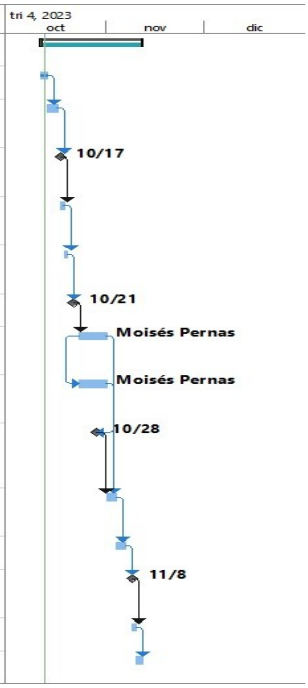


Figure 3: Análisis y diseño

3. En la fase siguiente Figure 4, implementaremos la solución para RideRush. Instalaremos el entorno de trabajo, crearemos y subiremos el proyecto a GitLab, y desarrollaremos las historias de usuario por iteraciones. Aplicaremos patrones de diseño y buenas prácticas de codificación para lograr un código mantenible y escalable.

Id	Modo de tarea	Task Name	Duración	Comienzo	Fin
27		**PEC 3 - IMPLEMENTACIÓN**	**176 hrs**	**lun 11/13/23**	**mar 1/2/24**
28		Configuración del entorno de desarrollo	4 hrs	lun 11/13/23	lun 11/13/23
29		IT 1 - Implementación de la historia de usuario X	28 hrs	mar 11/14/23	mar 11/21/23
30		IT 2 - Implementación de la historia de usuario X	32 hrs	mié 11/22/23	jue 11/30/23
31		IT 3 - Implementación de la historia de usuario X	32 hrs	vie 12/1/23	sáb 12/9/23
32		Tutor - Feedback de la historia de	8 hrs	lun 12/11/23	mar 12/12/23
33		IT 4 - Implementación	32 hrs	mié 12/13/23	jue 12/21/23
34		Documentación de la API	12 hrs	vie 12/22/23	lun 12/25/23
35		Tutor - Feedback de la documentación	8 hrs	mar 12/26/23	mié 12/27/23
36		Refactorización	12 hrs	jue 12/28/23	sáb 12/30/23
37		Memoria PEC 3 - Redacción y	8 hrs	lun 1/1/24	mar 1/2/24

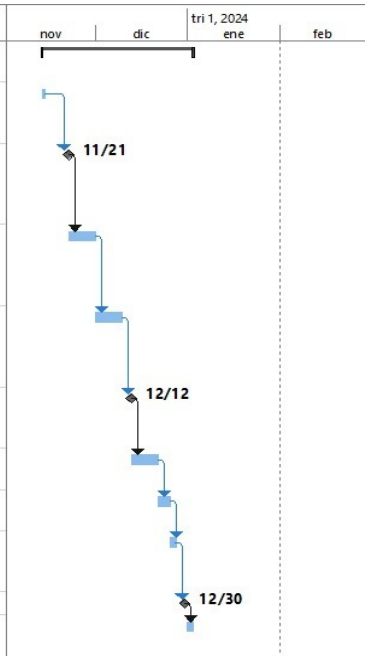
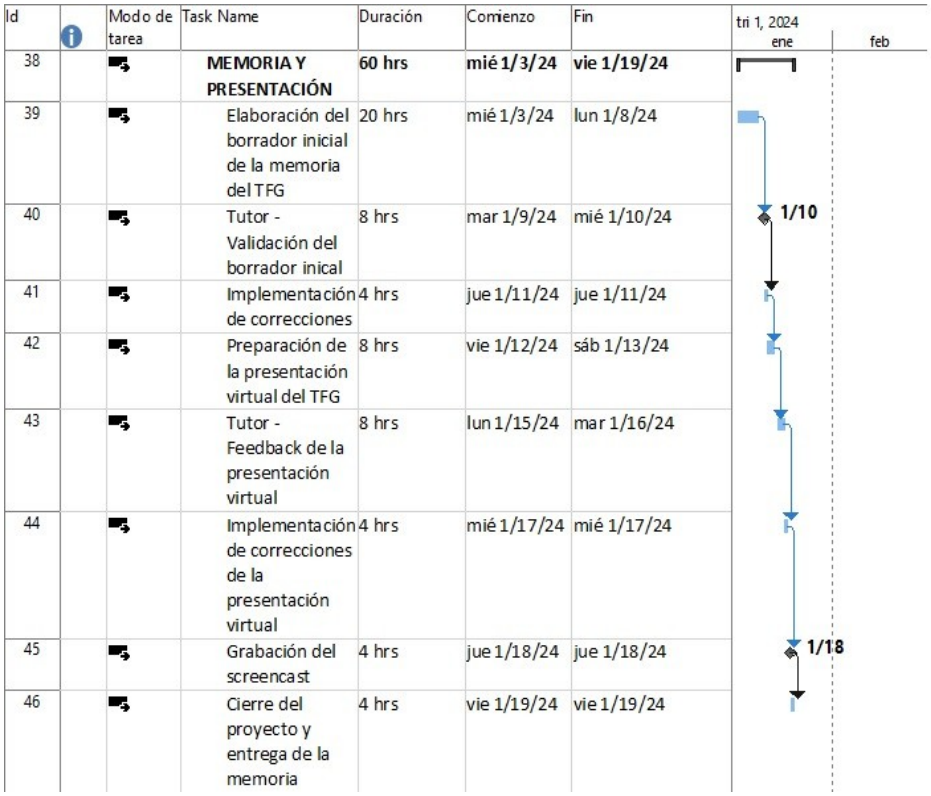


Figure 4: Implementación

4. Por último, Figure 5, finalizaremos la memoria y la presentación del proyecto. Revisaremos y completaremos la documentación de la memoria, que hemos ido elaborando a lo largo del semestre, siguiendo las normas y el formato establecidos. Prepararemos la presentación al tribunal que evaluará nuestro trabajo, utilizando un soporte visual adecuado y ensayando la exposición oral.

   Además, incluiremos un manual de instalación, si nos fuera requerido, que explique los pasos y los requisitos necesarios para instalar y ejecutar la plataforma. Con estas acciones, culminaremos el proyecto.



Id	Modo de tarea	Task Name	Duración	Comienzo	Fin
38		**MEMORIA Y PRESENTACIÓN**	**60 hrs**	**mié 1/3/24**	**vie 1/19/24**
39		Elaboración del borrador inicial de la memoria del TFG	20 hrs	mié 1/3/24	lun 1/8/24
40		Tutor - Validación del borrador inical	8 hrs	mar 1/9/24	mié 1/10/24
41		Implementación de correcciones	4 hrs	jue 1/11/24	jue 1/11/24
42		Preparación de la presentación virtual del TFG	8 hrs	vie 1/12/24	sáb 1/13/24
43		Tutor - Feedback de la presentación virtual	8 hrs	lun 1/15/24	mar 1/16/24
44		Implementación de correcciones de la presentación virtual	4 hrs	mié 1/17/24	mié 1/17/24
45		Grabación del screencast	4 hrs	jue 1/18/24	jue 1/18/24
46		Cierre del proyecto y entrega de la memoria	4 hrs	vie 1/19/24	vie 1/19/24

Figure 5: Memoria y presentación

3 Estado del arte

El presente capítulo se sumerge en el análisis detallado del estado actual de la investigación y desarrollo en el ámbito de la arquitectura de microservicios y su aplicación en el sector de alquiler de vehículos. Explorar el estado del arte es crucial para contextualizar este trabajo, identificar tendencias y comprender soluciones previas. Esta revisión busca establecer un fundamento sólido, resaltando contribuciones significativas y señalando las lagunas que esta investigación abordará.

En el ámbito específico del alquiler de vehículos, el panorama actual se caracteriza por una competencia dinámica entre diversas empresas que buscan adaptarse a las crecientes demandas de los consumidores y a las tendencias del mercado. En este

contexto, resulta esencial examinar las estrategias y enfoques adoptados por las principales compañías.

3.1 Análisis del marco tecnológico en el alquiler de vehículos

En el sector de alquiler de vehículos, empresas líderes como Enterprise y Hertz (ver Figure 6) experimentan un crecimiento sostenido, evidenciando no solo una expansión cuantitativa en flotas y ubicaciones, sino también la adopción de enfoques tecnológicos avanzados.

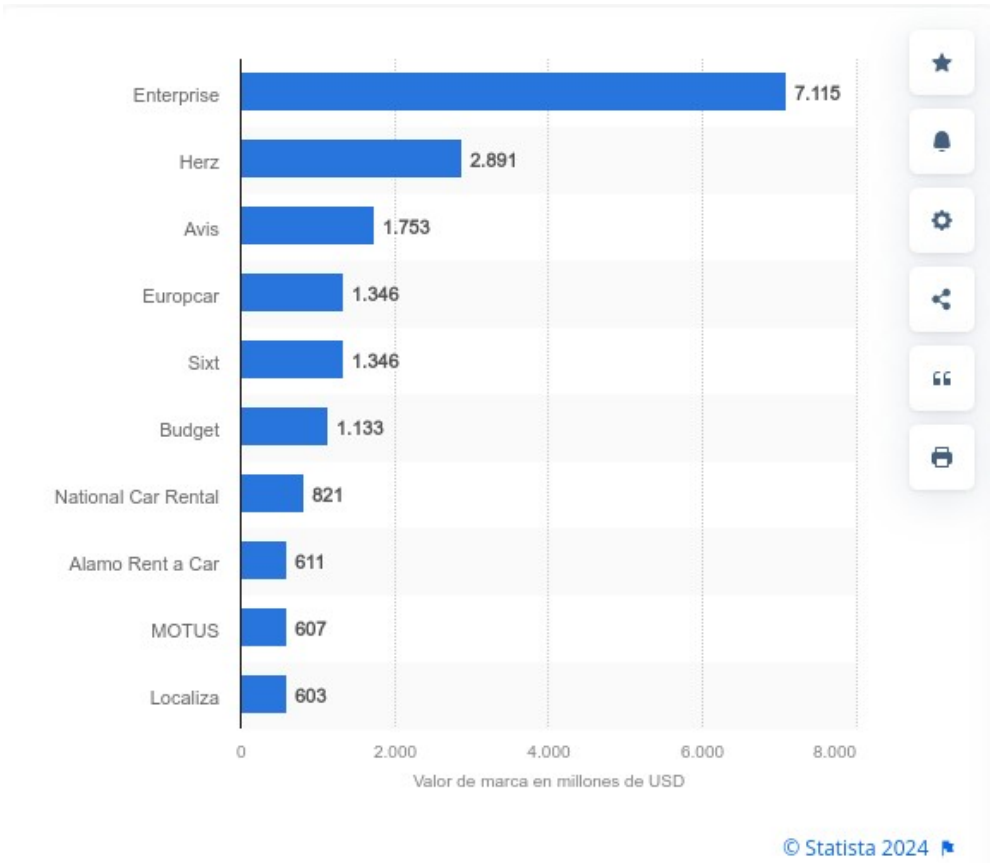


Figure 6: *Ranking*. Principales empresas de servicios de alquiler de vehículos [22]

Este dinamismo del mercado subraya la importancia crucial de la adopción de tecnologías innovadoras para mantener la competitividad. En este contexto, una

arquitectura basada en componentes no solo aporta flexibilidad y escalabilidad, sino que también se alinea estratégicamente con la creciente demanda del mercado. Este enfoque busca proporcionar servicios altamente personalizados y eficientes a lo largo de todo el proceso de alquiler, respondiendo a las expectativas del cliente en constante evolución.

3.2 Arquitecturas basadas en microservicios

En el panorama actual del desarrollo de software, las Arquitecturas Basadas en Microservicios (MSA) emergen como un paradigma de diseño eficiente para abordar los desafíos inherentes a la complejidad y evolución constante de las aplicaciones modernas. Este enfoque propone descomponer una aplicación monolítica en un conjunto de servicios pequeños, independientes y autónomos, cada uno encargado de una funcionalidad específica.

Las MSA se distinguen por su énfasis en la modularidad, donde cada microservicio opera de forma independiente, comunicándose con otros a través de mecanismos ligeros. Esta descomposición en unidades autónomas facilita el desarrollo, despliegue y escalado de servicios de manera individual, permitiendo una mayor flexibilidad en el ciclo de vida de la aplicación. La adopción de MSA promueve la agilidad en el desarrollo, al permitir que equipos trabajen en paralelo en diferentes microservicios sin interferencias. [1]

En el ámbito de las MSA, la orquestación y la coreografía representan dos enfoques cruciales para la coordinación de microservicios Figure 7. La orquestación implica la intervención de un componente central, un orquestador, que coordina y controla la ejecución de los microservicios para alcanzar una funcionalidad completa del sistema. Por otro lado, la coreografía distribuye la lógica de coordinación entre los propios microservicios, permitiendo una interacción más descentralizada y autónoma.

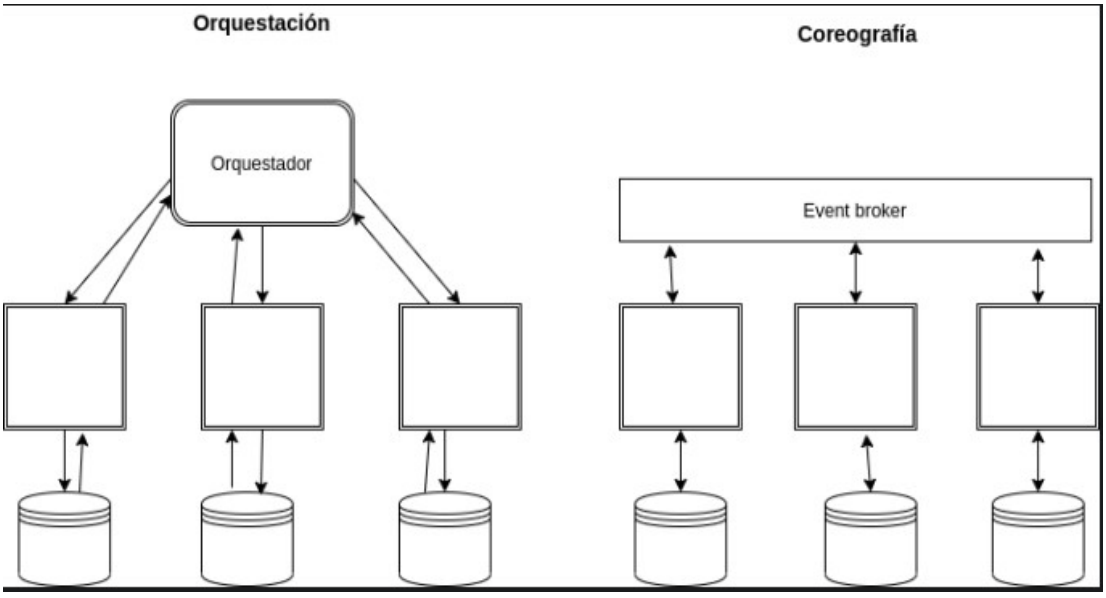


Figure 7: Estilos de coordinación entre microservicios. [23]

Las MSA, al incorporar la orquestación o la coreografía según el contexto, ofrece beneficios significativos en términos de escalabilidad y agilidad. Sin embargo, cada enfoque presenta desafíos únicos, desde la gestión de la comunicación entre microservicios hasta la garantía de la consistencia de datos en entornos distribuidos. En 6.3, veremos como se emplean ambas técnicas para la comunicación entre los microservicios.

3.3 Comparación de Arquitecturas: MSA frente a SOA y Monolíticas

En la elección de la arquitectura adecuada para un sistema distribuido, es esencial analizar las distintas opciones disponibles. En este apartado, exploraremos dos comparaciones fundamentales: Microservicios versus Arquitecturas Orientadas a Servicios (SOA) y Microservicios frente a Arquitecturas Monolíticas. Cada una de estas comparaciones destaca las diferencias fundamentales en principios de diseño, comunicación entre servicios, escalabilidad y agilidad en el desarrollo.

3.3.1 Microservicios vs. Arquitecturas Orientadas a Servicios (SOA)

Las Arquitecturas Basadas en Microservicios (MSA) y las Arquitecturas Orientadas a Servicios (SOA) comparten similitudes en su objetivo de descomponer sistemas complejos en componentes más manejables. Sin embargo, divergen en sus principios fundamentales y enfoques arquitectónicos. Este subapartado abordará las diferencias clave en términos de modularidad, comunicación entre servicios, escalabilidad y agilidad en el desarrollo.

Las MSA, como mencionado anteriormente, se centran en la modularidad y la autonomía de los servicios. Cada microservicio es independiente y puede ser desarrollado, implementado y escalado de manera autónoma. Por otro lado, SOA se basa en la idea de servicios más grandes y granulares, a menudo compartiendo recursos comunes. Los servicios en SOA pueden ser más pesados y requieren una coordinación centralizada.

En términos de despliegue, MSA favorece la implementación continua y la entrega ágil, mientras que SOA tiende a ser más rígida en sus ciclos de vida y despliegues menos frecuentes. Esta diferencia radica en la naturaleza modular y autónoma de los microservicios, lo que facilita actualizaciones y cambios sin afectar a todo el sistema.

La comunicación entre servicios es crucial en ambas arquitecturas. MSA favorece el uso de protocolos ligeros como HTTP/REST y GraphQL, lo que facilita la interacción entre microservicios. Por otro lado, SOA suele utilizar protocolos más pesados como SOAP, lo que puede aumentar la complejidad de la comunicación.

Finalmente, en términos de escalabilidad, MSA ofrece una escalabilidad más fina, ya que cada microservicio puede ser escalado de forma independiente según las necesidades. SOA, al tener servicios más grandes, puede enfrentar desafíos en la escalabilidad granular y eficiente.

3.3.2 Microservicios vs. Monolíticas

La comparación entre Microservicios y Arquitecturas Monolíticas resalta las ventajas significativas de la modularidad y autonomía de los primeros sobre la simplicidad y singularidad de los últimos. Este subapartado explorará cómo las Arquitecturas Basadas en Microservicios superan los desafíos de las arquitecturas monolíticas en términos de escalabilidad, mantenimiento y resiliencia

En una arquitectura monolítica, toda la aplicación está contenida en un único código base, lo que facilita el desarrollo, pero dificulta la implementación de actualizaciones sin afectar todo el sistema. Por otro lado, MSA permite el desarrollo y despliegue continuo, ya que cada microservicio es independiente y puede ser actualizado sin afectar otros componentes.

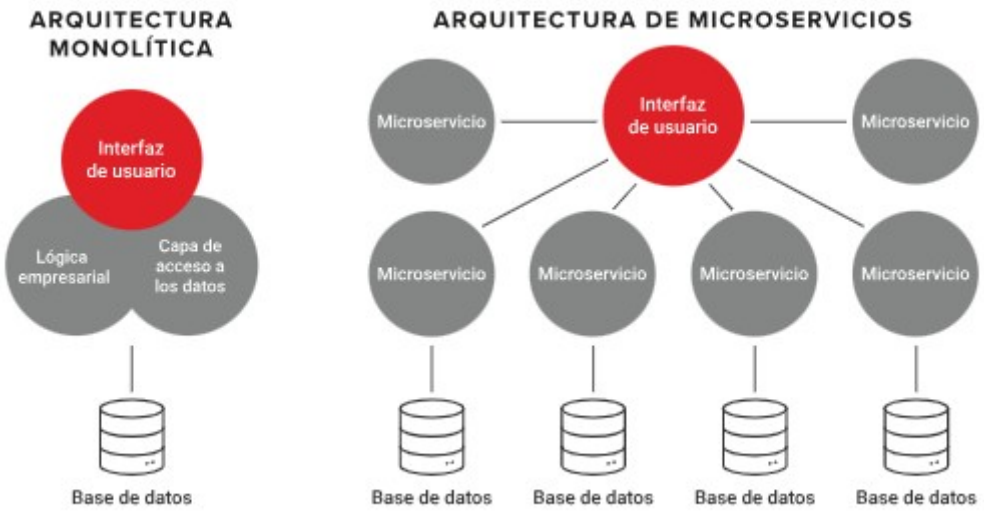


Figure 8: Comparativa entre las estructuras de las MA y MSA [21]

La resiliencia y tolerancia a fallos son aspectos críticos en sistemas distribuidos. MSA, al ser modular y autónoma, ofrece mayor resiliencia, ya que la falla de un microservicio no necesariamente afecta a toda la aplicación. En cambio, en una arquitectura monolítica, una falla puede tener consecuencias más amplias.

En términos de escalabilidad, MSA proporciona una mayor flexibilidad al permitir la escalabilidad individual de microservicios según la carga. En las arquitecturas monolíticas, la escalabilidad suele ser más limitada, ya que toda la aplicación debe escalarse en conjunto. Además, MSA facilita el mantenimiento, ya que cambios y actualizaciones pueden ser realizados de forma más granular.

Esta comparación destaca cómo las arquitecturas basadas en microservicios ofrecen ventajas significativas sobre las arquitecturas monolíticas y SOA en términos de modularidad, escalabilidad y agilidad en el desarrollo y despliegue. Estas diferencias fundamentales deben ser consideradas cuidadosamente al seleccionar la arquitectura más adecuada para nuestro proyecto.

4 Análisis detallado de requerimientos

En esta fase crucial del proyecto, nos adentramos en un análisis exhaustivo que sentará las bases para el desarrollo de la arquitectura de microservicios. Comenzaremos explorando las necesidades y requisitos del negocio, identificando de manera detallada tanto los aspectos funcionales como los no funcionales. Este desglose permitirá una comprensión completa de las expectativas y objetivos, sirviendo como el pilar sobre el cual se construirá la nueva arquitectura.

Dentro de este análisis, se llevará a cabo una evaluación minuciosa de los casos de uso y los actores involucrados, proporcionando una visión holística de las interacciones esperadas en el sistema. Además, nos sumergiremos en la planificación temporal, definiendo hitos y etapas del proyecto, estimando tiempos para cada fase y anticipándonos a posibles desafíos.

A medida que avanzamos en este proceso de análisis, no solo delineamos los requisitos funcionales y no funcionales, sino que también establecemos una sólida base temporal que guiará la ejecución efectiva del proyecto. Este enfoque integral garantiza una comprensión profunda antes de embarcarnos en el diseño y la implementación de la arquitectura de microservicios.

4.1 Requisitos de la aplicación

La obtención de requisitos es una parte fundamental del desarrollo de software, pues expresan las necesidades que debe cubrir el sistemas que desarrollamos y qué restricciones debe satisfacer. También, ayudan a establecer el alcance, los objetivos y las expectativas del proyecto, así como a facilitar la comunicación entre los desarrolladores y los usuarios.

Una especificación de requisitos efectiva es precisa, integral, coherente y verificable. Además, debe ser lo suficientemente flexible para adaptarse a posibles cambios durante el desarrollo. En esta sección, presentamos, de manera no exhaustiva y no formal, los requisitos de la aplicación, centrándonos en sus aspectos funcionales y no funcionales.

4.1.1 Requisitos funcionales: Definiendo la interacción del sistema

Los requisitos funcionales delinean el comportamiento del sistema en respuesta a estímulos externos. Tras la identificación de los *stakeholders* y la obtención de requisitos candidatos mediante técnicas como *brainstorming*, modelo de roles de usuario, entrevistas, cuestionarios y observación, se determinó la incorporación de diversas funciones al sistema. Estas funciones están diseñadas para mejorar la experiencia del usuario y garantizar una interacción fluida con la plataforma de alquiler de vehículos:

1. **Registro**: Los usuarios deben contar con la capacidad de crear una cuenta en el sistema.
2. **Autenticación**: Los usuarios pueden iniciar sesión de manera segura en el sistema.
3. **Recuperación de contraseña**: Funcionalidad que permite a los usuarios recuperar su contraseña en caso de olvido.
4. **Edición de perfil:** Se proporciona a los usuarios la opción de editar la información de su perfil según sea necesario.
5. **Búsqueda**: Los usuarios tienen la facultad de buscar vehículos según marca o categoría específica.
6. **Búsqueda por oficina:** Los usuario deben poder buscar los vehículos disponibles para una oficina en concreto.
7. **Notificación**: La plataforma garantiza que los administradores reciban los detalles sobre las nuevas reservas a través del correo electrónico.
8. **Alta y baja de vehículos**: Los administradores deben poder dar de alta y eliminar vehículos del inventario.
9. **Edición de reservas:** Los administradores deben poder editar la información de una reserva.
10. **Activación y desactivación de cuentas de usuario**: Los administradores deben poder activar o desactivar cuentas de usuario.

4.1.2 Requisitos no funcionales: Fundamentos para la eficiencia y calidad

En paralelo a los requisitos funcionales, los requisitos no funcionales son esenciales para moldear el sistema, estableciendo limitaciones y características que impactan directamente en su rendimiento general. A diferencia de los funcionales, estos requisitos se centran en propiedades que no están directamente vinculadas con las funciones específicas del sistema, sino más bien en su calidad y eficiencia.

En este proyecto, abordamos una variedad de requisitos no funcionales que complementan y enriquecen la experiencia del usuario y la efectividad global del sistema. Estos requisitos aseguran que, además de cumplir con las funciones específicas, el sistema exhiba niveles óptimos de seguridad, rendimiento y escalabilidad, contribuyendo así a su éxito y aceptación en el entorno operativo previsto.

- **Enfoque de desarrollo**: el proyecto se basará en la metodología ágil SCRUM, que permite iterar y adaptarse a los cambios de forma rápida y eficiente.
- **Arquitectura**: la arquitectura de la API seguirá los principios y estándares descritos en [2], que definen las mejores prácticas para diseñar e implementar APIs RESTful.
- **Formato de datos**: la API utilizará JSON como formato para intercambiar información entre el cliente y el servidor, debido a su simplicidad y compatibilidad con múltiples plataformas.
- **Rendimiento**: la API deberá ser capaz de procesar 1000 solicitudes por segundo, lo que implica optimizar el uso de recursos y aplicar técnicas de escalado horizontal y balanceo de carga.
- **Escalabilidad**: El sistema debe ser capaz de aumentar o disminuir el número de instancias de cada microservicio según la demanda. El sistema debe usar un balanceador de carga para distribuir las peticiones entre las instancias disponibles.
- **Base de datos por microservicio**: Cada microservicio deberá contar con su propia base de datos, garantizando así la independencia y autonomía de cada componente. Esta decisión se alinea con los principios de la arquitectura de microservicios, facilitando la escalabilidad y el mantenimiento individualizado de cada servicio.
- **Extensibilidad:** la API deberá ser capaz de añadir nuevas funciones sin afectar a las funciones existentes, siguiendo el principio de diseño abierto/cerrado. Se deberán utilizar mecanismos de versionado y documentación que permitan evolucionar la API sin romper la compatibilidad con los clientes.

4.2 Casos de uso del sistema

En el contexto del desarrollo de software, los casos de uso y los actores desempeñan un papel esencial al ofrecer una comprensión detallada de cómo los usuarios finales interactuarán con el sistema. Un caso de uso se centra en representar una funcionalidad específica del sistema que añade valor al usuario final, poniendo énfasis en la interacción entre el usuario y el sistema, sin adentrarse en los detalles internos de este último.

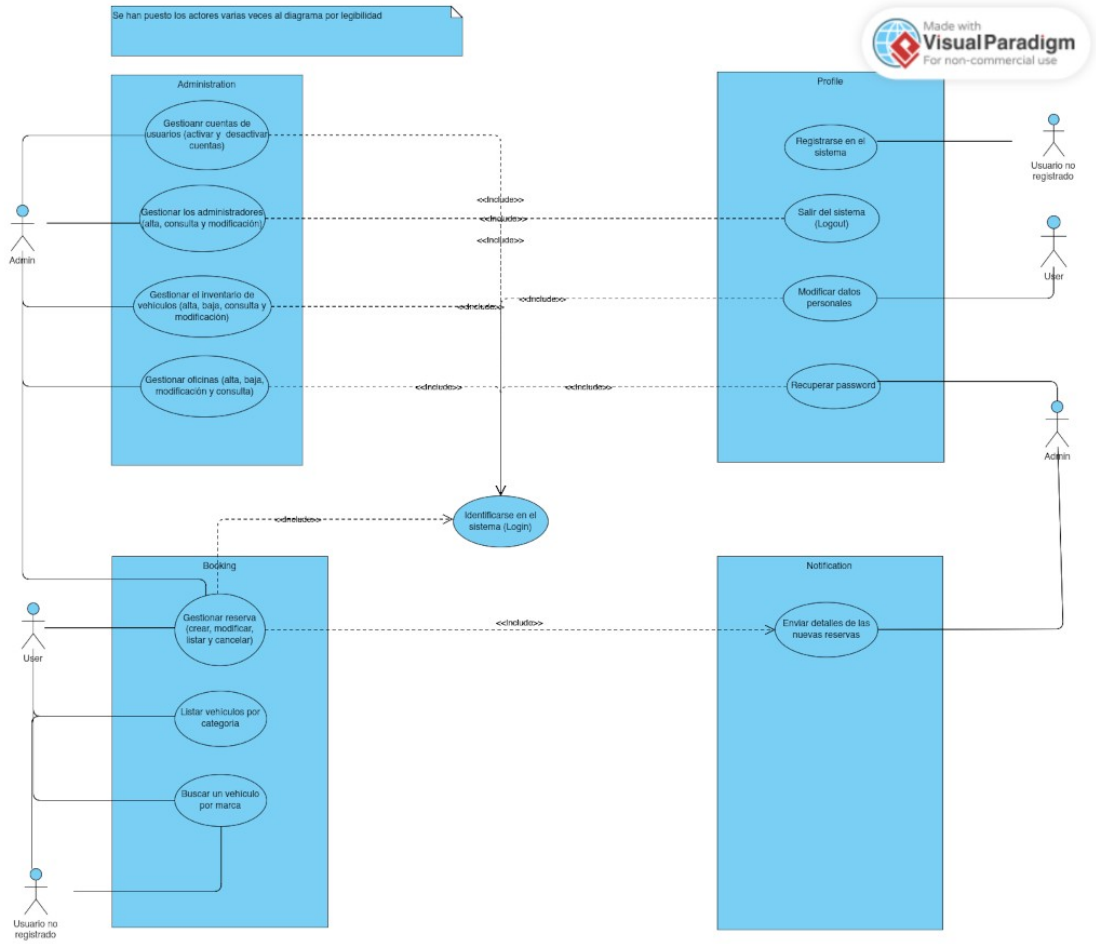


Figure 9: Modelo de casos de uso y actores del sistema

En paralelo, un actor es una entidad que interactúa con el sistema, abarcando desde usuarios humanos hasta otros sistemas o elementos del entorno. La definición precisa de casos de uso y actores permite diseñar un sistema que cumpla con las necesidades de los usuarios y brinde una experiencia efectiva y eficiente. En Figure 9, se presenta el modelo de casos de uso y actores correspondiente al sistema.

4.3 Historias de usuario

Por otro lado, las historias de usuario constituyen un componente fundamental en la metodología de desarrollo de software ágil, ofreciendo una visión detallada de las necesidades y expectativas del usuario final. En esta sección, se aborda minuciosamente cada historia de usuario, presentando su contexto a través de la descripción de precondiciones, postcondiciones, flujos principales y alternativos. Este enfoque meticuloso se alinea con prácticas formales de ingeniería de software, asegurando una comprensión exhaustiva de los requisitos funcionales del sistema.

Además, para optimizar la evaluación y priorización de estas historias, se emplea una escala de Fibonacci del 1 al 13 para estimar la complejidad y el esfuerzo de implementación de cada historia, se asigna un valor de 1 a 5 para indicar la probabilidad de dificultades durante la implementación (riesgo en desarrollo) y se utiliza una escala de 1 al 10 que refleja la importancia estratégica de la historia para el negocio (valor de negocios)

4.3.1 Optimizando la calidad de las historias de usuario

En primer lugar, cada historia de usuario se somete a un exhaustivo análisis mediante el empleo de la metodología INVEST [3]. Esta metodología, ampliamente reconocida en el campo de la ingeniería de software, se erige como un marco estructurado que garantiza la adecuada definición y efectividad de cada historia. En líneas generales, las evaluaciones realizadas siguiendo las directrices de INVEST se orientan a asegurar la coherencia, relevancia y valor de cada historia en el contexto del

desarrollo del sistema. A continuación, se detallan las directrices que han guiado este proceso de evaluación:

- **I (Independent):** evaluamos las dependencias entre las historias para facilitar las decisiones sobre el orden de implementación y la división o unificación de historias. Además, nos aseguramos de que las historias estén lo menos acopladas posible para mantener su independencia.
- **N (Negotiable)**: la necesidad del usuario debe estar claramente definida, pero la implementación debe ser flexible para permitir refinamientos futuros. Estos refinamientos pueden ser sugeridos por diversos grupos de interés.
- **V (Valuable)**: las historias deben proporcionar valor a los clientes y usuarios. Este valor ha sido confirmado por el Propietario del Producto para los diferentes grupos de interés.
- **E (Estimable)**: las historias de usuario deben estar construidas de tal forma que se pueda valorar el esfuerzo en su desarrollo. Esto implica que las historias de usuario deben ser lo suficientemente claras y precisas para que el desarrollador pueda estimar el tiempo y los recursos necesarios para completarlas.
- **S(Sized Appropriately)**: las historias de usuario deben tener un tamaño acorde al momento en que se deban implementar. Las más próximas a su desarrollo deben estar más detalladas y ser más pequeñas, lo que facilitará su estimación e implementación.Por otro lado, las historias que están más lejos en el horizonte pueden tener un nivel de detalle inferior, acorde con la incertidumbre inherente a estas.

4.3.2 Estableciendo prioridades: Asignando valor a las historias de usuario

En segundo lugar, si bien cada historia de usuario posee su propia relevancia, es imperativo concentrarse en aquellas que aporten el mayor valor al sistema. El propietario del producto asume la responsabilidad de asignar un valor a cada historia incluida en el sistema, considerando las siguientes variables:

- Beneficios derivados de la implementación de la funcionalidad.
- Pérdida o coste resultante de retrasar su implementación.
- Riesgos asociados a su implementación.
- Alineación con los objetivos del negocio.
- Valor diferencial en comparación con productos competidores.

Es fundamental tener presente que la definición de valor puede variar entre diferentes clientes. Por lo tanto, se emplea una escala cualitativa con significado intrínseco, brindando más información que una simple clasificación de prioridades como alta, media o baja.

Además, para facilitar al propietario del producto la tarea de priorizar la pila de producto de manera adecuada, se utiliza el cálculo del *Return of Investment* (ROI), aplicado de acuerdo a [3]. Este cálculo se define como:

$$CD = VN + CT + RR \; y \; VO$$

- **CD (Coste de demora)**
- **VN (Valor de negocio)**
- **CT (Criticidad en el tiempo)**: esta métrica se refiere a la necesidad de entregar el elemento en un marco de tiempo específico, asociado con la disminución gradual del valor. Cuanto mayor sea la necesidad de entrega, mayor será este factor
- **RR (Reducción del riesgo)** y **VO (valor de oportunidad)**: esta métrica asigna un valor relativo a la eliminación de uno o varios riesgos, o un valor por las oportunidades de negocio potenciales que el elemento pueda generar.

4.3.3 Análisis pormenorizado de historias de usuario

En tercer lugar, al aplicar estos criterios, se plasman las especificaciones detalladas de las historias de usuario del sistema.

Table 1: Registro de usuario

Historia de Usuario	
ID: 1	**Rol:** Interesado
Nombre historia: Registro de usuario mediante formulario	
Prioridad en negocio: 7	**Riesgo en desarrollo: 3**
Puntos estimados: 5	**Iteración asignada: 1**
Descripción: Como interesado **quiero** registrarme en el sistema **para poder** acceder a funcionalidades avanzadas.	
Criterios de aceptación: **Escenario 1: usuario no registrado** **Dado que** el usuario no está registrado **Y que** el usuario cumplimenta los campos requeridos en el registro **Cuando** el usuario hace clic en el botón de registro **Entonces** los usuarios serán los mismos más uno nuevo con los datos indicados **Y** el usuario recibirá un correo electrónico de confirmación con un enlace para activar su cuenta **Y** el usuario podrá acceder a las funcionalidades avanzadas **Escenario 2: usuario registrado** **Dado que** el usuario ya está registrado **Y que** el usuario cumplimenta la información requerida en el registro **Cuando** el usuario hace clic en el botón de registro **Entonces** el sistema muestra un mensaje informando de que la dirección de correo está asociada a un usuario existente. **Y** se proporciona un enlace "*Back to Login*" para que inicie sesión **Escenario 3: Registro fallido debido a campos inválidos** **Dado que** el usuario está en la página de registro **Y** que no cumplimenta todos los campos requeridos **Cuando** el usuario interesado hace clic en el botón de registro **Entonces** el sistema muestra un mensaje de error indicando los campos inválidos **Y** el usuario interesado pude corregir los campos indicados	
Criterio de calidad: • **I (*Independent*)**: el requisito en sí mismo es independiente. • **N (*Negotiable*)**: el registro en el sistema se puede hacer de varias formas. Por ejemplo, se puede realizar el alta con la cuenta de otros servicios, por lo que la forma de implementarla está abierta. • **T (*Testable*):** se puede verificar que el usuario se ha registrado correctamente. Por otro lado, si el usuario ya dispone de una cuenta creada se mostrará un mensaje informativo.	

Table 2: Autenticarse en el sistema

Historia de Usuario	
ID: 2	**Rol:** Usuario
Nombre historia: Autenticarse en el sistema	
Prioridad en negocio: 8	**Riesgo en desarrollo: 3**
Puntos estimados: 3	**Iteración asignada: 1**
Descripción: Como usuario registrado **quiero** autenticarme en el sistema **para poder** acceder a las funcionalidades del sistema	
Criterios de aceptación: **Escenario 1**: **usuario registrado en el sistema** **Y** podrá visualizar su nombre y apellido en la parte superior derecha **Dado que** el usuario ya está registrado en el sistema **Y que** el usuario está en la página de inicio de sesión **Y que** el usuario ingresa la dirección de correo electrónico correcta **Y que** el usuario ingresa la contraseña correcta **Cuando** el usuario hace clic en el botón Iniciar sesión **Entonces** el usuario debe estar autenticado **Y** el usuario será redirigido a la página principal (*Landing page*) **Y** el usuario podrá visualizar su nombre y apellido en la parte superior derecha **Escenario 2: campos vacíos** **Dado que** algún campo de los requeridos (correo y/o contraseña) está vacío **Y que** el usuario está en la página de inicio de sesión **Cuando** el usuario hace clic en el botón Iniciar sesión **Entonces** el sistema muestra un error **Y** el usuario es informado de los campos obligatorios. **Escenario 3: correo electrónico o contraseña incorrecta** **Dado que** el correo electrónico o contraseña son incorrectos **Cuando** el usuario hace clic en el botón Iniciar sesión **Entonces** el sistema muestra un error **Y** el usuario es informado de que el correo electrónico o la contraseña son incorrectos.	
Criterio de calidad: • **I (*Independent*)**: depende de que el usuario esté registrado, por lo tanto, tiene dependencia directa con ID: 1 • **N (*Negotiable*):** los detalles específicos de cómo se implementará la autenticación pueden ser negociables. Por ejemplo, podría ser a través de un nombre de usuario y contraseña, autenticación de dos factores, etc. • **T (*Testable*)**: es posible verificar que la autenticación funciona correctamente a través de pruebas. Por ejemplo, asegurándose de que un usuario pueda iniciar sesión con credenciales válidas y no pueda hacerlo con credenciales inválidas	

Table 3: Recuperación de contraseña

Historia de Usuario	
ID: 3	**Rol:** Usuario
Nombre historia: Recuperación de contraseña	
Prioridad en negocio: 7	**Riesgo en desarrollo: 2**
Puntos estimados: 5	**Iteración asignada: 1**
Descripción: Como usuario **quiero** recuperar mi contraseña **para poder** acceder a mi cuenta en caso de olvidar mi contraseña	
Criterios de aceptación: **Escenario 1: el usuario recupera su contraseña** **Dado que** el usuario ha olvidado su contraseña **Y** el usuario selecciona "*Forgot Password*" en la página de inicio de sesión **Y** el usuario introduce su dirección de correo electrónico **Y** la dirección de correo está asociada a una cuenta **Cuando** el usuario hace clic en "*Submit*" **Entonces** el sistema informa que se ha enviado un correo a la cuenta asociada **Y** el sistema envía un correo electrónico al usuario con un enlace temporal **Y** el usuario hace clic en el enlace **Y** el usuario puede cambiar su contraseña **Escenario 2: cuenta de correo electrónico no asociada** **Dado que** el usuario ha olvidado su contraseña **Y** el usuario selecciona "*Forgot Password*" en la página de inicio de sesión **Y** el usuario introduce su dirección de correo electrónico **Y** la dirección de correo no está asociada a una cuenta **Cuando** el usuario hace clic en "*Submit*" **Entonces** el sistema informa que se ha enviado un correo a la cuenta asociada **Y** el sistema **no** envía un correo electrónico al usuario **Y** el usuario no puede cambiar su contraseña	
Criterio de calidad: • **I (*Independent*)**: depende de que el usuario esté registrado, por lo tanto, tiene dependencia directa con ID: 1 • **N (*Negotiable*):**: los detalles específicos de cómo se implementará la recuperación de contraseña pueden ser negociables. Por ejemplo, podría ser a través de un correo electrónico de recuperación, preguntas de seguridad, etc. • **T (*Testable*)**: es posible verificar que la recuperación de contraseña funciona correctamente a través de pruebas. Por ejemplo, asegurándose de que un usuario pueda recuperar su contraseña con éxito y cambiarla.	

Table 4: Editar de la información del perfil

Historia de Usuario		
ID: 4	**Rol:** Usuario	
Nombre historia: Editar de la información del perfil		
Prioridad en negocio: 3		**Riesgo en desarrollo: 2**
Puntos estimados: 3		**Iteración asignada: 1**
Descripción: Como un usuario con una cuenta activa **quiero** editar la información de mi perfil **para poder** mantener mi información actualizada		
Criterios de aceptación: **Escenario 1: el usuario edita su información de perfil exitosamente** **Dado que** el usuario ha iniciado sesión en su cuenta **Y** el usuario navega a la vista de su perfil **Y** el usuario edita su nombre, *email,* etc. **Cuando** el usuario hace *clic* "*Save*" **Entonces** la información del perfil del usuario se actualiza **Y** el sistema muestra un mensaje de confirmación al usuario **Escenario 2: campos vacíos** **Dado que** el usuario ha iniciado sesión en su cuenta **Y** el usuario navega a la vista de su perfil **Y** el usuario no cumplimenta algún campo obligatorio **Entonces** el sistema desactiva el botón de “Save” **Y** el usuario es informado de los campos obligatorios. **Y** la información del perfil del usuario permanece inalterable		
Criterio de calidad: • **I (*Independent*)**: depende de que el usuario esté registrado y autenticado, por lo tanto, tiene dependencia directa con ID: 1 y 2. • **N (*Negotiable):***: los detalles específicos de cómo se implementará la edición de la información del perfil pueden ser negociables. Por ejemplo, qué campos se pueden editar, si se requiere una verificación de contraseña, etc. • **T (*Testable*)**: es posible verificar que la edición de la información del perfil funciona correctamente a través de pruebas. Por ejemplo, asegurándose de que un usuario pueda cambiar su información y que los cambios se reflejen correctamente en su perfil.		

Table 5: Buscar vehículos disponibles por oficina

Historia de Usuario	
ID: 5	**Rol:** Interesado
Nombre historia: Buscar vehículo por oficina	
Prioridad en negocio: 7	**Riesgo en desarrollo: 2**
Puntos estimados: 3	**Iteración asignada: 1**
Descripción: Como interesado **quiero** buscar vehículos por oficina y fechas específicas **para** limitar la búsqueda de coches a los disponibles en esta oficina en la fecha señalada	
Criterios de aceptación: **Escenario 1: búsqueda exitosa** **Dado que** el interesado está en la página de búsqueda de vehículos **Y que** el interesado selecciona una oficina específica **Y que** el interesado elige fechas correctas para la búsqueda **Cuando** el interesado hace clic en el botón de búsqueda **Entonces** el sistema muestra la lista de vehículos disponibles en la oficina seleccionada para las fechas indicadas **Escenario 2: búsqueda con oficina no disponible** **Dado que** el interesado está en la página de búsqueda de vehículos **Y que** el interesado selecciona una oficina específica no disponible **Y que** el interesado elige fechas específicas para la búsqueda **Cuando** el interesado hace clic en el botón de búsqueda **Entonces** el sistema muestra una lista vacía **Escenario 3: validaciones al realizar una búsqueda** **Dado que** el usuario está en la página de búsqueda de vehículos **Cuando** el usuario selecciona una oficina en concreto **Y** el usuario elige fechas de inicio y finalización para la búsqueda **Entonces** el sistema realiza las siguientes validaciones: - La fecha de inicio debe ser igual o posterior a la fecha actual - La fecha de finalización no puede ser posterior a la fecha de inicio - Los vehículos deben estar disponible entre la fecha de inicio y finalización **Si** alguna de estas validaciones falla, el sistema muestra un mensaje de error correspondiente **De lo contrario**, el sistema muestra la lista de vehículos disponibles en la oficina	
Criterio de calidad: • **I (*Independent*)**: baja dependencia con dar alta un vehículo ID: 10 • **N (*Negotiable*)**: la forma de implementarse no está cerrada (se pueden realizar búsquedas exactas, aproximadas, etc.). • **T (*Testable*)**: se puede verificar que el resultado de la búsqueda coincide con los vehículos disponibles en nuestra base de datos.	

Table 6: Buscar vehículo por marca

Historia de Usuario	
ID: 6	**Rol:** Interesado
Nombre historia: Buscar vehículo por marca	
Prioridad en negocio: 7	**Riesgo en desarrollo: 2**
Puntos estimados: 3	**Iteración asignada: 2**
Descripción: Como interesado, quiero realizar búsquedas de vehículos según la marca para limitar los resultados a los vehículos disponibles de una marca específica.	
Criterios de aceptación: **Escenario 1: búsqueda exitosa** **Dado que** el interesado está en la página de búsqueda de vehículos **Y que** el interesado selecciona una marca específica **Cuando** el interesado hace clic en el botón de búsqueda **Entonces** el sistema muestra una lista de vehículos disponibles de la marca seleccionada **Escenario 2: búsqueda sin resultados** **Dado que** el interesado está en la página de búsqueda de vehículos **Y que** el interesado elige una marca para la cual no hay vehículos disponibles **Cuando** el interesado hace clic en el botón de búsqueda **Entonces** el sistema muestra una lista vacía de vehículos **Escenario 3: búsqueda con marca no válida** **Dado que** el interesado está en la página de búsqueda de vehículos **Y que** el interesado ingresa una marca que no es válida o no existe **Cuando** el interesado hace clic en el botón de búsqueda **Entonces** el sistema muestra una lista vacía de vehículos	
Criterio de calidad: • **I (*Independent*)**: baja dependencia con dar alta un vehículo ID: 10 • **N (*Negotiable)*:** la forma de implementarse no está cerrada (se pueden realizar búsquedas exactas, aproximadas, etc.). • **T (*Testable*)**: se puede verificar que el resultado de la búsqueda coincide con la marca en nuestra base de datos.	

Table 7: Consultar los detalles de un vehículo por su ID

Historia de Usuario	
ID: 7	**Rol:** Interesado
Nombre historia: Consultar detalles de un vehículo por ID	
Prioridad en negocio: 7	**Riesgo en desarrollo: 2**
Puntos estimados: 3	**Iteración asignada: 2**
Descripción: Como interesado, **quiero** consultar los detalles de un vehículos por su identificador (ID) **para poder v**alorar si me interesa alquilarlo	
Criterios de aceptación: **Escenario 1: consulta exitosa por ID existente** **Dado que** el interesado está en la página de consulta de vehículos **Cuando** el interesado hace clic en el botón de consultar **Entonces** el sistema muestra la información detallada del vehículo con el ID especificado **Y** el sistema presenta todos los detalles relevantes del vehículo correspondiente al ID proporcionado **Escenario 2: consulta sin resultados por ID inexistente** **Dado que** el interesado está en la página de consulta de vehículos **Y que** el interesado ingresa un ID que no existe en el campo de consulta **Cuando** el interesado hace clic en el botón de consultar **Entonces** el sistema muestra un mensaje indicando que no se encontraron resultados para el ID ingresado **Y** el sistema no presenta información de ningún vehículo	
Criterio de calidad: • **I (*Independent*)**: baja dependencia con dar alta un vehículo ID: 10 • **N (*Negotiable*):** la forma de implementarse no está cerrada • **T (*Testable*)**: se puede verificar que el resultado de la búsqueda coincide con la marca en nuestra base de datos.	

Table 8: Realizar una reserva

Historia de Usuario	
ID: 8	**Rol:** Usuario
Nombre historia: Realizar una reserva	
Prioridad en negocio: 9	**Riesgo en desarrollo: 5**
Puntos estimados: 8	**Iteración asignada: 2**
Descripción: como usuario registrado **quiero** realizar una reserva **para poder** asegurar el uso de un vehículo en una fecha específicas	
Criterios de aceptación: **Escenario 1: usuario realiza una reserva exitosamente** **Dado que** el usuario está registrado **Y que** el usuario está autenticado en el sistema **Y que** el usuario selecciona un vehículo disponible **Y que** el usuario introduce una fecha y hora válidas para la reserva **Cuando** el usuario confirma la reserva **Entonces** las reservas serán las misma más una nueva reserva con los datos indicados **Y** el sistema muestra un mensaje informando que la reserva se realzo con éxito **Y** el sistema envía una notificación a la bandeja de entada del correo del administrador con los detalles de la nueva reserva **Escenario 2: usuario registrado intenta realizar una reserva sin iniciar sesión** **Dado que** el usuario está registrado **Y el** usuario no está autenticado en el sistema **Y el** usuario selecciona un vehículo disponible **Y el** usuario introduce una fecha y hora válidas para la reserva **Cuando** el usuario confirma la reserva **Entonces** el sistema informa al usuario que debe estar autenticado **Y** el sistema redirige al usuario a la página de inicio de sesión	
Criterio de calidad: • **I (*Independent*)**: el usuario debe estar registrado, autenticado y haber vehículos dato de altas por lo que depende del requisito ID: 1, 2 y 10. **N (*Negotiable):***: los detalles específicos de cómo se realizará la reserva pueden ser negociables. Por ejemplo, qué información se requiere para hacer la reserva, etc. • **T (*Testable*)**: es posible verificar que la realización de una reserva funciona correctamente a través de pruebas. Por ejemplo, asegurándose de que un usuario pueda hacer una reserva y que la reserva se refleje correctamente en el sistema.	

Table 9: Cancelar reserva

Historia de Usuario	
ID: 9	**Rol:** Usuario
Nombre historia: Cancelar reserva	
Prioridad en negocio: 8	**Riesgo en desarrollo: 3**
Puntos estimados: 3	**Iteración asignada: 2**
Descripción: Como un usuario con reserva activa **quiero** cancelar mi reserva **para poder** gestionar mis reservas de manera eficiente	
Criterios de aceptación: **Escenario 1: el usuario cancela una reserva exitosamente** **Dado que** el usuario se ha identificado correctamente **Y** el usuario está en la vista de gestión de reservas **Y** el usuario selecciona una reserva activa **Cuando** el usuario hace clic en "Canclear" **Entonces** el número de reservas activas serán las mismas menos una la indicada **Y** el sistema muestra un mensaje informando el usuario del éxito de la operación	
Criterio de calidad: • **I (*Independent*)**: El usuario debe estar autenticado y tener una reserva activa por lo que depende del requisito ID: 2 y 8. • **N (*Negotiable*)**: Los detalles específicos de cómo se implementará la cancelación de la reserva pueden ser negociables. Por ejemplo, podría haber un límite de tiempo para cancelar, o podría haber una penalización por cancelación tardía • **T (*Testable*)**: Es posible verificar que la cancelación de la reserva funciona correctamente a través de pruebas. Por ejemplo, asegurándose de que un usuario pueda cancelar su reserva y que la cancelación se refleje correctamente en el sistema.	

Table 10: Dar de alta un vehículo

Historia de Usuario		
ID: 10	**Rol:** Administrador	
Nombre historia: Dar de alta un vehículo		
Prioridad en negocio: 7		**Riesgo en desarrollo: 3**
Puntos estimados: 5		**Iteración asignada: 1**
Descripción: como administrador **quiero** registrar un nuevo vehículo **para** que esté disponible en la flota y pueda ser alquilado.		
Criterios de aceptación: **Escenario 1: identificador del vehículo no existe** **Dado que** el administrador tiene el rol adecuado **Y que** el administrador cumplimenta los campos requeridos **Cuando** el administrador hace clic en el botón "*Save*" **Entonces** el número de vehículo en el inventario serán los mismos más uno con los datos indicados **Y** el administrador es informado del éxito de la operación **Y** el vehículo estará disponible para ser alquilado **Escenario 2: matricula del vehículo existe** **Dado que** la matricula del vehículo existe **Y que** el administrador tiene el rol adecuado **Y que** el administrador cumplimenta los campos requeridos **Cuando** el administrador hace clic en el botón de crear **Entonces** el sistema muestra un mensaje de error informando que el vehículo ya existe **Y** el administrador es direccionado a la vista de crear de vehículos **Y** el administrador puede editar los datos		
Criterio de calidad: • **I (*Independent*)**: el administrador debe estar autenticado en el sistema por lo que depende del requisito ID: 2 • **N (*Negotiable)***: los detalles específicos de cómo se dará de alta un vehículo pueden ser negociables. Por ejemplo, qué información se requiere para el registro del vehículo, si se necesita una verificación, etc. • **T (*Testable*)**: es posible verificar que el alta de un vehículo funciona correctamente a través de pruebas. Por ejemplo, asegurándose de que el administrador pueda registrar un vehículo y que el registro se refleje correctamente en el sistema.		

Table 11: Eliminar un vehículo

Historia de Usuario	
ID: 11	**Rol:** Administrador
Nombre historia: Eliminar un vehículo	
Prioridad en negocio: 6	**Riesgo en desarrollo: 3**
Puntos estimados: 5	**Iteración asignada: 2**
Descripción: Como administrador **quiero** dar de baja una vehículo del inventario **para poder** mantener el inventario actualizado	
Criterios de aceptación: **Escenario 1: identificador del vehículo existe** **Dado que** identificador del vehículo existe **Y que** el administrador tiene el rol adecuado **Cuando** el administrador hace clic en el botón de eliminar **Entonces** el número de vehículo en el inventario serán los mismos menos uno menos el eliminado **Y** el administrador es informado del éxito de la operación **Y** el vehículos no estará disponible para ser alquilado **Escenario 2: identificador del vehículo no existe** **Dado que** identificador del no vehículo existe **Y que** el administrador tiene el rol adecuado **Cuando** el administrador hace clic en el botón de eliminar **Entonces** el número de vehículo en el inventario serán los mismos **Y** el administrador es informado que no existe un vehículo con ese id	
Criterio de calidad: • **I (*Independent*)**: El usuario debe estar autenticado en el sistema y el vehículo dado de alta por lo que depende del requisito ID: 2 y 10 • **N (*Negotiable)***: Los detalles específicos de cómo se implementará la eliminación de un vehículo pueden ser negociables. Por ejemplo, podría haber un proceso de confirmación para asegurar que el usuario realmente desea eliminar el vehículo. • **T (*Testable*)**: Es posible verificar que la eliminación de un vehículo funciona correctamente a través de pruebas. Por ejemplo, asegurándose de que un administrador pueda eliminar un vehículo y que la eliminación se refleje correctamente en el sistema.	

Table 12: Editar la información de una reserva

Historia de Usuario	
ID: 12	**Rol:** Administrador
Nombre historia: Editar la información de una reserva	
Prioridad en negocio: 5	**Riesgo en desarrollo: 3**
Puntos estimados: 5	**Iteración asignada: 3**
Descripción: Como administrador **quiero** editar los datos de una reserva **para poder** hacer las correcciones que sean necesarias en el sistema.	
Criterios de aceptación: **Escenario 1: actualización de reserva exitosa** **Dado que** le administrado está en la vista de gestión de reservas **Y que** identificador de reserva existe **Y que** selecciona una reserva en concreto **Y que** actualiza la información necesaria **Y que** el administrador tiene el rol adecuado **Cuando** el administrador hace clic en el botón guardar cambios **Entonces** le sistema actualiza la información **Y** el administrador es informado del éxito de la operación **Escenario 2: actualización fallida de la reserva** **Dado que** le administrado está en la vista de gestión de reservas **Y que** identificador de reserva existe **Y que** selecciona una reserva en concreto **Y que** actualiza la información necesaria con datos inválidos **Y que** el administrador tiene el rol adecuado **Cuando** el administrador hace clic en el botón guardar cambios **Entonces** el sistema muestra un mensaje informando del error **Y** el administrador puede corregir los campos erróneos de la reserva que estaba editando	
Criterio de calidad: • **I (*Independent*)**: El administrador debe estar autenticado y tener reservas activas en el sistema por lo que depende del requisito ID: 4 y 7 • **N (*Negotiable*)**: Los detalles específicos de cómo se editará la información de una reserva pueden ser negociables. Por ejemplo, qué campos se pueden editar, si se requiere una verificación de contraseña, etc. • **T (*Testable*)**: Es posible verificar que la edición de la información de una reserva funciona correctamente a través de pruebas. Por ejemplo, asegurándose de que un aministrador pueda cambiar su información de reserva y que los cambios se reflejen correctamente en el sistema.	

Table 13: Eliminar una reserva

ID: 13	**Rol:** Administrador
Nombre historia: Eliminar una reserva	
Prioridad en negocio: 5	**Riesgo en desarrollo: 3**
Puntos estimados: 5	**Iteración asignada: 3**
Descripción: como administrador **quiero** eliminar una reserva **para poder** mantener la integridad y precisión de la información relacionada con las reservas, asegurando un funcionamiento eficiente y confiable del sistema.	
Criterios de aceptación: **Escenario 1: Eliminación de una reserva exitosa** **Dado que** le administrado está en la vista de gestión de reservas **Y que** identificador de reserva existe **Y que** selecciona una reserva en concreto **Y que** la reserva está cancelada **Y que** el administrador tiene el rol adecuado **Cuando** el administrador hace clic en el botón "*Delete*" **Entonces** el número de reservas serán las mismas menos una la eliminada **Y** el administrador es informado del éxito de la operación **Escenario 2: Eliminación fallida de la reserva** **Dado que** le administrado está en la vista de gestión de reservas **Y que** identificador de reserva existe **Y que** selecciona una reserva en concreto **Y que** la reserva no esta cancelada **Y que** el administrador tiene el rol adecuado **Cuando** el administrador hace clic en el botón "*Delete*" **Entonces** el sistema muestra un mensaje informando del error **Y** el administrador es informado que solo las reservas canceladas pueden ser eliminadas de la base de datos	
Criterio de calidad: • **I (*Independent*)**: El administrador debe estar autenticado y tener reservas activas en el sistema por lo que depende del requisito ID: 2 y 9 • **N (*Negotiable*)**: Los detalles específicos de cómo se editará la información de una reserva pueden ser negociables. Por ejemplo, qué campos se pueden editar, si se requiere una verificación de contraseña, etc. • **T (*Testable*)**: Es posible verificar que la edición de la información de una reserva funciona correctamente a través de pruebas. Por ejemplo, asegurándose de que un aministrador pueda cambiar su información de reserva y que los cambios se reflejen correctamente en el sistema.	

Table 14: Desactivar la cuenta de usuario por un administrador

Historia de Usuario	
ID: 14	**Rol:** Administrador
Nombre historia: Desactivar la cuenta de usuario por un administrador	
Prioridad en negocio: 6	**Riesgo en desarrollo: 2**
Puntos estimados: 3	**Iteración asignada: 3**
Descripción: Como un administrador **quiero** desactivar la cuenta de un usuario **para poder** gestionar las cuenta de los usuarios según sea necesario.	
Criterios de aceptación: **Escenario 1: el administrador desactiva una cuenta de usuario** **Dado que** el administrador está en la vista de gestión de usuarios **Y** el administrador tiene el rol adecuado **Y** el administrador selecciona un usuario específico **Y** el administrador selecciona "Desactivar cuenta" **Y** el administrador confirma la la acción **Entonces** las cuentas de usuarios activas serán las mismas menos una la cuenta desactivada **Y** el usuario no podrá iniciar sesión **Escenario 2: el administrador cancela la acción de desactivar cuenta de usuario** **Dado que** el administrador está en la vista de gestión de usuarios **Y** el administrador tiene el rol adecuado **Y** el administrador selecciona un usuario específico **Y** el administrador selecciona "Desactivar cuenta" **Y** el administrador cancela la la acción **Entonces** las cuentas de usuarios activas serán las mismas **Y** el usuario podrá iniciar sesión	
Criterio de calidad: • **I (*Independent*)**: el administrador debe estar autenticado en el sistema y el usuario tener una cuenta por lo que depende del requisito ID: 1 y 2 • **N (*Negotiable*)**: los detalles específicos de cómo un administrador desactivará la cuenta de un usuario pueden ser negociables. Por ejemplo, podría haber un proceso de confirmación para asegurar que el administrador realmente desea desactivar la cuenta. • **T (*Testable*)**: es posible verificar que se desactiva de la cuenta de usuario por un administrador funciona correctamente a través de pruebas. Por ejemplo, asegurándose de que un administrador pueda desactivar una cuenta de usuario y que la anulación se refleje correctamente en el sistema.	

Table 15: Activar cuenta de usuario

Historia de Usuario	
ID: 15	**Rol:** Administrador
Nombre historia: Activar cuenta de usuario	
Prioridad en negocio: 6	**Riesgo en desarrollo: 2**
Puntos estimados: 3	**Iteración asignada: 3**
Descripción: Como un administrador **quiero** activar la cuenta de un usuario **para poder** gestionar las cuenta de los usuarios según sea necesario.	
Criterios de aceptación: **Escenario 1: el administrador activa una cuenta de usuario** **Dado que** el administrador está en la vista de gestión de usuarios **Y** el administrador tiene el rol adecuado **Y** el administrador selecciona un usuario con una cuenta desactivada **Y** el administrador selecciona "Activar cuenta" **Cunado** el administrador confirma la la acción **Entonces** las cuentas de usuarios activas serán las mismas más una con los datos indicados **Y** el usuario podrá iniciar sesión	
Criterio de calidad: • **I (*Independent*)**: el administrador debe estar autenticado en el sistema y el usuario tener una cuenta, y esta estar desactivada por lo que depende del requisito ID: 1, 2, 14 • **N (*Negotiable)***: los detalles específicos de cómo se activará la cuenta de un usuario pueden ser negociables. Por ejemplo, podría haber un proceso de confirmación para asegurar que el administrador realmente desea activar la cuenta. • **T (*Testable*)**: es posible verificar que la activación de la cuenta de usuario funciona correctamente a través de pruebas. Por ejemplo, asegurándose de que un administrador pueda activar una cuenta de usuario y que la activación se refleje correctamente en el sistema.	

4.4 Pilares ágiles. Definición de listo y hecho

Finalmente, nos adentramos en dos pilares fundamentales: la "*Definition of Ready*" (DoR) y la "*Definition of Done*" (DoD). Estos términos establecen los criterios que deben cumplirse antes de que una tarea pueda iniciarse (DoR) y después de que se considere finalizada (DoD).

Estos criterios proporcionan un marco de trabajo claro para el equipo de desarrollo, facilitando la planificación, ejecución y entrega de tareas de manera eficiente y efectiva.

Al definir y adherirse a estos criterios, el desarrollador puede elevar la calidad del software y minimizar el trabajo. A continuación, se detallan los criterios específicos aplicados a nuestro proyecto.

DoR

- La definición es precisa y no da lugar a ambigüedades.
- El desarrollador comprende el requisito y cuál es su motivación.
- Todas las dependencias están resueltas.
- Aporta un beneficio al negocio.
- Se han estimado sus puntos de historia.
- Se puede desarrollar en un Sprint.
- La historia es testable y tiene criterios de aceptación claros.
- La historia es validada por el *Product Owner*.

DoD

- El *Product Owner* ha revisado y aprobado la historia de usuario.
- Se han realizado todas las tareas asociadas.
- Se cumple con todos los criterios de aceptación de la historia de usuario.
- Se cumplen los requisitos no funcionales.
- El proyecto sigue los estándares de ingeniería/arquitectura.
- El proyecto compila sin errores e incidencias.
- Se ha documentado y/o comentado el código.
- Se ha completado la refactorización del código.

5 Diseño del sistema

En esta fase, después de haber definido exhaustivamente los requisitos y aspectos funcionales del sistema, avanzamos hacia la elaboración del diseño. Este proceso conlleva la creación de modelos visuales que representan las distintas capas del sistema y sus interrelaciones. Exploraremos elementos clave, como el diseño de las pantallas de usuario, el modelo relacional de la base de datos, los diagramas de clases principales y la arquitectura global del sistema.

5.1 Prototipado de interfaces de usuario

El proceso de prototipado de interfaces de usuario desempeña un papel fundamental en nuestro enfoque de diseño. Desde bocetos iniciales hasta modelos interactivos de alta fidelidad, utilizamos diversas formas de prototipos para visualizar y evaluar la interacción del usuario con el sistema. Este enfoque iterativo nos permite perfeccionar el diseño antes de la implementación, garantizando una experiencia de usuario mejorada y eficiencia en el desarrollo.

Además de su beneficio intrínseco, el prototipado fomenta la comunicación y colaboración efectivas entre los miembros del equipo y los *stakeholders*. Facilita una comprensión más clara de los objetivos y requisitos del sistema. En nuestro proyecto, emplearemos [4] para generar prototipos de interfaces de usuario, aprovechando también un diseño inicial de [5] para la *landing page*. En este sentid, consideramos la carga progresiva de contenidos en una sola página como una opción, adaptable según las necesidades del usuario y abiertos a diferentes implementaciones.

Presentamos a continuación las principales secciones de forma independiente:

1. Encabezado de la página de inicio Figure 10: integra un buscador que permite a los usuarios consultar rápidamente la disponibilidad de vehículos. Ingresando preferencias como fecha, oficina, el sistema muestra opciones relevantes. Este

buscador en el encabezado mejora la experiencia al proporcionar un acceso directo a información clave, enfocándose en la eficacia y la satisfacción del usuario.

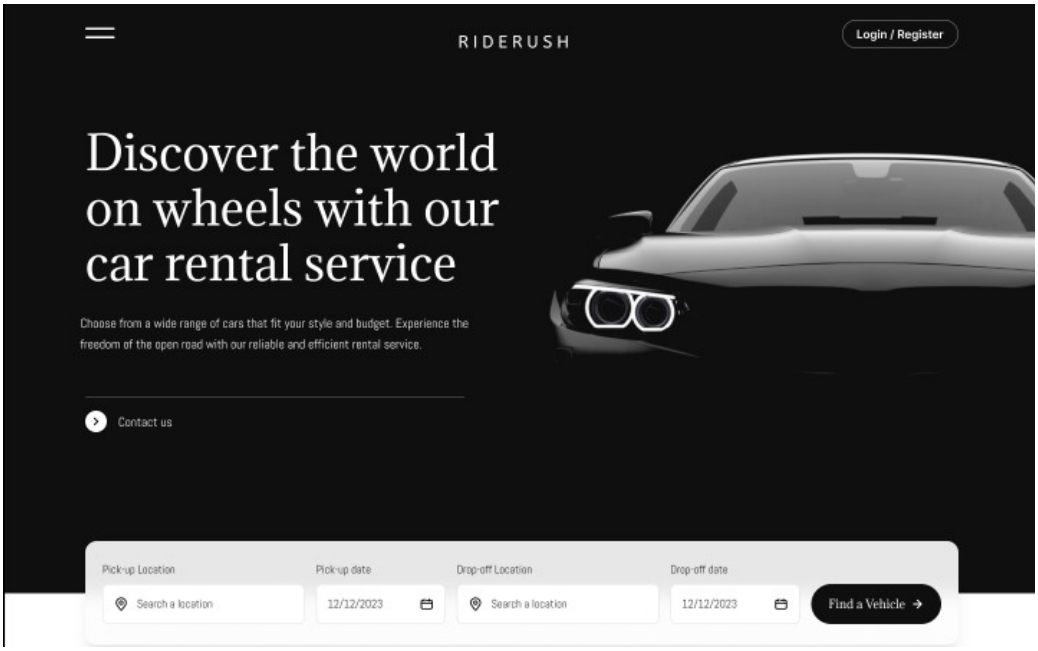


Figure 10: *Landing page*. *Header* con el buscador

2. Sección de selección personalizada Figure 11: la funcionalidad destacada de selección personalizada emerge como un componente crucial. Esta herramienta permite a los usuarios filtrar los vehículos disponibles según marca y tipo, ofreciendo una experiencia de búsqueda altamente personalizada. La meticulosa atención a esta sección tiene como objetivo capacitar a los usuarios, otorgándoles el control para definir criterios específicos que mejoren su experiencia en el alquiler de vehículos.

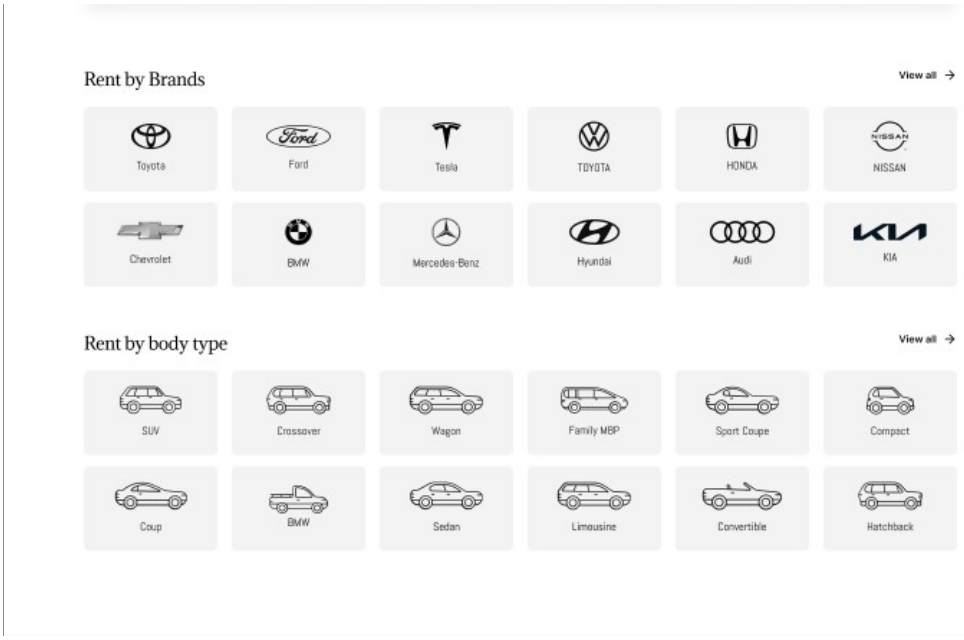


Figure 11: Landing page. Selección personalizada de vehículos por marca y tipo

3. Colección de vehículos destacados Figure 12: en continuidad con la experiencia personalizada, la página de inicio presenta la sección de "Colección de vehículos impresionantes". Este segmento resalta una cuidadosa selección de vehículos notables, ya sea por su rendimiento excepcional, diseño distintivo o características únicas. Desde coches deportivos de alta gama hasta vehículos clásicos o modelos eléctricos de última generación, esta colección diversa busca cautivar la atención de los visitantes. Más que una simple vitrina, esta sección actúa como un reflejo del compromiso de nuestro servicio de alquiler con la calidad y la variedad, motivando a los usuarios a explorar más opciones de alquiler disponibles.

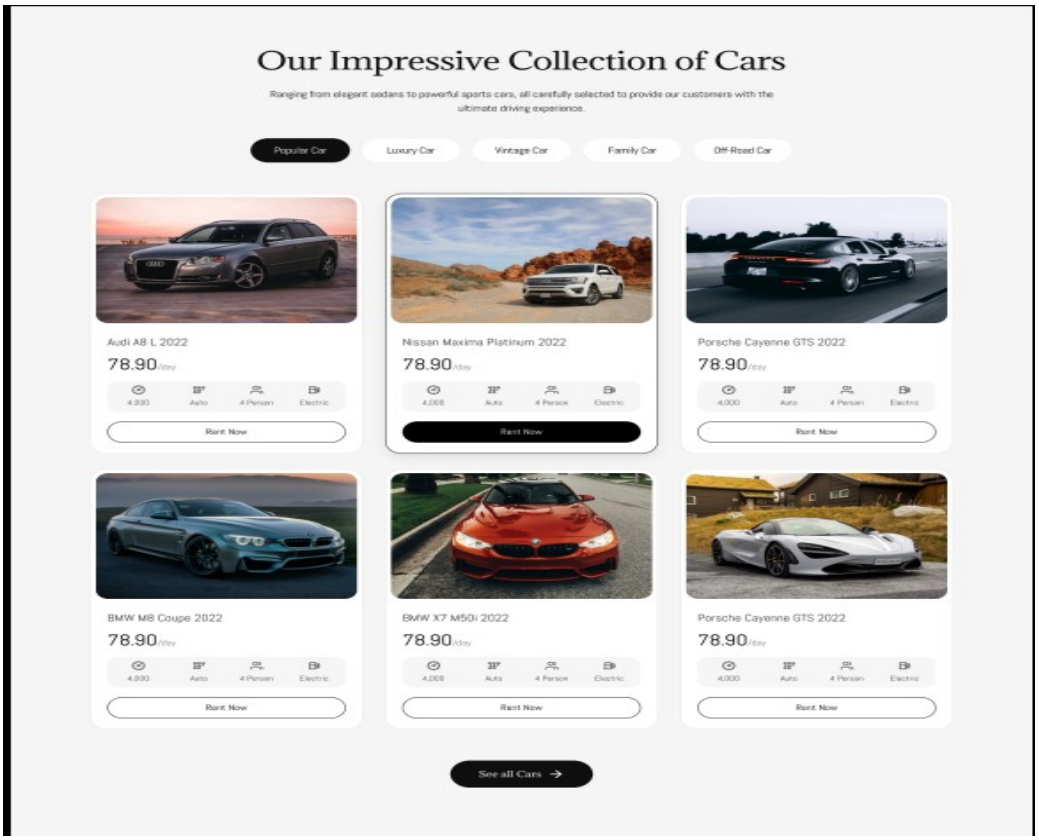


Figure 12: Landing page. Colleción de vehículos impresionantes

4. Flujo del proceso de alquiler Figure 13: Enfocándonos en la claridad y simplicidad, la página de inicio incluye la sección "Flujo del proceso de alquiler". Aquí, proporcionamos a los usuarios tres sencillas pautas que guían el proceso de alquiler de un vehículo. Esta representación visual busca facilitar la comprensión y navegación de los usuarios a lo largo de las etapas esenciales para garantizar un proceso de alquiler eficiente y sin complicaciones.

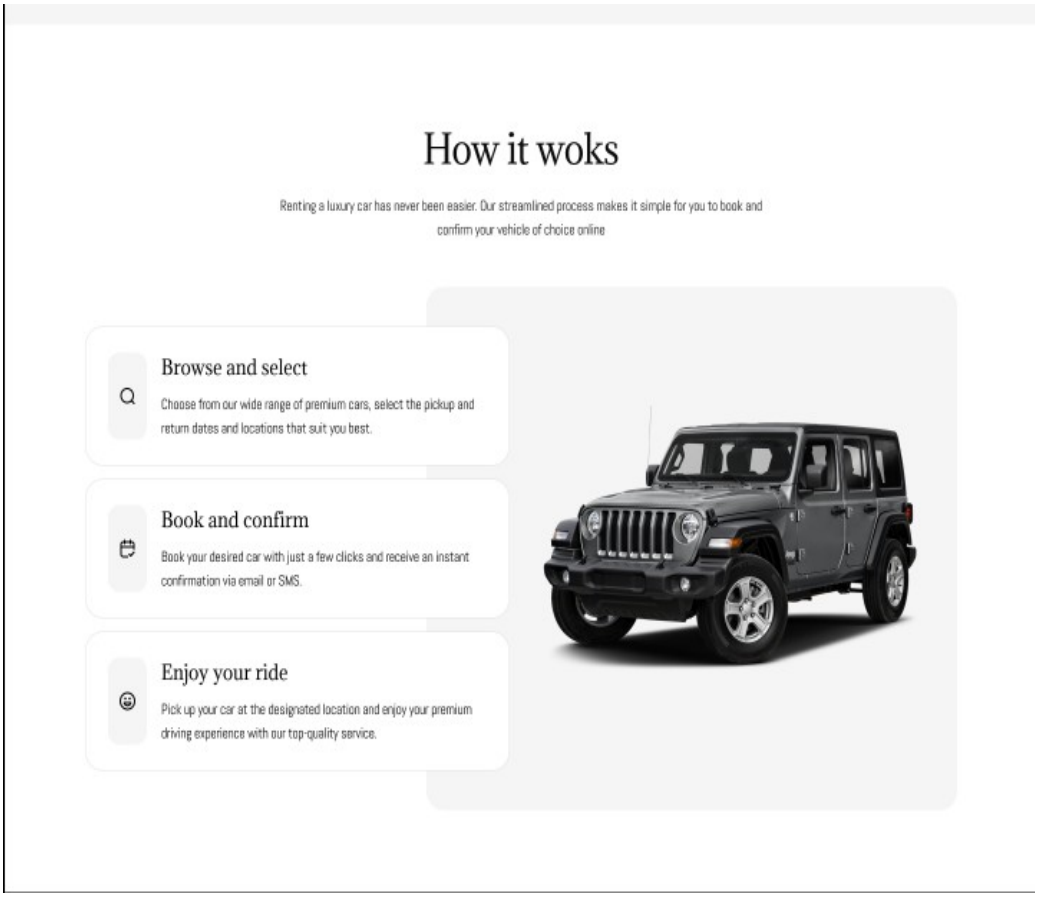


Figure 13: *Landing page*. Flujo del proceso de alquiler

5. Pie de página Figure 14: concluyendo la experiencia del usuario, nuestro diseño incluye un pie de página que no solo actúa como un elemento de navegación adicional sino que también ofrece información esencial. Desde enlaces directos a políticas de privacidad y términos de servicio hasta opciones de contacto, este componente brinda a los usuarios acceso rápido a información relevante y les proporciona la confianza de una experiencia de alquiler transparente y bien gestionada.

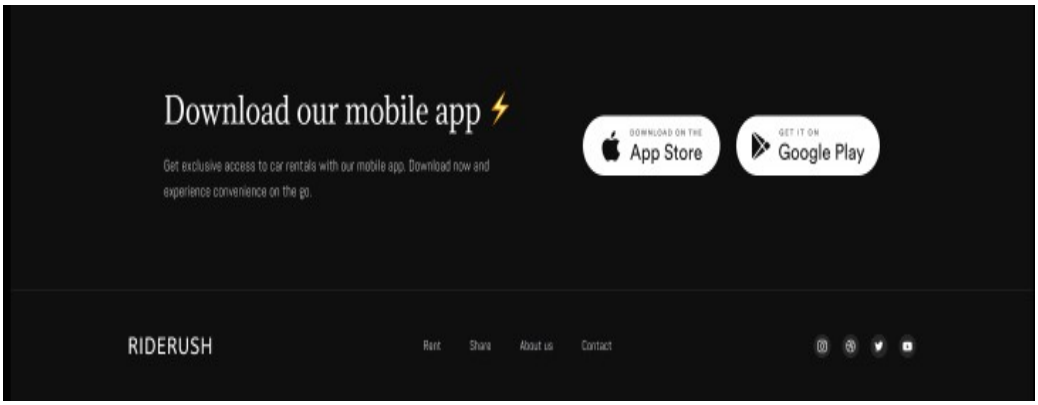


Figure 14: *Landing page. Footer*

6. Por otro lado, el desarrollo del panel de control de las cuentas de usuario esencial para permitir a los usuarios tener un control total sobre su experiencia en la plataforma. En Figure 15 y Figure 16 se presenta dos paneles intuitivo y fácil de usar que ofrece diversas funcionalidades.

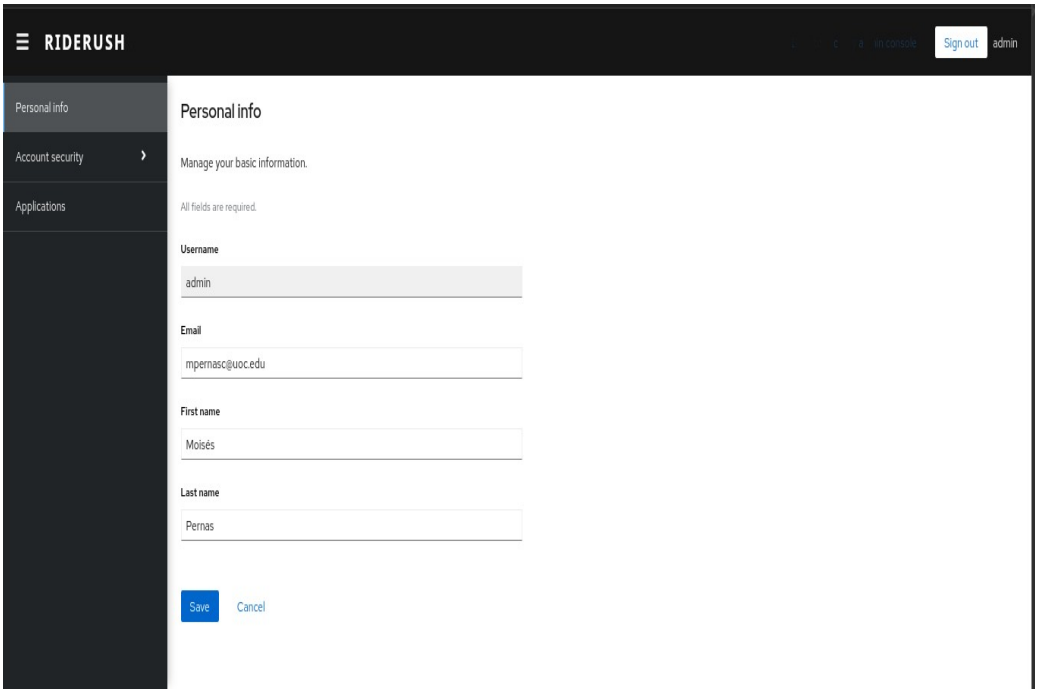


Figure 15: Panel de control. Personalización y edición del perfil de usuario

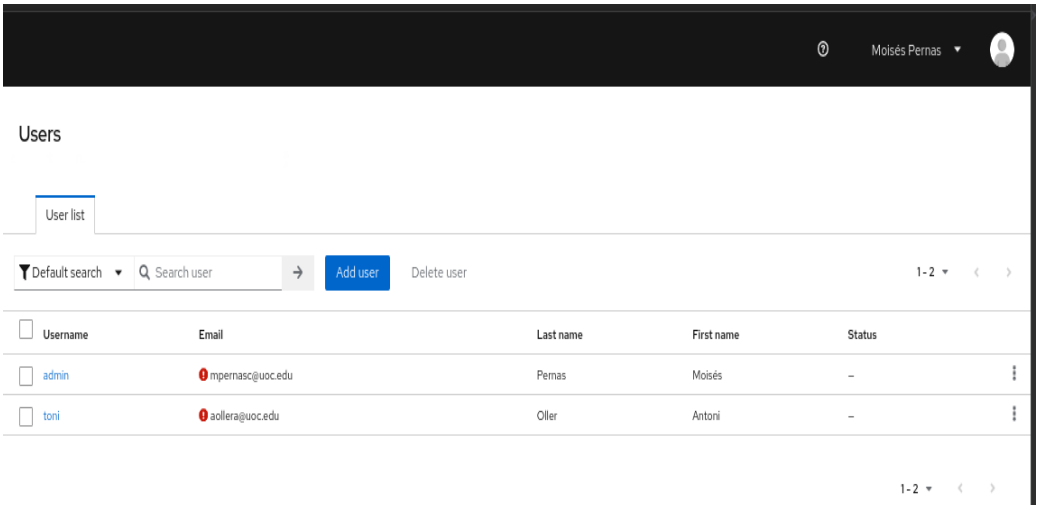


Figure 16: Panel de control. Gestión de las cuentas de usuario

7. Las Figure 17 y Figure 18 muestran dos de las vistas esenciales para el servicio de alquiler de vehículos, centrados en la creación de cuentas y el inicio de sesión. Estos formularios son fundamentales para que los usuarios accedan y gestionen eficientemente sus reservas. Diseñados con énfasis en la facilidad de uso, garantizan una experiencia intuitiva, permitiendo a los usuarios aprovechar rápidamente nuestros servicios con eficiencia y comodidad.

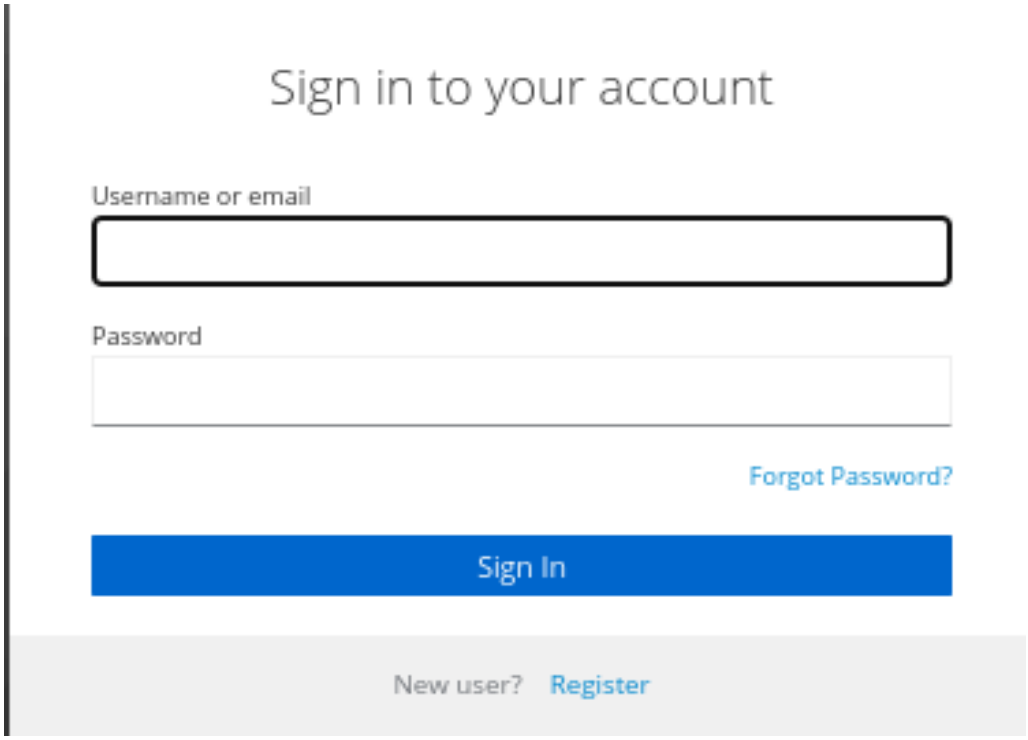


Figure 17: Formulario de *login*

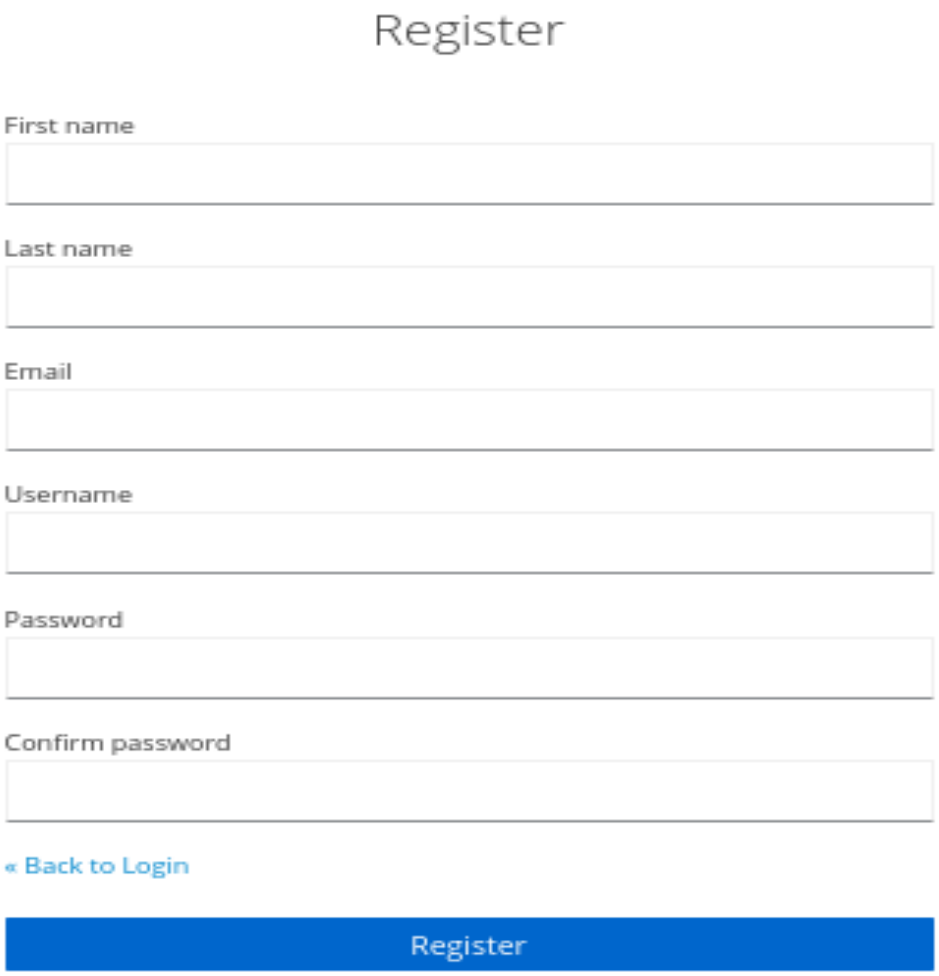


Figure 18: Formulario de registro

Finalmente, aunque se ha presentado el prototipado de las diferentes interfaces como ejemplos representativos, es importante destacar que no ha sido posible completar exhaustivamente todas las interfaces de usuario dentro del plazo establecido. La prioridad se centró en el desarrollo y operatividad del *backend* para todas las funcionalidades. Estas interfaces visuales sirven como herramientas ilustrativas para

comprender el flujo del sistema, pero el foco principal del trabajo reside en la arquitectura de microservicios y en abordar los desafíos técnicos asociados.

5.2 Esquema relacional de la base de datos

En el marco de la arquitectura propuesta para el desarrollo del proyecto, detallada en 5.4, se establecen bases de datos o esquemas que ostentan una naturaleza simple y desacoplada para cada uno de los microservicios. Esta elección se fundamenta en el principio de que cada microservicio encapsula un dominio de negocio específico, y la base de datos se configura como una implementación detallada de dicho dominio.

La simplicidad y desacoplamiento de los esquemas de base de datos posibilitan el desarrollo, despliegue, escalabilidad y mantenimiento independiente de cada microservicio. Asimismo, este enfoque concede la flexibilidad de elegir la tecnología de base de datos más idónea para las necesidades particulares de cada microservicio, sin la restricción de adherirse necesariamente a bases de datos relacionales. En consecuencia, la implementación de un esquema relacional complejo no resulta imperativa en esta arquitectura.

No obstante, se proporciona en la Figure 19 un esquema relacional para el sistema, con la salvedad de que esta representación es una simplificación. En la práctica, la persistencia de los datos variará según la base de datos utilizada por cada microservicio.

Además, en lugar de recurrir a claves foráneas para gestionar las relaciones entre las entidades, se optará por el uso de índices. Esta elección se justifica por la distribución potencial de las entidades en diferentes servicios y bases de datos dentro de un sistema de microservicios. Los índices posibilitarán referenciar eficientemente estas entidades sin la necesidad de establecer relaciones directas entre las bases de datos.

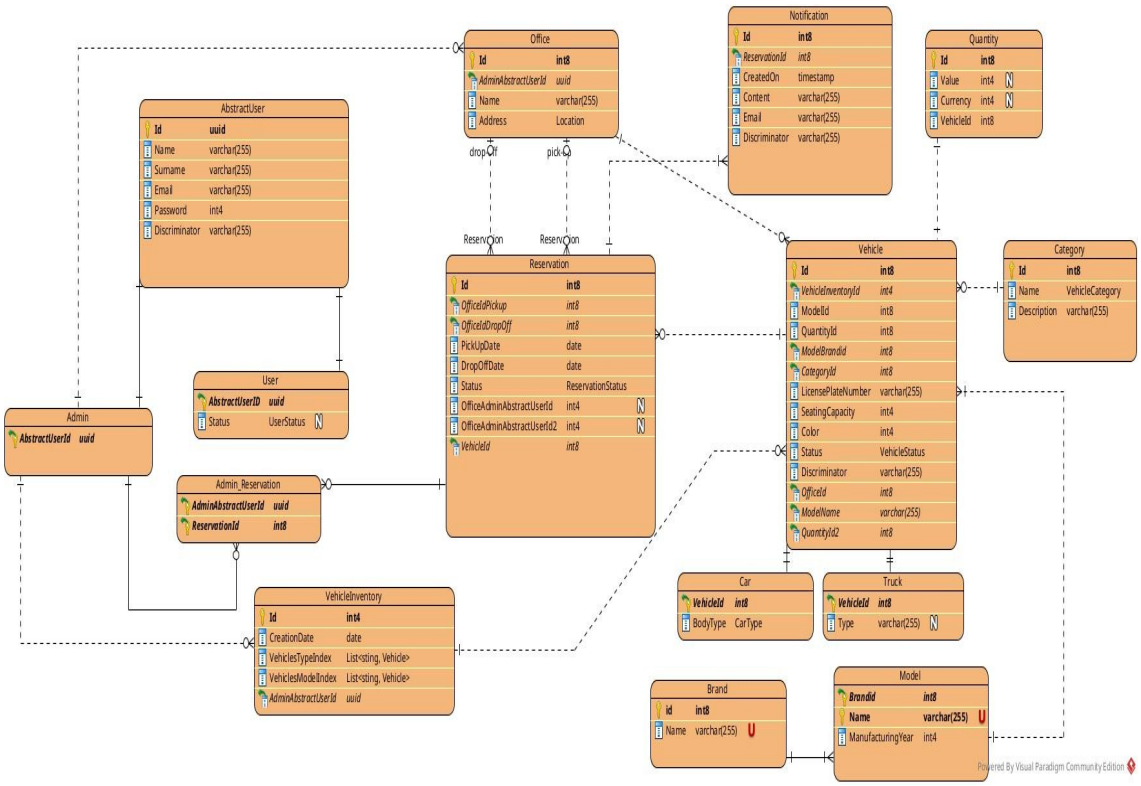


Figure 19: Esquema relacional

5.3 Diagrama de clases del dominio

Por otro lado, y con el propósito de desarrollar un sistema fácil de mantener y altamente flexible, hemos confeccionado un diagrama de clases que brinda una representación visual de la estructura de nuestro sistema, delineando las clases y sus relaciones.

El diagrama de clases no solo sirve como una representación gráfica del código, sino que también actúa como una herramienta analítica que facilita la identificación de posibles problemáticas y oportunidades de mejora en la arquitectura del sistema. En Figure 20, se presenta el diagrama de clases desde la perspectiva de la información.

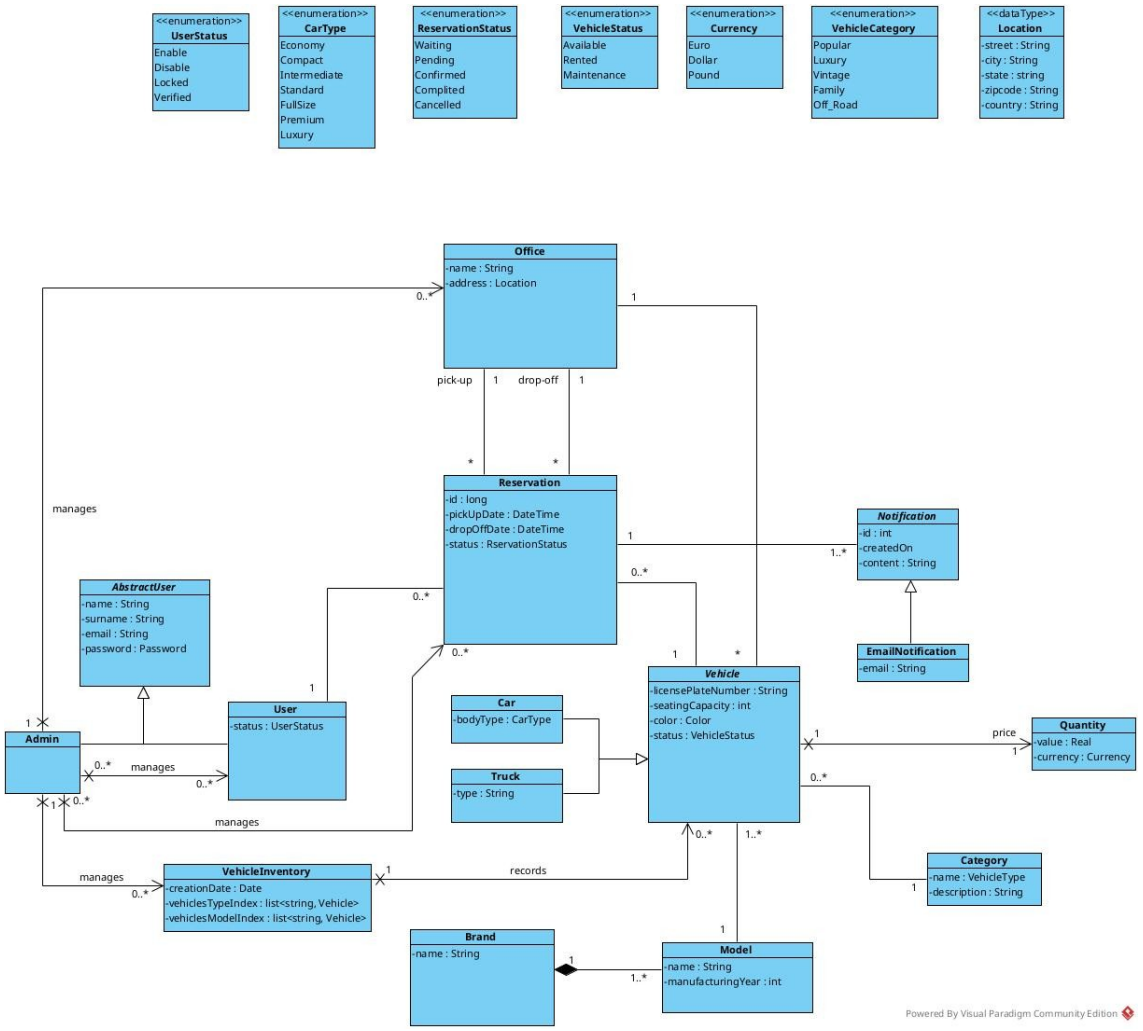


Figure 20: Diagrama punto de vista de la información

En este diseño:

- Modelamos las reservas y los vehículos como clases con una asociación. La multiplicidad de la asociación indica que una reserva solo puede estar asociada con un vehículo, en cumplimiento con una regla de negocio específica.
- Se asume que todos los usuarios del servicio poseen un permiso de conducir válido, estableciendo así una restricción de integridad implícita.

- Adicionalmente, se establece la restricción de que la fecha de devolución debe ser posterior a la fecha de recogida, asegurando la coherencia temporal de las operaciones en el sistema.

Este diagrama no solo es una representación visual, sino una herramienta esencial para guiar el desarrollo del software, destacando las relaciones esenciales entre las entidades y las restricciones clave que deben cumplirse en el sistema.

5.4 Arquitectura del sistema

El enfoque de desarrollo para este proyecto se sustenta en los principios SOLID [6], con especial énfasis en el principio de Responsabilidad Única (SRP). Este principio cobra una relevancia central en la arquitectura de microservicios, donde cada microservicio se dedica a una única responsabilidad o funcionalidad, generando servicios pequeños, independientes y autónomos.

Al explorar en detalle la aplicación de estos principios, surge la arquitectura hexagonal como un componente clave. Este modelo propone una separación efectiva entre la lógica de negocio y la infraestructura, visualizando cada microservicio como un hexágono con puertos de entrada y salida, facilitando la interacción con interfaces de usuario y servicios externos.

La convergencia de la arquitectura de microservicios con la arquitectura hexagonal ofrece un nivel sobresaliente de desacoplamiento y modularidad. Esta integración no solo facilita el desarrollo, las pruebas y el despliegue de cada microservicio de manera independiente, sino que también proporciona una guía clara para el desarrollo del software, enlazando de manera coherente los principios SOLID con la estructura arquitectónica del sistema.

5.5 Descripción general de alto nivel

En la Figure 6, se presenta un esquema de alto nivel que abarca los microservicios y tecnologías fundamentales para el desarrollo del proyecto. El servicio API Gateway asume el rol de punto de entrada principal para toda la arquitectura. Este componente se conecta al servicio de descubrimiento Eureka para obtener la ubicación precisa de cada microservicio, seguido de la invocación al microservicio específico correspondiente. El API Gateway desempeña también un papel central en la autenticación de usuarios, aunque la autorización se delega a los microservicios individuales.

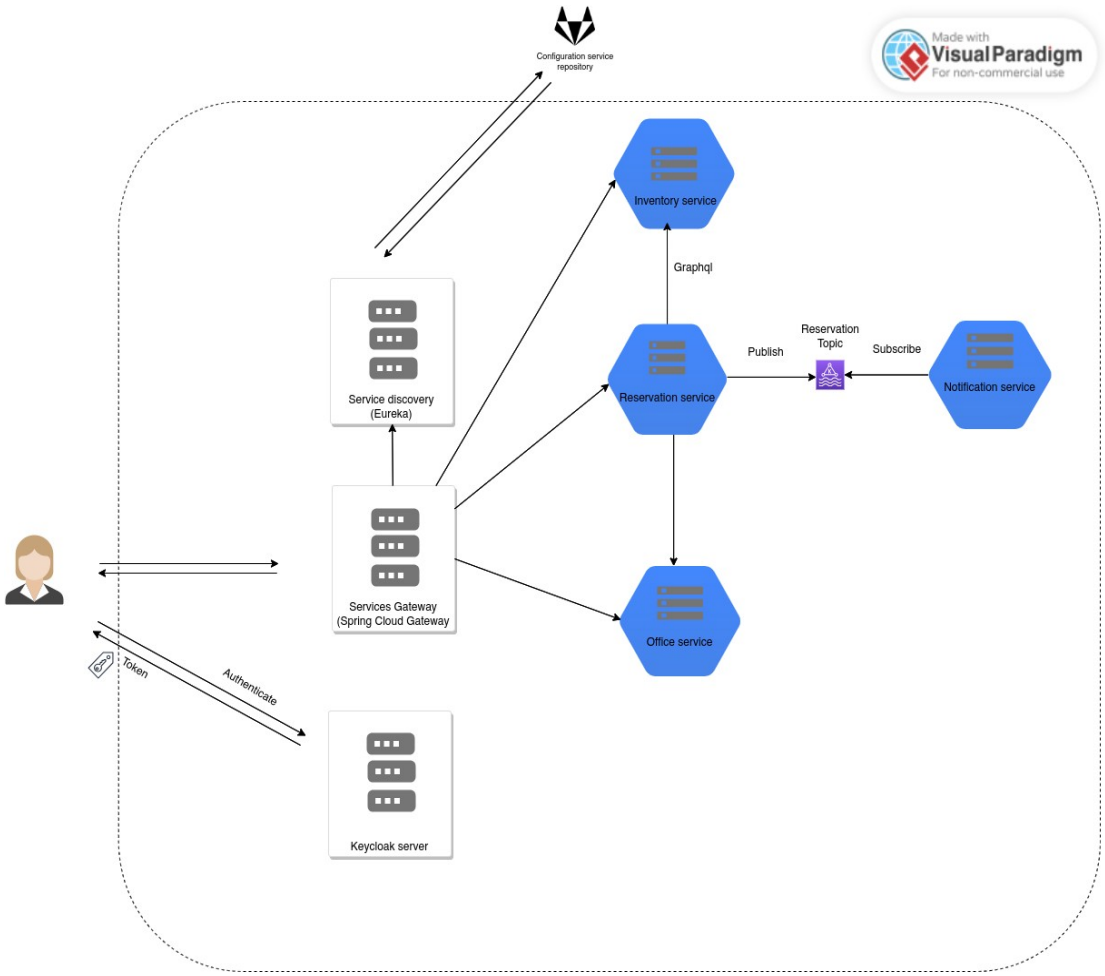


Figure 21: Descripción general de alto nivel de los servicios

En el proceso de reserva de un vehículo, la autenticación del usuario a través de Keycloak es esencial para obtener un token de acceso, posteriormente validado por el microservicio de reservas. Este procedimiento se replica en diferentes flujos, y dependiendo del rol del usuario, se le concede o deniega ciertos privilegios, como la actualización del inventario de vehículos o la gestión de oficinas. Al completar una reserva, el microservicio de reservas publica un mensaje en un tópico de Kafka, activando el servicio de notificaciones para enviar un correo electrónico al usuario con los detalles de la reserva.

Adicionalmente, se implementa un servidor de configuración (Spring Cloud Config) con el propósito de centralizar en un único punto (GitLab) la configuración de perfiles (desarrollo y producción) para cada microservicio. Esta estrategia simplifica la gestión de cambios en las propiedades de cada microservicio, asegurando coherencia y eficiencia en la administración de la configuración.

5.6 Diagrama de arquitectura de microservicios

Para ilustrar la estructura de nuestra arquitectura de microservicios, hemos optado por implementar el modelo C4, diseñado por Simon Brown [7]. Este enfoque se destaca como una herramienta valiosa para comprender la arquitectura de productos de software. Basado en el principio de ir de lo general a lo particular, el modelo C4 facilita un acercamiento progresivo a cada elemento significativo del sistema. Este modelo establece cuatro niveles de especificación:

1. Diagrama de contexto: proporciona una perspectiva de alto nivel, mostrando las interacciones entre el sistema y sus entornos externos.
2. Diagrama de contenedores: se adentra en la estructura interna del sistema, identificando los contenedores y sus relaciones, proporcionando una visión más detallada que el diagrama de contexto.

3. Diagrama de componentes: ofrece una vista detallada de los componentes internos de cada contenedor, mostrando cómo se relacionan y comunican entre sí.
4. Diagrama de código: se centra en detalles específicos a nivel de implementación, proporcionando una visión detallada del código fuente y sus relaciones.

Esta jerarquía de niveles en el modelo C4 posibilita a los equipos obtener una comprensión integral de la arquitectura, desde una visión panorámica hasta una exploración minuciosa de los aspectos más específicos del sistema, a continuación, exploramos más detalladamente cada uno de estos niveles

5.6.1 Diagrama de contexto

En Figure 22 se presenta el diagrama de contexto. Esta representación inicial proporciona una visión clara de las responsabilidades de nuestro sistema, así como de las relaciones clave con los usuarios y otros sistemas con los que interactúa. Este diagrama sirve como punto de partida para explorar más a fondo la arquitectura general del sistema y sus conexiones fundamentales.

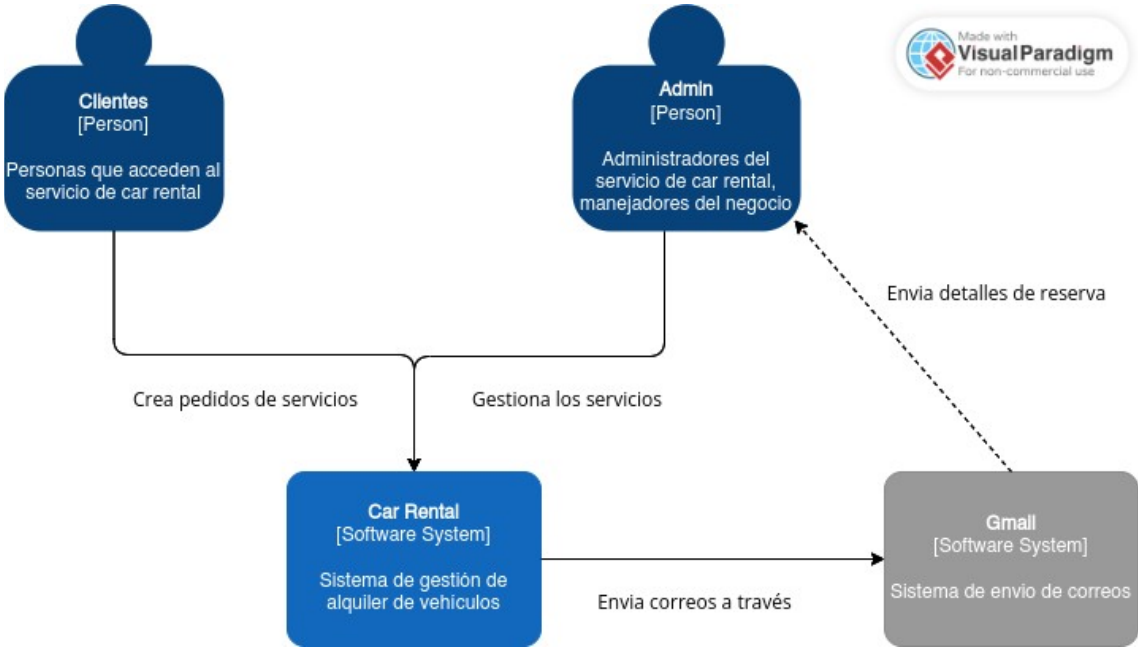


Figure 22: Diagrama de contexto

5.6.2 Diagrama de contenedores

En Figure 23, presentamos un análisis más detallado de nuestro sistema mediante el diagrama de contenedores. Este artefacto visual proporciona una visión exhaustiva de los bloques técnicos de alto nivel que constituyen nuestro sistema. Cada contenedor representa una unidad lógica independiente que encapsula componentes y servicios relacionados. Observamos cómo estas entidades técnicas distribuyen y comparten responsabilidades entre sí, ofreciendo una comprensión más profunda de la arquitectura subyacente.

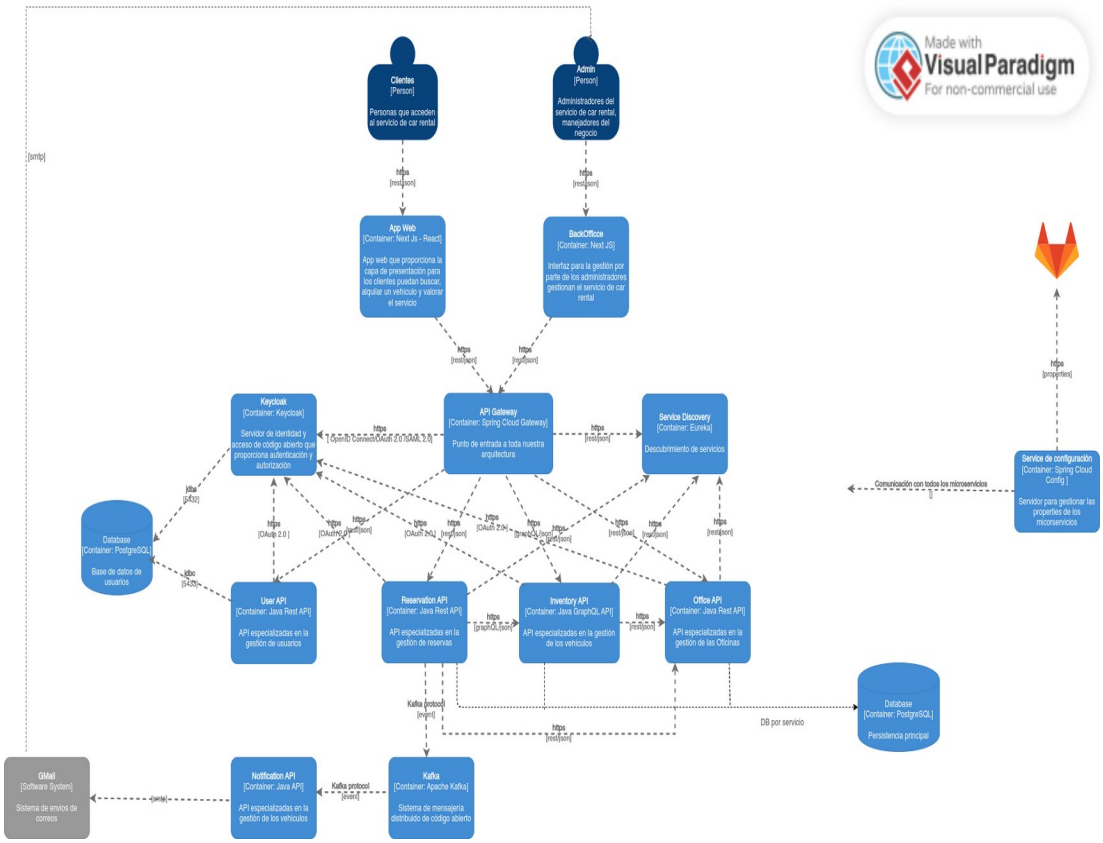


Figure 23: Diagrama de contenedores

Los contenedores identificados en este diagrama representan puntos clave de interacción y colaboración dentro del sistema. Se destacan las relaciones y comunicaciones entre los distintos contenedores, permitiendo una apreciación más clara de cómo se integran para lograr los objetivos del sistema. Este enfoque detallado sienta las bases para el siguiente nivel de especificación, el diagrama de componentes, donde exploraremos más a fondo la estructura interna de cada contenedor.

5.6.1 Diagrama de componentes

En el diagrama de componentes, representado en Figure 24 se examina de manera minuciosa la interacción y relaciones entre los componentes alojados en el contenedor de reservas. Este nivel de detalle nos sumerge en la implementación concreta del

sistema, permitiendo una comprensión profunda de cómo los elementos individuales colaboran para cumplir con las funcionalidades asignadas.

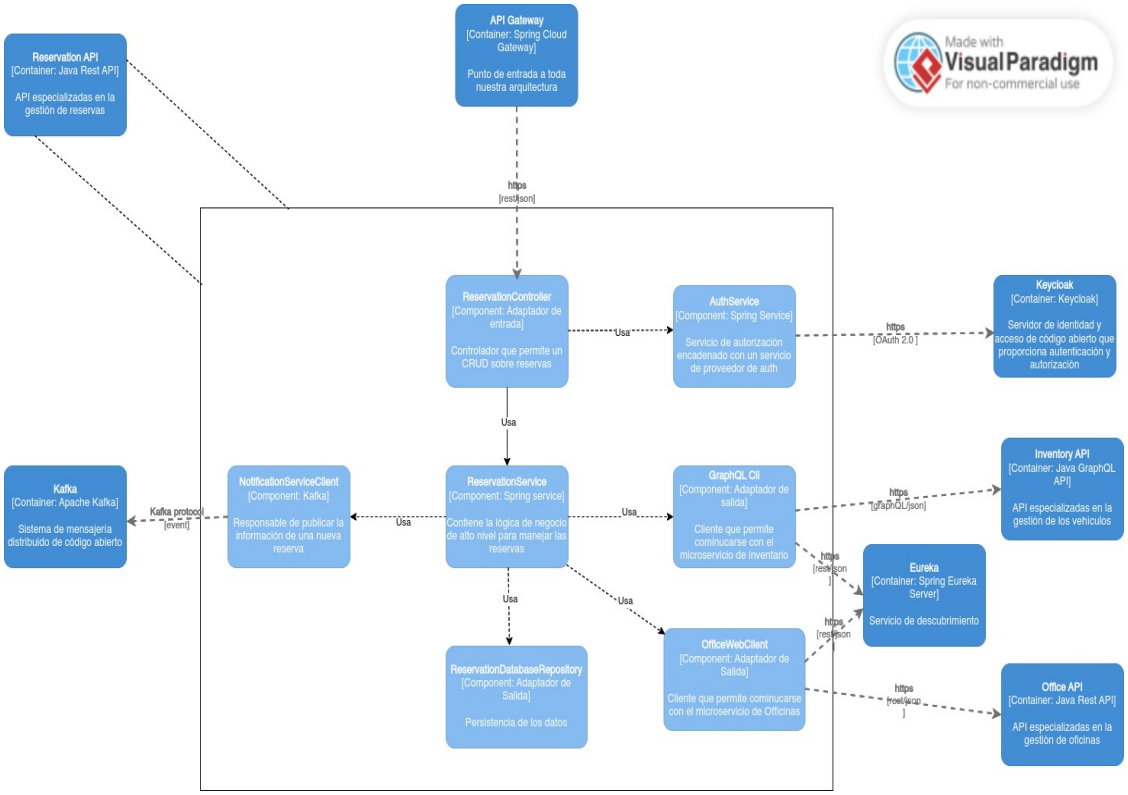


Figure 24: Diagrama de componentes

Cada componente identificado en el diagrama representa una unidad lógica distinta del sistema, desglosando las responsabilidades específicas y las interconexiones entre ellos. Además de explorar la interacción interna, el diagrama también destaca las relaciones externas, mostrando cómo los componentes del contenedor de reservas se comunican con otros contenedores o sistemas.

Esta visión detallada del nivel de componentes proporciona una base sólida para analizar la estructura interna y las dependencias funcionales del sistema. A medida que profundizamos en este nivel de abstracción, obtenemos información valiosa sobre la

implementación efectiva de las características y servicios que componen el módulo de reservas.

Por último, se ha decidido consolidar la lógica de negocio en un único servicio que abarque tanto la capa de aplicación como la capa de dominio debido a la simplicidad de la lógica del sistema. Sin embargo, se reconoce que, en escenarios de mayor complejidad, sería más apropiado implementar servicios separados para gestionar estas capas.

6 Implementación

Completado el análisis y el diseño robusto de RideRush: plataforma integral de alquiler y gestión de flotas turísticas, estamos preparados para ingresar a la fase de implementación. Esta fase critica del proyecto marca el paso de la conceptualización a la materialización, donde las líneas de código se convertirán en la columna vertebral de la solución propuesta.

A lo largo de esta sección, exploraremos la transformación de los conceptos y diagramas previos en un sistema funcional y eficiente, abordando los retos técnicos y afinando cada componente para asegurar una ejecución coherente con la visión estratégica delineada en etapas anteriores.

6.1 Elecciones tecnológicas

En la travesía de RideRush hacia la vanguardia tecnológica, cada herramienta y tecnología seleccionada desempeña un papel clave en la construcción de nuestra plataforma integral de alquiler y gestión de flotas turísticas. Desde el entorno de desarrollo IntelliJ IDEA hasta el sistema de mensajería distribuido Apache Kafka, cada elección está estratégicamente alineada con nuestra visión de ofrecer servicios altamente personalizados y eficientes en el competitivo mercado del alquiler de vehículos.

Estas tecnologías, cuidadosamente seleccionadas, forman el tejido digital que potenciará la experiencia de RideRush, permitiéndole no solo mantenerse, sino liderar en este dinámico panorama empresarial. A continuación, presentamos un resumen conciso de las herramientas y tecnologías que impulsarán nuestro camino hacia la innovación.

- IntelliJ IDEA [8] es nuestro entorno de desarrollo integrado (IDE), brindando a los desarrolladores un conjunto completo de herramientas para el desarrollo de software.
- GitLab [9] es nuestra plataforma integral que proporciona un conjunto completo de herramientas para el desarrollo de software, facilitando la colaboración y la gestión del ciclo de vida del desarrollo.
- Apache Maven [10] es nuestra herramienta de automatización para construcción, informes y documentación de proyectos de software. Simplifica la construcción y distribución, facilitando la colaboración y la creación de proyectos escalables y mantenibles.
- Spring [11], un marco de trabajo integral para Java, nos brinda funcionalidades esenciales para el desarrollo de aplicaciones web, de microservicios, de datos y de integración. Utilizamos Spring Boot, Spring Cloud, Spring Data y Spring Security para diferentes aspectos del proyecto.
- Podman [12] es nuestro motor de contenedores, ofreciendo una solución de contenedorización segura, flexible y compatible. No requiere privilegios de root para ejecutarse, brindando mayor seguridad y flexibilidad.
- Apache Kafka [13] es nuestro sistema de mensajería distribuido de alta escalabilidad y rendimiento, diseñado para procesar grandes cantidades de datos en tiempo real.
- Apache Zookeeper [14] actúa como nuestro servicio de coordinación de clusters distribuido, proporcionando servicios básicos para la coordinación, elección de líderes y gestión de grupos.

- Keycloak [15], nuestro servidor de identidad de código abierto, ofrece una solución integral para autenticación y autorización en aplicaciones web y móviles, garantizando seguridad, escalabilidad y facilidad de uso.
- Next.js [16], herramienta para la construcción de la interfaz de usuario de RideRush, confiando en su capacidad para ofrecer una experiencia de usuario moderna, ágil y altamente receptiva.

6.2 Sprint 1: Seguridad y personalización

Completado la instalación de las herramientas. Nos adentramos en los procesos cruciales de registro, autenticación, recuperación de contraseña y edición de perfiles en RideRush, es decir, historias con ID: 1, 2, 3 y 4. Se opta por integrar Keycloak , pues permitirá que, RideRush se beneficie de características clave como la federación de identidad, soporte para protocolos estándar como OAuth 2.0 y OpenID Connect, y una interfaz de administración intuitiva.

Este enfoque basado en Keycloak no solo fortalece la seguridad y la integridad del sistema, sino que también simplifica la administración de usuarios y sus credenciales. Los usuarios de RideRush experimentarán un proceso de registro y autenticación fluido y seguro, respaldado por la robustez y la confiabilidad de Keycloak. Además, la implementación de Keycloak permite una fácil expansión y personalización en futuras etapas del proyecto, asegurando la adaptabilidad de RideRush a medida que evolucionan las necesidades y tecnologías de autenticación.

6.2.1 Creando el realm de RideRush

En primer lugar, comenzamos configurando el entorno Keycloak. En este primer paso, descargamos la imagen específica de Keycloak:21.1.1. A continuación, en nuestro archivo docker-compose.yml, ajustamos las variables y conectamos Keycloak a su base

de datos PostgreSQL. Previamente, hemos creado y montado un volumen[1]. Con esta base, RideRush está listo para definir su propio realm y avanzar en el proceso de seguridad.

Después de la creación del realm, procedemos a la configuración de los clientes, dos en este contexto específico: uno destinado al *frontend* y otro al *backend*. Figure 25 Seguidamente, llevamos a cabo la creación de los roles del realm, concretamente "realm-admin" y "realm-user" Figure 26. Este paso es esencial para establecer las distintas capas de acceso y autoridad en RideRush, asegurando una gestión eficiente y segura de los usuarios y sus roles en la plataforma.

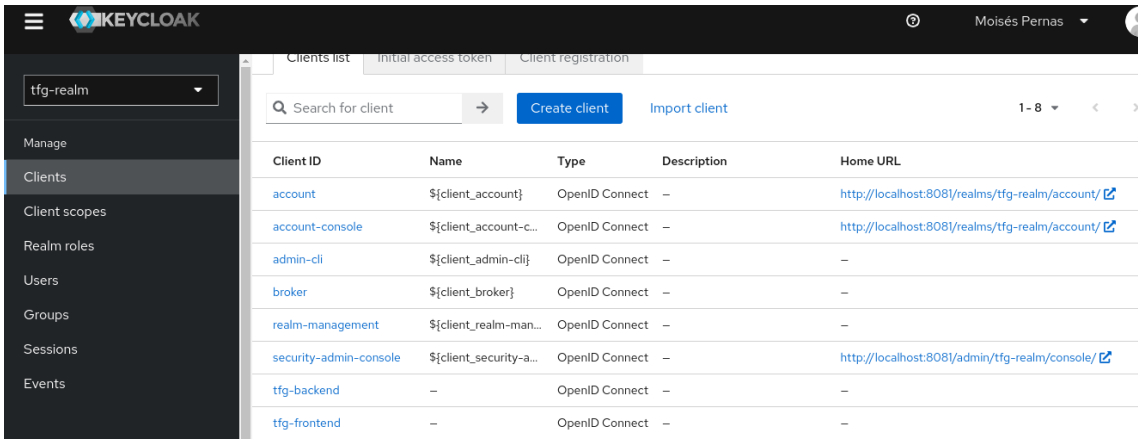


Figure 25: Panel de configuración del realm. Creación de clientes

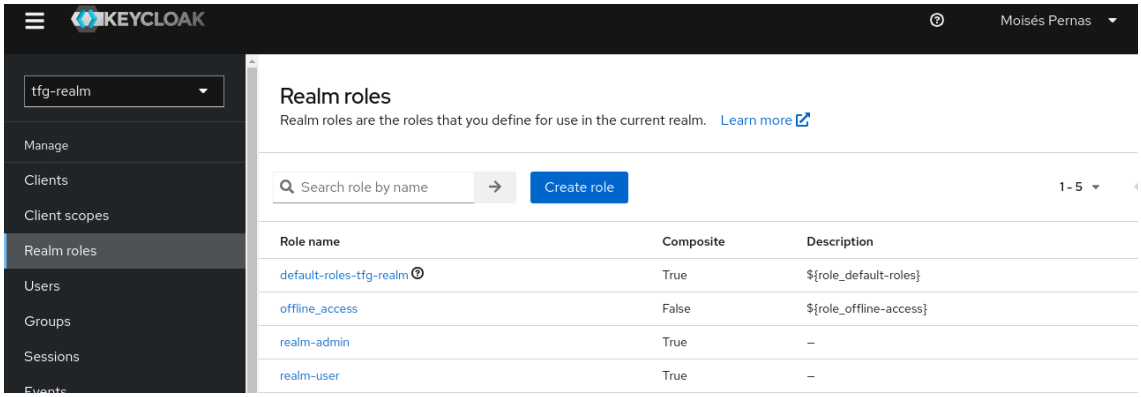


Figure 26: Panel de configuración del realm. Creación de roles

1 El propietario y el grupo de este volumen debe ser 100998:100998, para que se monte correctamente

Al culminar la configuración, el proveedor de identidad queda preparado para administrar operaciones fundamentales como registro, autenticación, autorización y edición de perfiles en RideRush. Únicamente restará ajustar la configuración del cliente, crear los componentes necesarios y gestionar el refresco automático de los tokens que la API utilizará Figure 27, completando así el proceso de implementación y optimización del sistema de seguridad.

En este sentido, señalar que las nuevas cuentas de usuarios son verificadas. Por tanto, en el proceso de registro, el usuario debe proporcionar una dirección de correo electrónico válida a la que se le enviará un mensaje con un enlace de confirmación. El usuario deberá hacer clic en el enlace dentro de los 5 minutos siguientes para activar su cuenta y acceder al servicio.

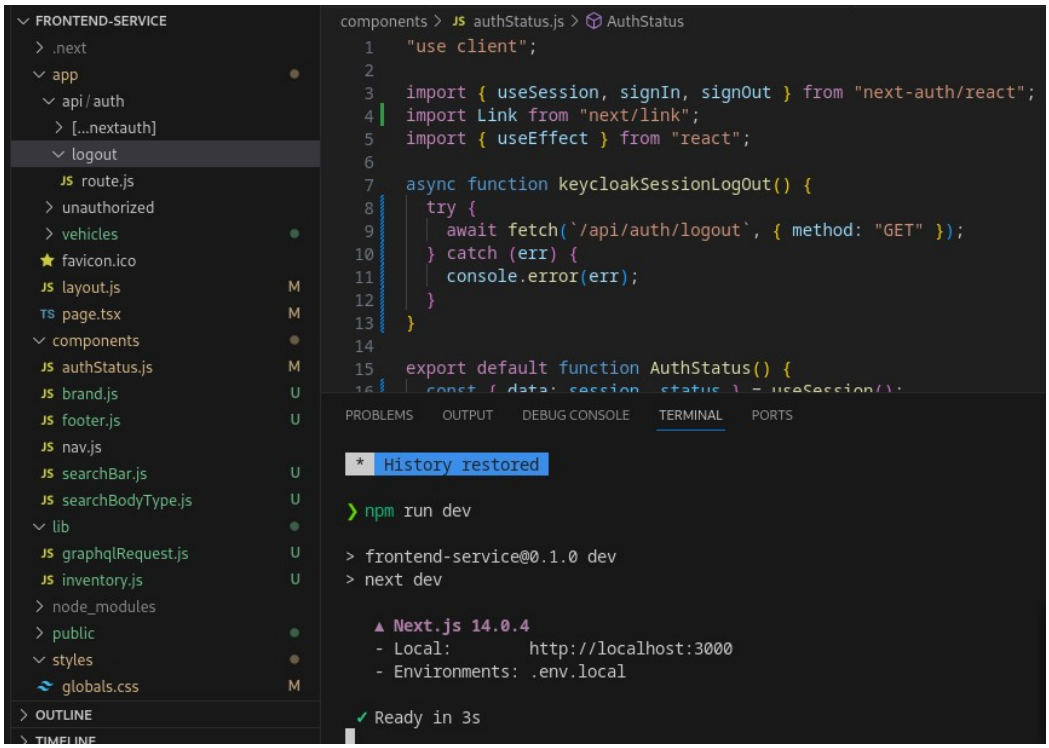


Figure 27: Next js. Estructura del proyecto

En Figure 28 se aprecia el usuario autenticado. Al hacer clic en su nombre lo diseccionará a la vista de su perfil usuario para que edite la información, si así lo estima. Además, al interactuar con la diferentes vistas se podrá comprobar que se cumple cada uno de los escenarios propuesto en las historias de usuario.

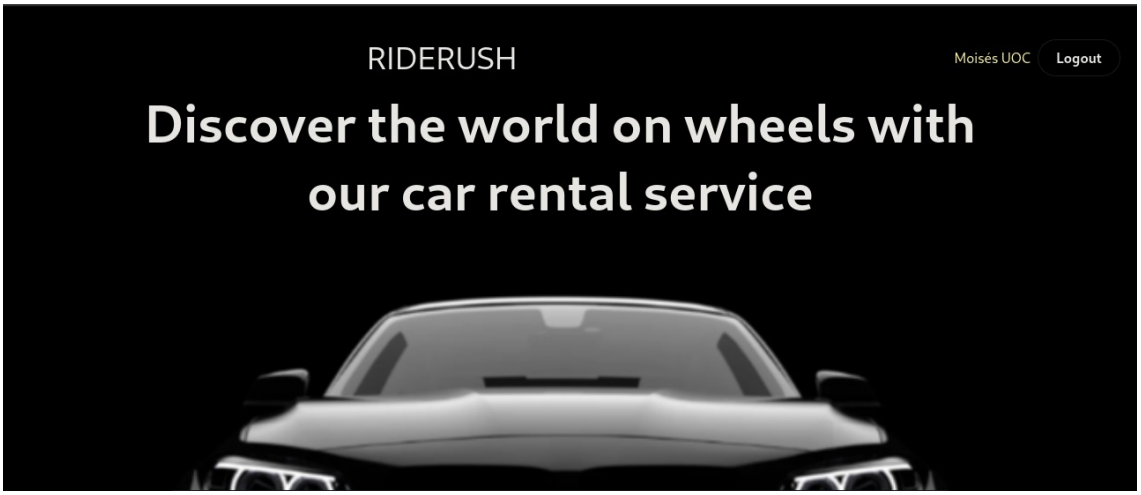


Figure 28: RideRush usario autenticado

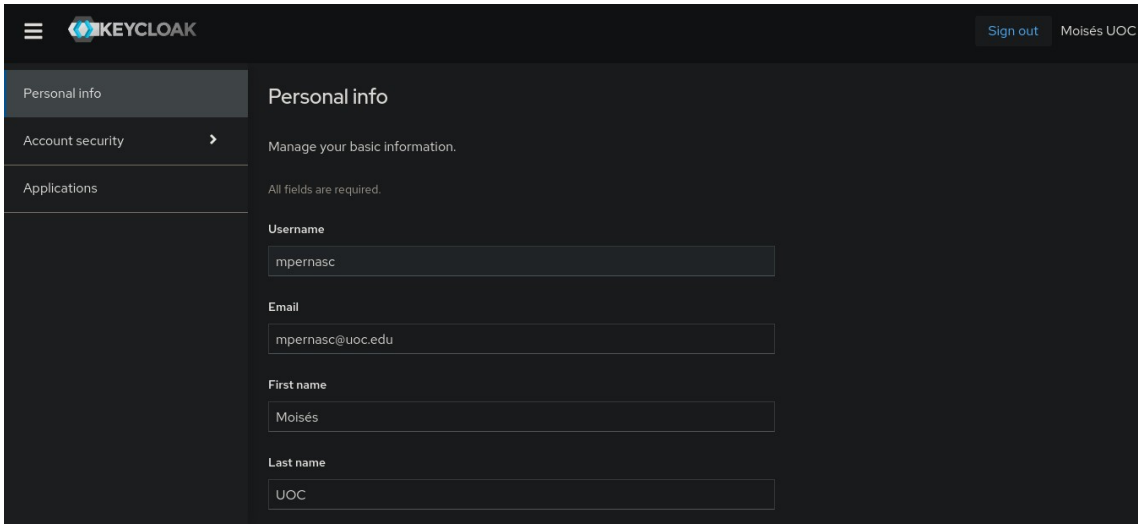


Figure 29: Vista del perfil de usuario

6.2.2 Sentando las bases para la gestión de la flota

Para culminar el Sprint, nos enfocamos en la implementación de las historias con ID 5 y 10, en la que sentamos las bases para el desarrollado el servicio de inventario, clave para la gestión de la flota de RideRush. Hemos utilizado GraphQL para exponer los *endpoints*, logrando una interacción eficiente y flexible con la base de datos y los datos de la flota y evitando el problema de *over-fetching* o *under-fetching* de información.

Además, hemos incorporado el patrón de estrategia para las operaciones de búsqueda, permitiendo cambiar los algoritmos según los requisitos. También, habilitamos un *endpoint, ver [17],* para GraphiQL, que facilita la exploración y comprensión de la API GraphQL desde el navegador. Con estas tecnologías y prácticas, hemos demostrado nuestro compromiso con la innovación y la excelencia en el desarrollo de RideRush, creando un sistema eficiente, modular y documentado.

6.3 Sprint 2: Completando la lógica en la gestión de flotas y primeros pasos en la gestión de reservas

En este Sprint, continuamos con en el desarrollo del microservicio de inventario, implementando las historias con ID 6, 7 y 11, que consisten en añadir nuevas estrategias de consulta para el servicio. Estas estrategias permiten obtener un vehículo de la flota por su identificador, por su oficina de origen, destino o disponibilidad. Estas consultas son esenciales para el funcionamiento de los microservicios de reservas y oficinas, que dependen de la información del inventario para realizar sus operaciones, Figure 30. Con estas estrategias de consulta, RideRush logra una mayor eficacia y rapidez en la gestión de la flota, ofreciendo a sus clientes un servicio de alquiler de vehículos turísticos personalizado y de calidad.

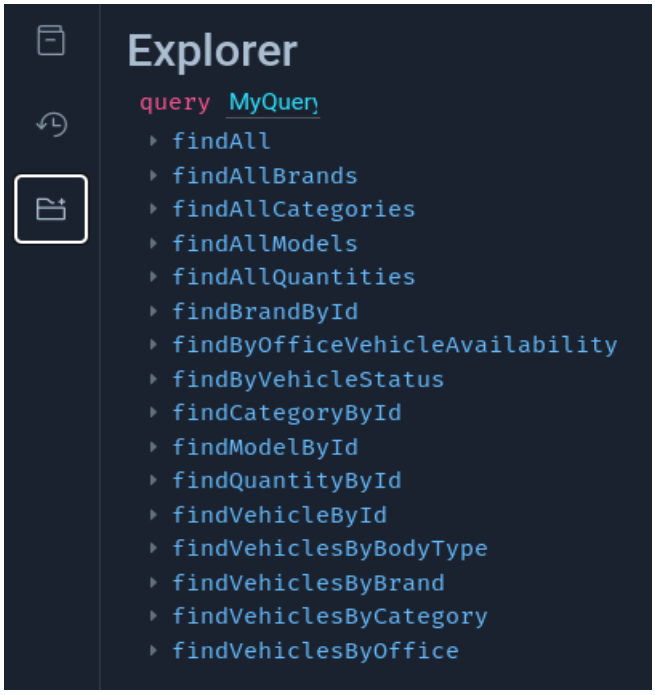


Figure 30: Gestión de flotas queries

Además, para optimizar la gestión, hemos implementado una serie de mutaciones claves que simplifican el manejo del inventario, marcas, categorías y precios y reducen los costes de administración.

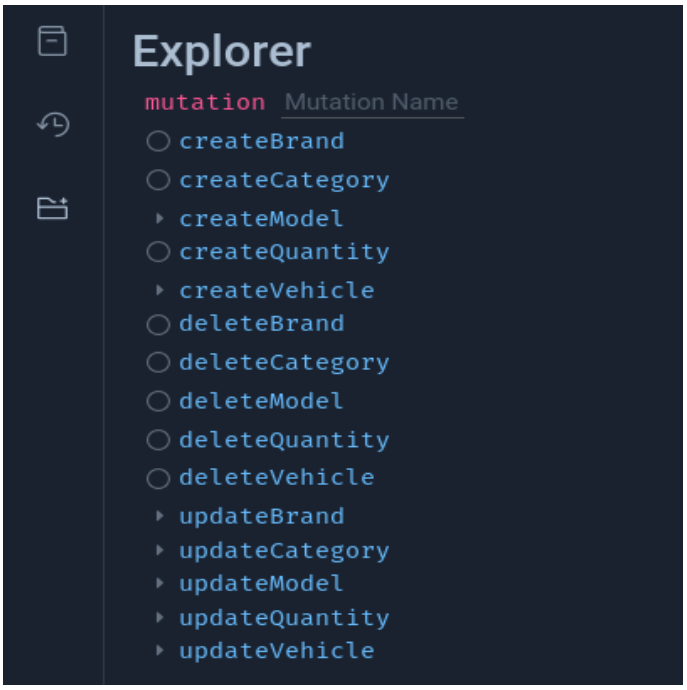


Figure 31: Inventory - service. Mutations

6.3.1 Servicio de reservas: comunicación, bloqueo y notificación

Por otro lado, durante este sprint, hemos progresado significativamente en la implementación del servicio de reservas, historias con ID: 8 y 9, una pieza fundamental encargada de determinar la disponibilidad de vehículos en la flota de RideRush. Para lograrlo, el servicio de reservas se integra con otros componentes clave, como el servicio de oficinas y el servicio de inventario.

El proceso comienza con una comunicación efectiva entre el servicio de reservas y el servicio de oficinas a través de una llamada REST, permitiendo la obtención del identificador de la oficina a partir del nombre proporcionado por el usuario. Posteriormente, se conecta con el servicio de inventario mediante una consulta GraphQL para obtener información detallada sobre los vehículos disponibles en esa oficina.

En el siguiente paso, el servicio de reservas aplica un filtro a los vehículos que ya están reservados durante las fechas que se solapen con las fecha proporcionadas por el usuario, presentando únicamente las opciones disponibles. Destacando su enfoque innovador, el servicio implementa un mecanismo de bloqueo optimista en el inventario, asumiendo de manera eficiente que las colisiones son poco probables.

Cuando un usuario intenta crear una reserva, el servicio de reservas verifica la disponibilidad del vehículo, asegurándose de que no haya sido reservado por otro usuario en un breve lapso de tiempo. En caso de disponibilidad, la reserva se persiste en la base de datos, garantizando un proceso transparente para el usuario. No obstante, si el vehículo ya ha sido reservado, se genera una excepción que informa al usuario sobre la situación.

Además de sus funciones primordiales, el servicio de reservas actúa como generador de eventos, publicando la información de la reserva en el bus de eventos de Kafka. Este flujo de datos es consumido por el servicio de notificación, que desencadena el envío de un correo electrónico al administrador Figure 32, proporcionándole los detalles de la reserva de manera oportuna y eficiente.

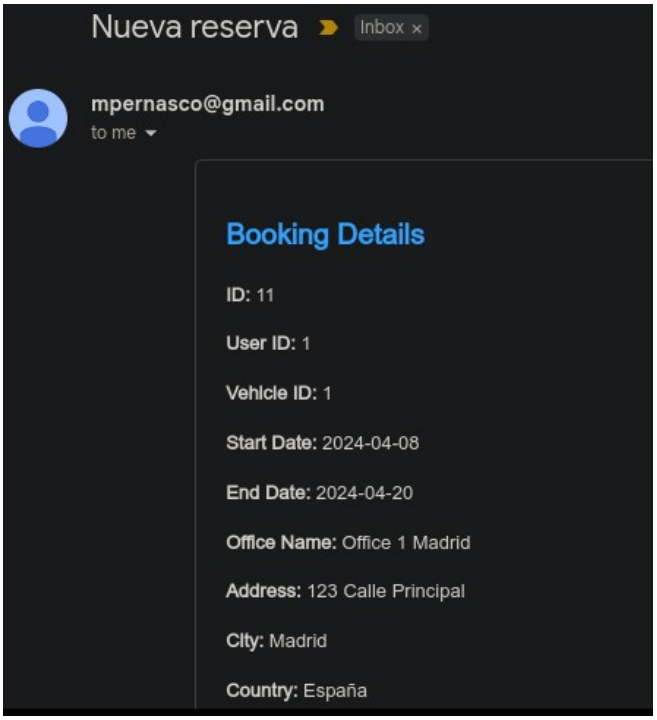


Figure 32: Email de notificación de nueva reserva

En cuanto a las llamadas sincrónicas no bloqueantes, el servicio de reservas optimiza la obtención de la ubicación de otros microservicios mediante Eureka, un sistema de registro y descubrimiento de servicios. Para ello, implementa un mecanismo de reintentos exponenciales para obtener instancias de estos microservicios, mejorando significativamente la tolerancia a fallos y la resiliencia del servicio de reservas frente a posibles errores de conexión con otros componentes del sistema.

Como último paso de este sprint, hemos documentado la API del servicio de reservas empleando Swagger , una herramienta que facilita el diseño, la construcción y la documentación de APIs RESTful. Hemos utilizado el estándar OpenAPI para definir la especificación de la API, que describe los endpoints, los parámetros, las respuestas y los modelos de datos del servicio. Con Swagger, hemos generado una documentación interactiva y fácil de usar, que permite probar y explorar la API desde el navegador. Con

esto, hemos completado nuestro producto mínimo viable, que ofrece las funcionalidades básicas para la gestión de reservas de vehículos turísticos

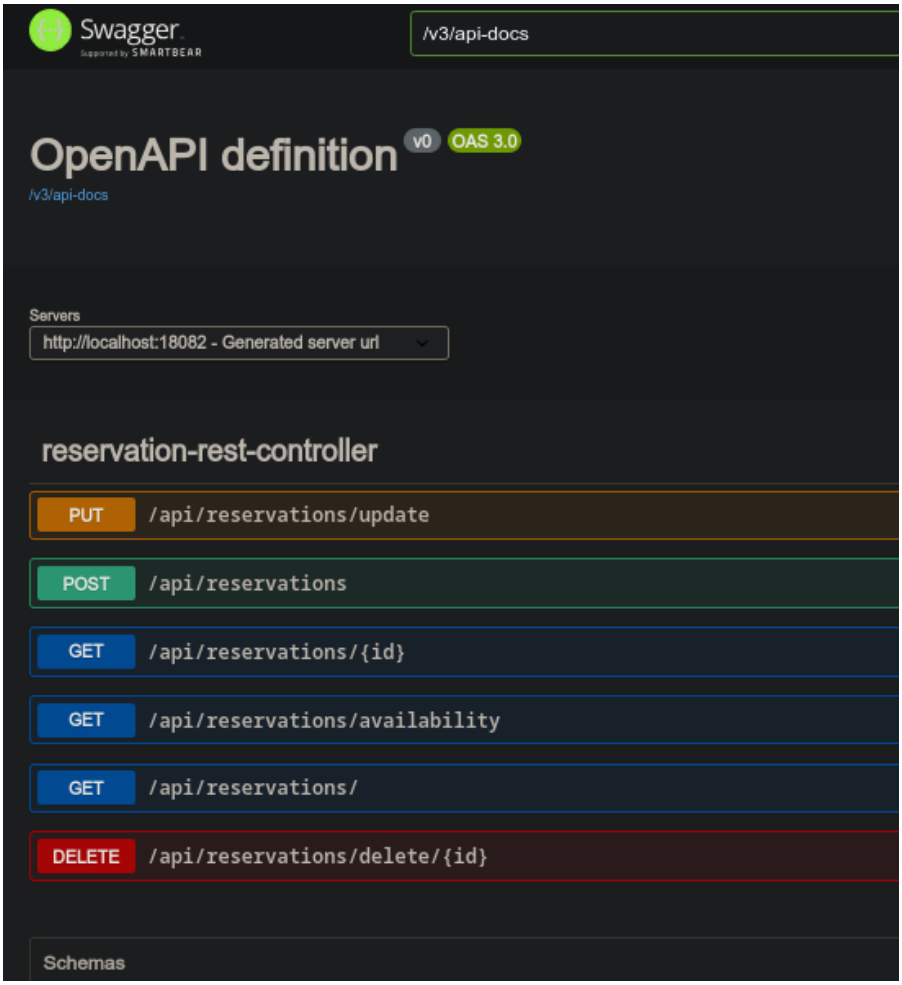


Figure 33: Reservation Service. Endpoints

6.4 Sprint 3: Culminando la implementación del producto mínimo viable

En este sprint, además de implementar las historias de usuario ID 12, 13, 14 y 15, hemos desarrollado tres microservicios adicionales:

- El microservicio de notificación se encarga de enviar correos electrónicos a los usuarios cuando realizan una reserva, confirmando los datos y agradeciendo su preferencia.
- El microservicio de oficina se encarga de gestionar las oficinas de recogida y entrega de los vehículos.
- El microservicio de API gateway se encarga de actuar como un punto de entrada único para todas las peticiones que llegan al sistema, enrutando cada una al microservicio correspondiente y aplicando políticas de seguridad y autenticación.

Además, para facilitar el despliegue y la escalabilidad de los microservicios, hemos utilizado Docker como plataforma de contenedores, que nos permite empaquetar y aislar las aplicaciones con sus dependencias. Hemos creado una pipeline en GitLab que se encarga de crear los contenedores y subirlos a Docker Hub, el repositorio oficial de imágenes de Docker. De esta forma, podemos desplegar los microservicios en cualquier entorno que soporte Docker, sin necesidad de instalar o configurar nada más.

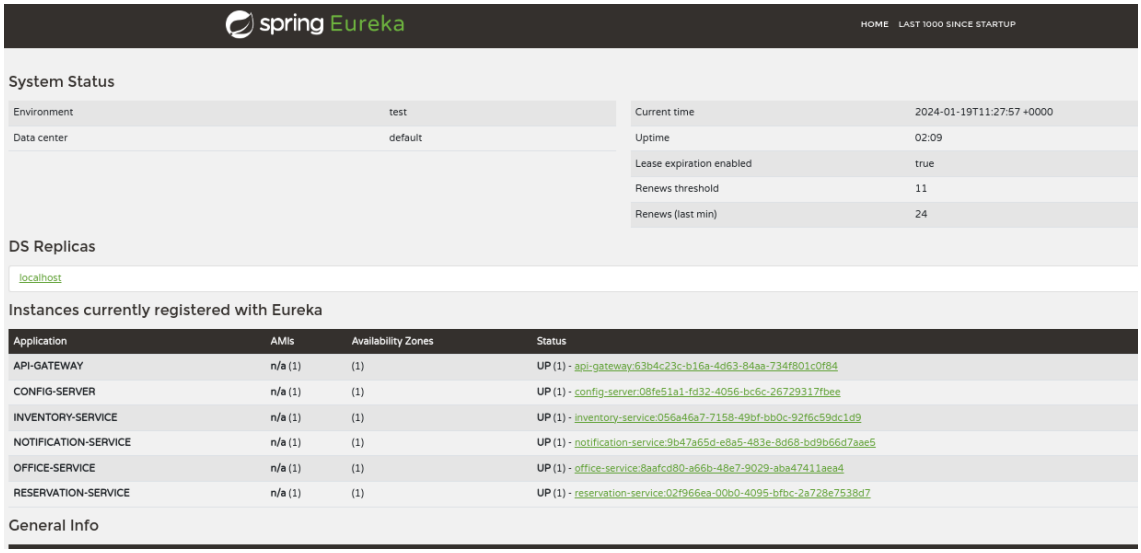


Figure 34: Eureka. Estado y ubicación de los microservicios

Por otro lado, y en este sentido, hemos implementado un servidor de configuración, segunda entrada en Figure 34, que gestiona los ficheros *properties* de forma centralizada en nuestro repositorio de GitLab, facilitando el despliegue y la actualización de los microservicios.

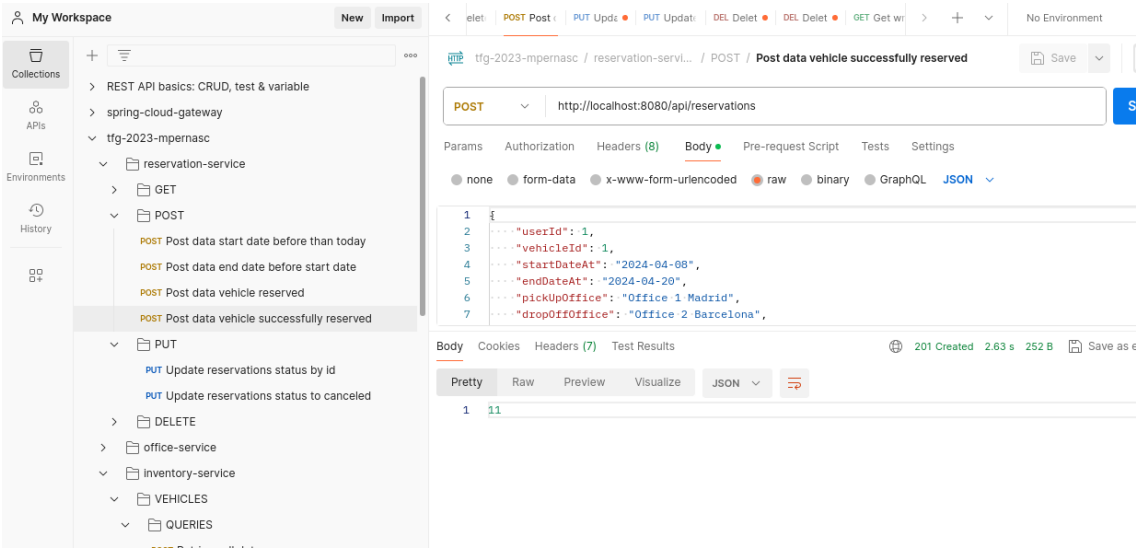
Con este sprint, concluimos la implementación de los microservicios que conforman el sistema de RideRush, ofreciendo un servicio de alquiler y gestión de vehículos turísticos innovador, eficiente y seguro. Utilizado tecnologías modernas y prácticas ágiles para desarrollar un producto mínimo viable que cumple con los requisitos y expectativas de los clientes.

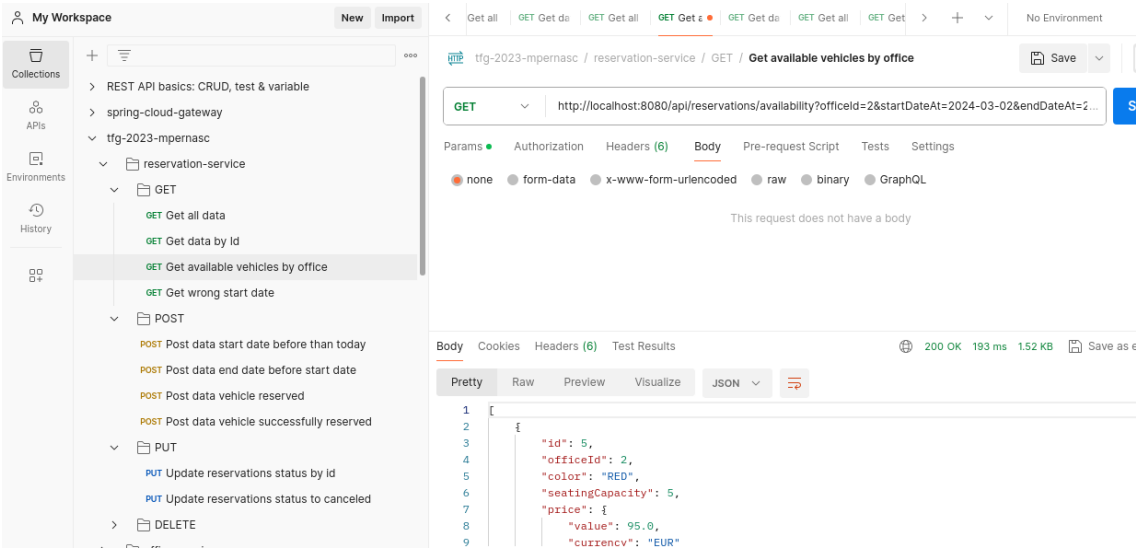
Finalmente, en 11, se presentan los diagramas de clases finales correspondientes a cada uno de los microservicios que componen la arquitectura propuesta. Estos diagramas reflejan el trabajo realizado durante el desarrollo del proyecto y permiten una mejor comprensión de la estructura y funcionalidad de los microservicios. También, se proporciona una tabla con el enlace al repositorio de cada uno de los microservicios.

7 Resultados y evaluación

La culminación del desarrollo e implementación del sistema RideRush nos ha proporcionado una visión clara de los resultados obtenidos y nos permite evaluar el impacto de nuestras decisiones y esfuerzos en la mejora de la plataforma.

Desde el punto de vista técnico, hemos logrado una arquitectura modular y flexible, basada en microservicios, que ha demostrado ser altamente eficiente y escalable. La implementación de tecnologías como Spring y GraphQL ha contribuido a una comunicación más eficaz entre los diversos servicios, mejorando la velocidad y la precisión en la gestión de reservas, la coordinación de flotas y otras funcionalidades clave.





Además, la adopción de Keycloak para la gestión de identidad ha fortalecido la seguridad del sistema, ofreciendo a los usuarios un proceso de registro y autenticación fluido y seguro. La integración de herramientas como Kafka para eventos y la utilización de estrategias de búsqueda han enriquecido las capacidades del sistema, anticipando futuras expansiones y mejoras.

En términos de rendimiento, hemos llevado a cabo pruebas para evaluar la capacidad de respuesta, la escalabilidad y la resiliencia del sistema en diferentes escenarios de carga Figure 39. Comprobamos que es posible ejecutar múltiples instancias de un solo microservicio y si realizamos varias solicitudes seguidas se balancea las solicitudes.

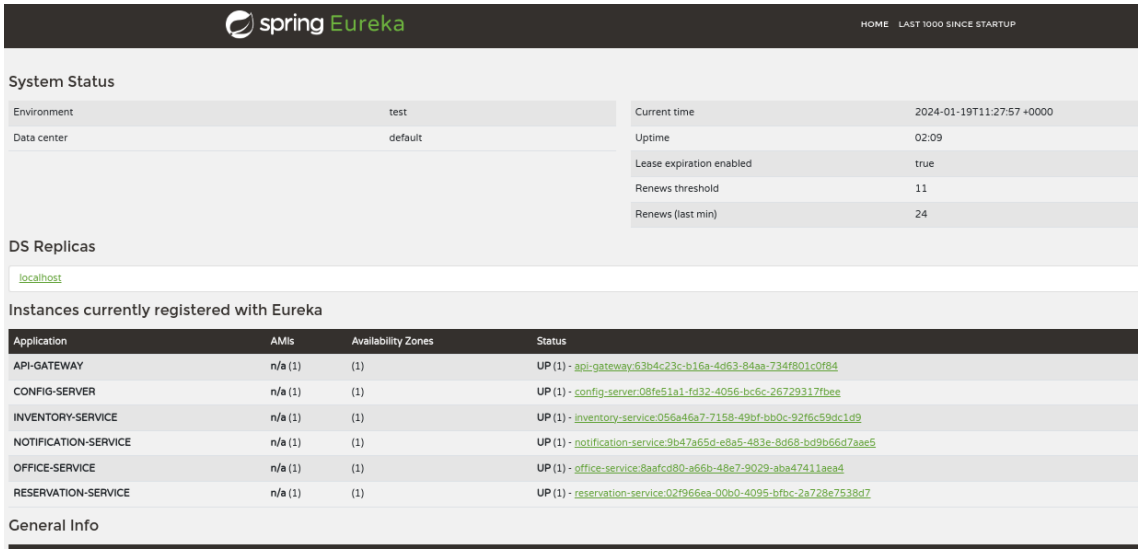


Figure 35: Eureka. Múltiples instancias de *reservation-service*

En la fase de evaluación, hemos recopilado retroalimentación tanto de usuarios internos como de posibles clientes beta, utilizando sus comentarios para realizar ajustes finales y asegurar una alineación aún más precisa con las expectativas del mercado. Con estos resultados en mano, estamos listos para el lanzamiento oficial de RideRush como una plataforma integral de alquiler y gestión de flotas turísticas.

Finalmente, en 11 se explican los pasos para ejecutar el proyecto. Junto con la entrega, se adjunta una colección de postman con las pruebas realizadas, que se puede importar fácilmente. Además, cada microservicio tiene su propia documentación de la API con Swagger, que facilita su consulta.

8 Conclusiones

Este proyecto ha supuesto un gran desafío personal, ya que se he implementado una plataforma de alquiler de vehículos turísticos desde cero, desempeñando varios roles y usando tecnologías y herramientas modernas. Este trabajo me ha dado la oportunidad de ejercer como analista, arquitecto, desarrollador y gestor de proyecto, a lo largo de todo

el proceso, lo que ha sido una experiencia muy valiosa a nivel académico y personal. Para realizarlo con éxito, he utilizado los conocimientos que había adquirido durante mis estudios, como:

- Análisis y diseño con patrones, que me ha ayudado a aplicar las mejores prácticas y soluciones en cada situación.
- Gestión de proyectos, que me ha permitido organizar el trabajo en sprints y adoptar una metodología ágil.
- Ingeniería de requisitos e Ingeniería del software, que me han facilitado la definición y el cumplimiento de los objetivos y las especificaciones del sistema.
- Diseño de bases de datos, que me ha servido para modelar e implementar la estructura de los datos, las relaciones entre las entidades y las restricciones de integridad, usando sistemas relacionales como PostgreSQL
- Proyecto de desarrollo del software, que me ha permitido integrar todos los componentes del sistema, realizar pruebas y validaciones de su funcionamiento.

El resultado de este proyecto es una plataforma de alquiler de vehículos turísticos llamada RideRush, que ofrece un servicio más eficiente, seguro y adaptable a las necesidades del mercado. Para lograrlo, he utilizado una arquitectura basada en microservicios, apoyada por herramientas como Spring y GraphQL, que proporcionan agilidad y escalabilidad al sistema. Además, he implementado Keycloak, una solución que garantiza la seguridad y la identidad de los usuarios, mediante un sistema de registro y autenticación robusto.

Este proyecto no solo representa la solución a los problemas planteados por RideRush, sino que también abre la puerta a futuros avances y mejoras. El sistema se adapta a las fluctuaciones del mercado y utiliza tecnologías innovadoras. RideRush se posiciona como un referente en el sector del turismo.

9 Trabajo futuro

A pesar de los logros significativos alcanzados en este proyecto, existen áreas clave que requieren atención adicional para garantizar el desarrollo continuo y la mejora de la plataforma RideRush.

1. Finalización de implementación de la interfaz del cliente web: aunque hemos avanzado significativamente en la implementación del *backend* , el cliente Next.js aún requiere la finalización de algunas funcionalidades. Es necesario centrarse en la mejora de la experiencia del usuario y la implementación de características pendientes para lograr una plataforma completamente integral.

2. Mejora en la monitorización: la monitorización actual se limita al punto de visibilidad con Eureka. Para obtener una visión más completa del rendimiento del sistema, se recomienda la implementación de herramientas como Prometheus y Grafana. Estas soluciones proporcionarán métricas detalladas y permitirán una identificación proactiva de posibles problemas.

3. Fortalecimiento de pruebas: La suite de pruebas actual es limitada, tanto en términos de pruebas unitarias como de integración. Se sugiere un enfoque más robusto en las pruebas automatizadas para garantizar la estabilidad y la calidad del código. La introducción de pruebas adicionales contribuirá a la identificación temprana de posibles errores y a la mejora general de la confiabilidad del sistema.

4. Implementación de seguridad adicional: aunque Keycloak refuerza la seguridad del sistema, es crucial continuar evaluando y mejorando las medidas de seguridad. La implementación de prácticas como la gestión de tokens de manera más granular y la revisión continua de posibles vulnerabilidades contribuirá a mantener un entorno seguro para los usuarios.

5. El despliegue es otro aspecto clave, ya que en un entorno real se requieren certificados, kubernetes y una pipeline automatizada para desplegar la plataforma en la nube. Aunque, lo intentamos agotamos el crédito disponible en Azure rápidamente.

Estos puntos representan oportunidades clave para la evolución y el perfeccionamiento continuo de la plataforma RideRush. Abordar estos aspectos en el trabajo futuro garantizará que la plataforma no solo cumpla con los estándares actuales, sino que también esté preparada para enfrentar los desafíos emergentes.

10 Bibliografía

[1] "Microservices," martinfowler.com. Accessed: Jan. 13, 2024. [Online]. Available: https://martinfowler.com/articles/microservices.html

[2] martinekuan, "Web API design best practices - Azure Architecture Center." Accessed: Jan. 15, 2024. [Online]. Available: https://learn.microsoft.com/en-us/azure/architecture/best-practices/api-design

[3] D. G. Reinertsen, *Principles of Product Development Flow: Second Generation Lean Product Development*. Celeritas, 2009.

[4] "Figma," Figma. Accessed: Nov. 14, 2023. [Online]. Available: https://www.figma.com

[5] "Figma (@lokman) | Figma Community," Figma. Accessed: Nov. 14, 2023. [Online]. Available: https://www.figma.com/@lokman?fuid=1299661389278114001

[6] "SOLID," *Wikipedia, la enciclopedia libre*. Mar. 04, 2023. Accessed: Nov. 10, 2023. [Online]. Available: https://es.wikipedia.org/w/index.php?title=SOLID&oldid=149676778

[7] "The C4 model for visualising software architecture." Accessed: Nov. 11, 2023. [Online]. Available: https://c4model.com/

[8] "IntelliJ IDEA – the Leading Java and Kotlin IDE," JetBrains. Accessed: Oct. 13, 2023. [Online]. Available: https://www.jetbrains.com/idea/

[9] "The DevSecOps Platform." Accessed: Oct. 13, 2023. [Online]. Available: https://about.gitlab.com/

[10] "Maven – Welcome to Apache Maven." Accessed: Oct. 13, 2023. [Online]. Available: https://maven.apache.org/

[11] "Home," Home. Accessed: Oct. 13, 2023. [Online]. Available: https://spring.io/

[12] "Podman." Accessed: Oct. 13, 2023. [Online]. Available: https://podman.io/

[13] "Apache Kafka," Apache Kafka. Accessed: Oct. 13, 2023. [Online]. Available: https://kafka.apache.org/

[14] "Apache ZooKeeper." Accessed: Oct. 13, 2023. [Online]. Available: https://zookeeper.apache.org/

[15] "Keycloak." Accessed: Oct. 13, 2023. [Online]. Available: https://www.keycloak.org/

[16] "Next.js by Vercel - The React Framework." Accessed: Jan. 19, 2024. [Online]. Available: https://nextjs.org/

[17] "GraphiQL." Accessed: Jan. 19, 2024. [Online]. Available: http://localhost:18081/graphiql?path=/api/inventory

[18] "Install Docker Engine," Docker Documentation. Accessed: Jan. 19, 2024. [Online]. Available: https://docs.docker.com/engine/install/

[19] "Podman Installation | Podman." Accessed: Jan. 19, 2024. [Online]. Available: https://podman.io/docs/installation

[20] "Node.js." Accessed: Jan. 19, 2024. [Online]. Available: https://nodejs.org/en

[21] “Arquitectura de Microservicios | Soluciones de Microservicios,” Chakray. Accessed: Jan. 14, 2024. [Online]. Available: https://www.chakray.com/es/experiencia/microservicios/
[22] “Empresas de alquiler de coches más valiosas del mundo en 2022,” Statista. Accessed: Jan. 13, 2024. [Online]. Available: https://es.statista.com/estadisticas/1024723/ranking-mundial-de-las-empresas-de-alquiler-de-coches-por-valor-de-marca/
[23] N. R. Calle, “Orquestación vs Coreografía en microservicios,” Refactorizando. Accessed: Jan. 13, 2024. [Online]. Available: https://refactorizando.com/orquestacion-vs-coreografia-microservicios/

11 Anexos

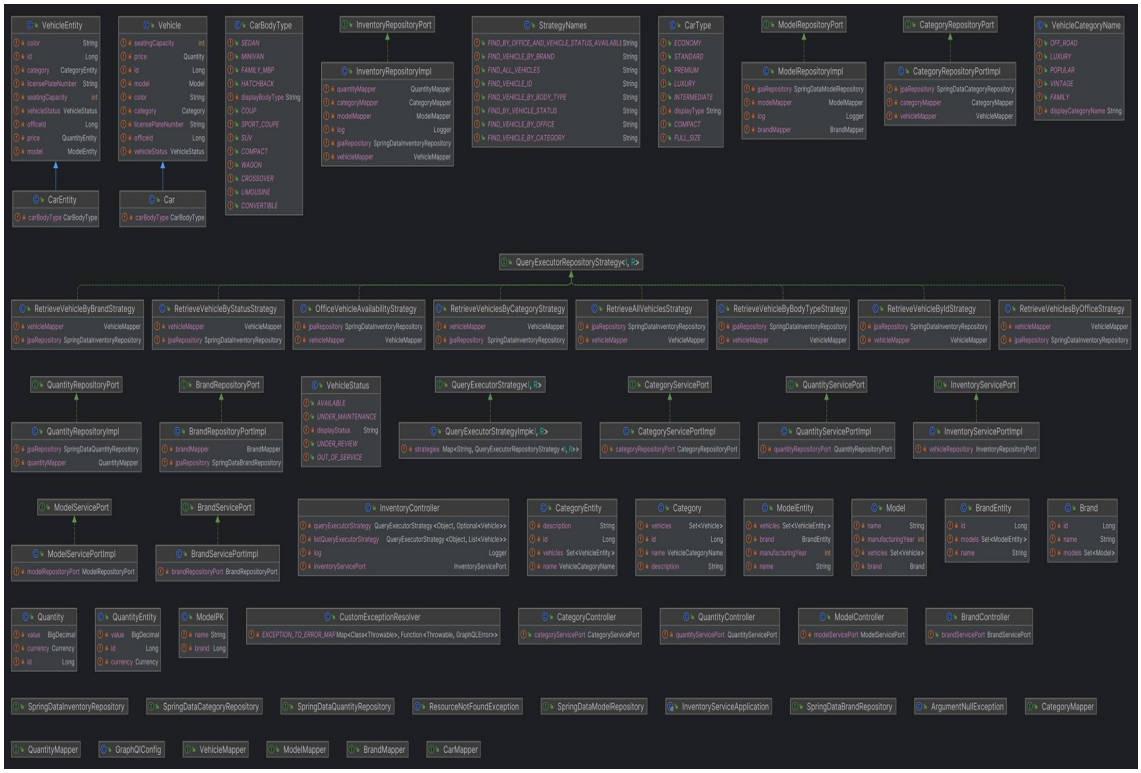


Figure 36: A.1. Inventory - services. Diagrama de clases

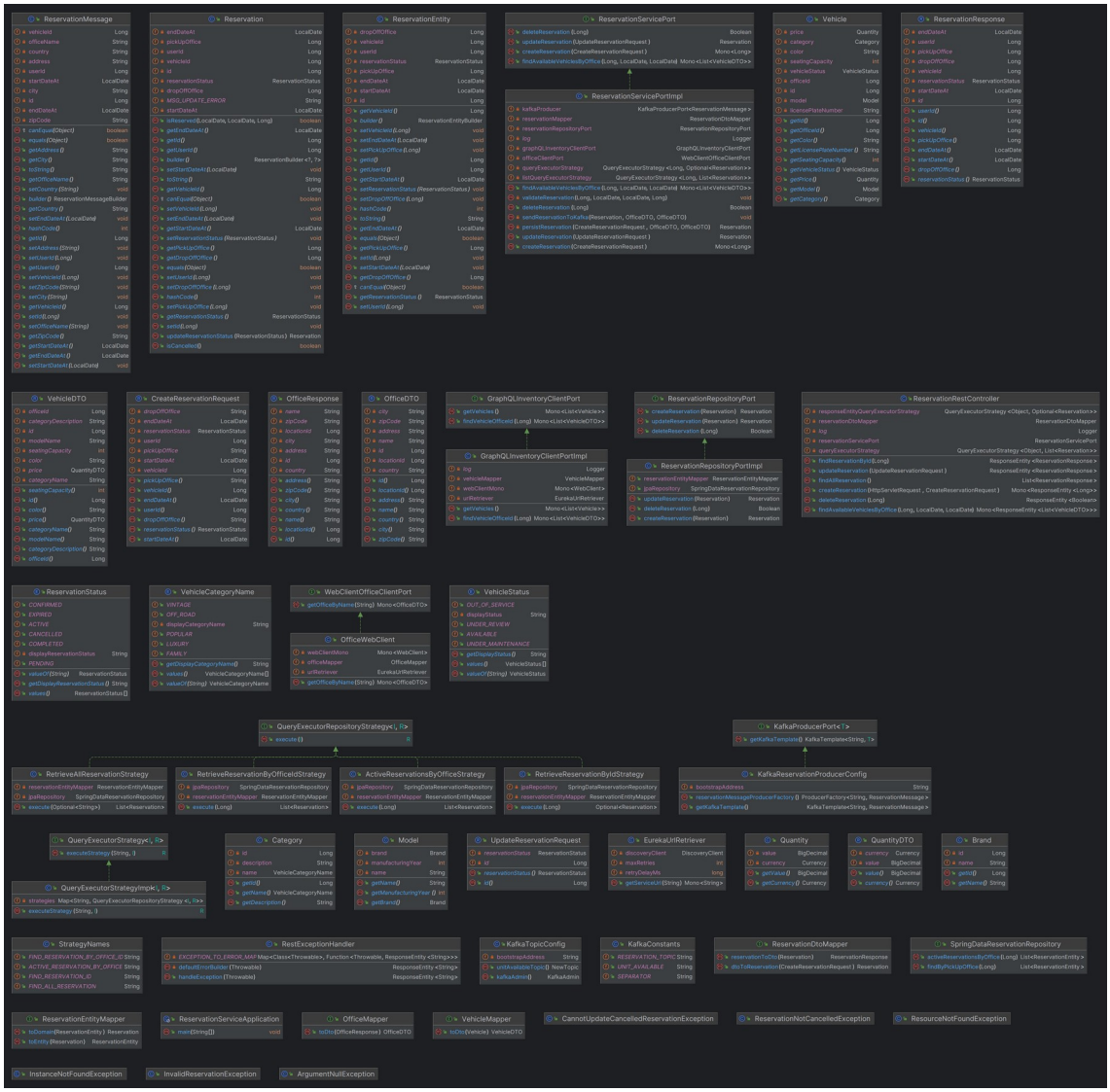


Figure 37: A.2. Reservation - service. Diagrama de clases

Figure 38: A.2. Office - service. Diagrama de clases

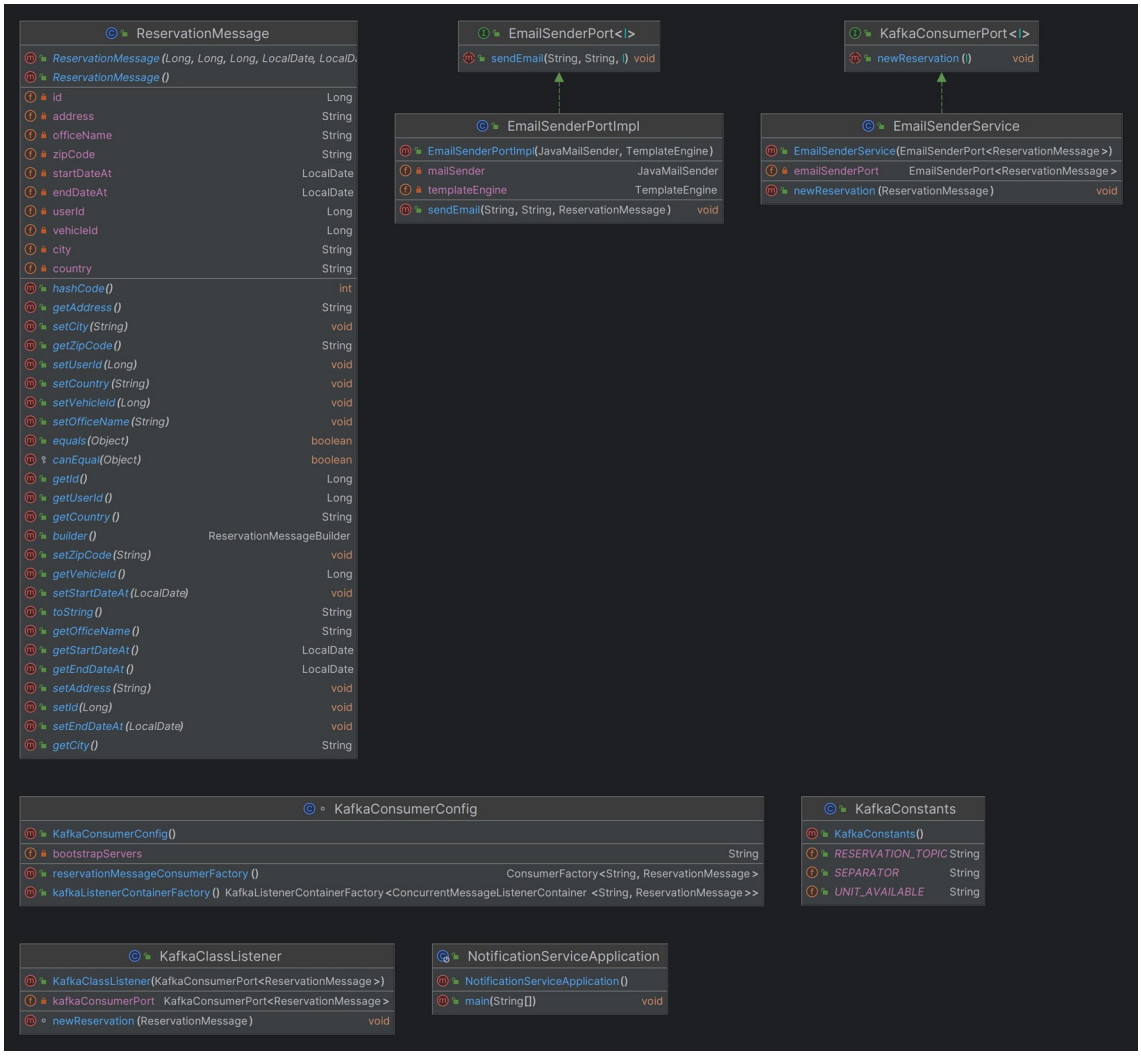


Figure 39: Notification - Service. Diagrama de clases

Table 16: Microservicios. Port mapper y repositorio

Servicio	Puerto externo	Puerto interno	Repositorio
Zookeeper	22181	2181	
kafka	29092	9092	
	19092	19992	
Inventorydb	54320	5432	
reservationdb	54321	5432	
officedb	54322	5432	
postgres	54322	5432	
keycloak	8081	8080	User: admin password: admin User: mpernasco@gmail.com password: admin
discovrey-service	8761	8761	https://gitlab.com/tfg-2023/discovery-service.git
config-server	8888	8888	https://gitlab.com/tfg-2023/config-server.git
api-gateway	8080	8080	https://gitlab.com/tfg-2023/api-gateway.git
inventory-service	18081	18081	https://gitlab.com/tfg-2023/inventory-service.git
reservation-service	18082	18982	https://gitlab.com/tfg-2023/reservation-service.git
notification-service	18083	18083	https://gitlab.com/tfg-2023/notification-service.git
office-service	18084	18084	https://gitlab.com/tfg-2023/office-service.git

Para la puesta en marcha, del proyecto, es recomendable disponer de docker o podman pude encontrar, toda la información en [18] o [19]. A continuación, debe ejecutar docker-compose up -d o podman-compose up -d, según el gestor de contenedores instalado.

Esto creará y ejecutará los servicios definidos en el archivo docker-compose.yaml, adjunto a la entrega, las bases de datos, microservicios, etc. Para realizar las pruebas a la aplicación, puede visitar los endpoint definidos para documentar el proyecto. O ejecutar solicitude desde postman a directamente al API-Gateway http://localhost:8080/path. Además, para facilitar las pruebas, se adjunta una colección de postman que puede importar con todas la solicitudes, *queries* y *mutation*.

Por otro lado, no fue posible terminar la implementación de cliente para el front. Así pues, será el único de los servicios que sí debe iniciarse directamente desde el IDE o el terminal ejecutando npm run dev. Es imprescindible tener instalado node [20]