Fleet manager: sistema gestor de flotas de vehículos

Javier Rubio Canca
Grado en Ingeniería Informática

Antoni Oller Arcas

19 de enero de 2023

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Fleet manager*: sistema gestor de flotas de vehículos
Nombre del autor:	Javier Rubio Canca
Nombre del consultor:	Antoni Oller Arcas
Fecha de entrega (mm/aaaa):	01/2023
Área del Trabajo Final:	Java EE
Titulación:	*Grado en Ingeniería Informática*

Resumen del Trabajo (máximo 250 palabras):

El objetivo de este trabajo final de grado es planificar y ejecutar un proyecto, basado en la gestión de flotas de vehículos, de desarrollo de software completo, desde la fase de planificación y análisis, hasta la implementación y puesta en marcha. Además, se busca cumplir estos objetivos mediante el uso de herramientas y tecnologías actuales y, sobre todo, metodologías estándar y/o ampliamente aceptadas.

Durante la planificación y análisis se aplica las competencias adquiridas a lo largo del grado en asignaturas como gestión de proyectos, ingeniería de requisitos, ingeniería del software, entre otras. En la fase de implementación se aplican conocimientos aprendidos de asignaturas como: programación orientada a objetos, análisis y diseño con patrones, etc.

La idea del proyecto no es un caso trivial o de poca consideración práctica, sino que, según mi experiencia profesional, es un producto aplicable a muchas empresas reales. La mayoría de las empresas desconocen la posibilidad de digitalizar procesos que, en la actualidad, por desconocimiento o incertidumbre, no informatizan.

Aunque el objetivo final del proyecto es la entrega de un MVP (mínimo producto viable) que incluiría la aplicación o aplicaciones de lado servidor y su despliegue en entorno *cloud*, y una aplicación frontal (navegador o dispositivo móvil), la realidad es que, en el plazo del que se dispone, la primera entrega únicamente dispondrá de las aplicaciones *back-end* y su puesta en marcha en servicios en la nube.

Abstract (in English, 250 words or less):

The purpose of this project is to make use of the knowledge acquired during the university degree by creating a project from its earliest phase (analysis) to it's final phase, the implementation and testing. Also, the objective is to generate a product as an outcome.

The deliverable will serve, and it is ordered by, the company Sirmetal S.L, an industrial equipment distributor sited in Barcelona (Spain) province, which has the necessity of a system to help them manage their vehicle park by providing their location, status and many other features.

Despite the multiple challenges that managing and executing a project involve, I wanted to self-impose another challenge: deliver the product in a cloud-based system. This matter required a lot of effor as I've never use DevOps tools other than for monitoring.

Also, for the execution of the project, one of the most important objectives is to follow latest industry standards, not only related to the tools and software, but also procedures and conventions.

Palabras clave (entre 4 y 8):

Gestión, flotas, geolocalización, J2EE, *spring framework*

Índice

Lista de figuras

1. Introducción

1.1 Contexto y justificación del Trabajo

Durante los últimos años se está experimentando, en el tejido empresarial, una gran necesidad de adoptar nuevos procesos, o bien, digitalizar procesos que anteriormente no lo estaban. Transformar procesos que se han mantenido en el tiempo como procesos manuales y/o no informatizados, ahora requieren de su automatización. Aunque, en la mayoría de los casos, no es crucial para la supervivencia de la empresa, sí que limita (o simplemente no aumenta) su competitividad y valor.

Gracias a mi experiencia como técnico de proyectos industriales, soy consciente de que, actualmente, aún existen muchos procesos, en multitud de empresas diferentes, que siguen gestionándose de manera manual. Algunos ejemplos son: ausencia de sistemas informatizados para el control horario, recepción y registro de entrada de mercancías, etc.

Este proyecto busca automatizar un proceso manual que se lleva a cabo en muchas empresas, especialmente PYMEs, que disponen de flota de vehículos para la entrega de materiales a sus clientes: gestionar y geolocalizar su flota de vehículos de reparto. Concretamente, es la empresa Sirmetal S.L, ubicada en la provincia de Barcelona, la que nos encarga el proceso de automatización.

La empresa tiene la necesidad de dar una respuesta rápida a los clientes que acuden a ellos para interesarse sobre el tiempo estimado de entrega del material que esperan. Además, requieren de un sistema que registre, con una frecuencia relativamente alta, el estado de carga y kilometraje de sus vehículos; así como su estado mecánico y necesidad de mantenimientos.

La empresa Embedded Sys S.L, suministradores de sistemas integrados, será la designada por Sirmetal S.L. para dotar a los vehículos de sistemas embebidos que serán los encargados de transmitir los detalles de los coches al sistema que se creará como resultado de este trabajo.

1.2 Objetivos del Trabajo

Los objetivos de este trabajo no se reservan exclusivamente a la creación de un producto, sino que se estudia con especial cuidado la manera en la que se produce y los procedimientos que se siguen, para que el producto pueda evolucionar e implementar nuevas funcionalidades de manera sencilla y estructurada.

Como objetivos estrictamente vinculados al producto, se listan los siguientes:

- Un sistema que registre la información enviada (ubicación y estado del vehículo) por los controladores integrados en cada uno de los múltiples vehículos de la empresa.

- Una aplicación de lado servidor que sirva información a las diferentes aplicaciones frontales que se implementarán en futuras iteraciones.
- Aplicación frontal para navegadores.

También se fijan los siguientes objetivos indirectos:

- El producto final debe estar desarrollado siguiendo procedimientos y estándares de la industria.
- El servicio debe ser entregado utilizando servicios *cloud*. Personalmente, quiero aprovechar al máximo este trabajo final de grado para aprender todo lo posible sobre servicios en la nube, herramientas de creación de *containers* y *kubernetes*.
- Familiarizarme con la consola de *Windows PowerShell*.

1.3 Enfoque y método seguido

Aunque existen diferentes soluciones disponibles en el mercado, muchas de ellas están diseñadas para grandes corporaciones con flotas para mercancías pesadas o maquinaria de construcción, y además de licencia propietaria.

La solución que más podría asimilarse a los requerimientos del proyecto es *Everlance* (www.everlance.com), pero se desconoce: el mecanismo de registro de ubicaciones, si las funcionalidades de la aplicación se ajustan a los requisitos funcionales de la empresa, el coste que podría constituir y, lo más importante, si disponen de distribuidor en nuestro país.

Una vez determinado que las opciones disponibles en el mercado no se adecuan completamente a los requisitos impuestos por el cliente y que no ofrecen opciones de personalización, se decide desarrollar una solución personalizada. El software a construir deberá cumplir con las funcionalidades demandadas por la empresa y ser capaz de extenderse con otras nuevas que puedan surgir en un futuro.

1.4 Planificación del Trabajo

La planificación temporal de la ejecución del proyecto vendrá condicionada por el plan de trabajo de la asignatura, y consta de las siguientes *deadlines*:

Actividad	Plazo de entrega
PEC1 – Plan de trabajo	11/10/22
PEC2 – Análisis y diseño	11/11/22
PEC3 - Implementación	02/01/23
Memoria y presentación	19/01/23
Tribunal	31/01/23

Teniendo en cuenta los plazos indicados en la tabla 1, se genera el diagrama de *Gantt* donde se especifican todas las tareas a llevar a cabo para cada actividad. Debido a las dimensiones del gráfico y para garantizar una correcta legibilidad, se incluye el diagrama en el anexo 1 – diagrama de *Gantt*.

1.5 Breve sumario de productos obtenidos

Aunque el objetivo era crear un producto final, desplegado en entorno *cloud*, que incluyera las aplicaciones de lado servidor y un frontal para navegadores web, el tiempo disponible para el desarrollo del producto únicamente me ha permitido crear las aplicaciones de lado servidor y sus respectivos despliegues en la nube.

No obstante, las convenciones y herramientas concretadas y aplicadas durante el diseño y la implementación, se ha obtenido un producto muy propicio a iteraciones incrementales de producto.

1.6 Breve descripción de los otros capítulos de la memoria

Esta memoria está compuesta por los siguientes capítulos:

- Análisis: se estudian y refinan los requisitos de cada uno de los *stakeholders*.
- Diseño: se diseña una solución que cumpla con los requisitos extraídos durante la fase de diseño.
- Implementación: durante este capítulo, se detalla el procedimiento de implementación, pruebas y despliegue de los diferentes componentes.
- Puntos de mejora y/o pendientes: se detallan aspectos a mejorar, así como el trabajo pendiente para obtener un producto final.
- Guía para el uso de la aplicación: se detalla el procedimiento para probar la aplicación.
- Valoración económica: se concretan y detallan los costes incurridos durante la ejecución del proyecto.

2. Análisis

2.1 Introducción

En esta sección, en la que se aborda el análisis de la aplicación, formado por un estudio que genera modelo de casos de uso y actores, fichas de casos de uso y modelos de interfaz.

2.2 Toma de requisitos

Para especificar los requerimientos de la aplicación que se quiere construir, expondré los testimonios recogidos de las partes interesadas (*stakeholders*) que estarán involucrados de la empresa Sirmetal S.L. Puesto que la necesidad surge de varios departamentos de la empresa, es posible que las necesidades sean muy diversas y subjetivas. No obstante, esto puede influir positivamente debido al nivel de detalle y variadas perspectivas que aportan.

Stakeholder 1

Nombre:	Sara Ramírez
Cargo:	Directora General
Testimonio:	En Sirmetal S.L., consolidados como principal proveedor de material metalúrgico en la provincia de Barcelona y contando con un crecimiento neto interanual del 30% durante los últimos 5 años, debemos ser fieles a nuestros valores de seguridad y compromiso ecológico. Minimizar el impacto en el medio ambiente es uno de los principales objetivos de nuestra empresa, que año a año incentiva que estudiemos de manera interdepartamental qué mejoras se pueden implementar para hacer la empresa más eficiente. La última acción que más impacto ha generado, tanto a la empresa como a su compromiso con el medio ambiente, se llevó a cabo durante el año 2020, y fue la renovación de nuestra flota de vehículos de reparto. El 100% de la flota, que contaba con motorizaciones de combustión, se han sustituido por vehículos completamente eléctricos. Justo a la renovación de la flota, se instalaron en nuestras tres delegaciones, repartidas por la provincia, cargadores eléctricos juntamente a placas fotovoltaicas, repartidas por todas las cubiertas de los edificios propiedad de la empresa. Siguiendo con nuestro nivel de compromiso con la innovación y buscando reforzar la imagen de empresa tecnológicamente moderna, nuestra siguiente gran inversión consistirá en un sistema informático que permita la administración y gestión de la flota de vehículos. El sistema deberá servir de soporte a varios departamentos de la organización. Para el departamento de logística y almacén, ofrecerá ubicación y condiciones de los vehículos. La unidad de finanzas podrá determinar costes según los registros de la aplicación.

Stakeholder 2

Nombre:	Carles Surrà
Cargo:	Jefe de logística y almacén
Testimonio:	Nuestro personal de reparto está formado por 12 profesionales especializados en logística. Cada mañana cargan las furgonetas y empiezan la jornada de reparto que, habitualmente, consta de entre 10 y 15 paradas en dependencias del cliente. Las principales problemáticas que requerimos solventar con esta nueva aplicación son las siguientes: • No sabemos dónde se ubican nuestros/as repartidores/as. Si un cliente se comunica con nosotros para consultar a qué hora se hará la entrega de su esperado material, no tenemos esa información. Para poder darles respuesta, tenemos que llamar al repartidor/a asignado/a que, probablemente, se encuentre en medio de un trayecto. Esta solución no es eficiente, porque tanto yo como el/la repartidor/a debemos dejar de lado nuestras tareas actuales para tratar de comunicarnos. Tampoco es seguro, puesto que el/la conductor/a puede estar conduciendo en ese momento. La aplicación debería facilitarnos la ubicación, más o menos reciente, de los/as conductores/as. • Desconocemos quién es el/la repartidor/a que conduce ese día cada furgoneta. Además de la ubicación, nos gustaría que nos facilitara información de quién está conduciendo el vehículo. • Queremos minimizar las tareas manuales. Por ejemplo: revisar el porcentaje de batería del coche, su kilometraje, saber si hay coches en estado averiado y cuando fue la última revisión en el taller.

Stakeholder 3

Nombre:	Ismael Monforte
Cargo:	Responsable IT
Testimonio:	Los recursos informáticos de la empresa son limitados, por lo que no puede considerarse un despliegue de la aplicación *on-premise*. Dicho esto, rogamos que tengan en cuenta la necesidad de un alojamiento completamente en la nube. En cuanto a seguridad, debe considerarse un estándar de seguridad que garantice que el acceso a la información será única y exclusivamente por parte de empleados de la empresa o autorizados por la misma. Con respecto a la información, es indispensable que la aplicación no almacene ningún dato de carácter personal, de lo contrario, deberá estudiarse y certificarse por nuestra empresa externa encargada en la materia, además de cumplir con las exigencias que impongan desde nuestro departamento legal.

Stakeholder 4

Nombre:	Eduardo Díaz
Cargo:	Director financiero
Testimonio:	Con el nuevo sistema esperamos tener información precisa y altamente disponible sobre el uso y desgaste de nuestra flota de vehículos, lo que nos permitirá el cálculo de costes de nuestro servicio de entregas sin depender de nuestro departamento logístico, que además los liberará de carga al no tener que realizar los informes mensuales de gastos de la flota. La información que necesitamos que nos proporcione esta aplicación es la cantidad de kilómetros que hace cada empleado a lo largo de un mes, así como el estado de la flota y la necesidad de revisiones y reparaciones.

Stakeholder 5

Nombre:	Aina Díaz
Cargo:	Encargada de servicio al cliente
Testimonio:	La funcionalidad que esperamos que nos ofrezca la aplicación es conocer la ubicación de cada uno de los repartidores para poder dar respuesta al cliente en el menor tiempo posible. Ahora mismo, para dar respuesta a un cliente, necesitamos contactar con el departamento logístico y consultarles a ellos/as el momento en el que se hará la entrega. Con esta nueva aplicación nos ahorraremos todo ese procedimiento ya que la aplicación nos mostrará la ubicación del profesional que hará la entrega.

2.3 Requisitos

En esta sección se detallan los requisitos extraídos de las diversas entrevistas realizadas a las partes interesadas. Se dividirán en requisitos funcionales y no funcionales:

2.3.1 Requisitos funcionales

Requisito	Descripción
REQ-001	El sistema debe poder recibir y registrar la ubicación de los vehículos con una frecuencia de 30 segundos (valor parametrizable).
REQ-002	La aplicación debe poder recibir y registrar, en tiempo real, la comunicación de avería de los vehículos.
REQ-003	La aplicación recibirá y registrará qué repartidor está asignado a cada uno de los vehículos.
REQ-004	El sistema debe poder recibir y registrar el kilometraje y porcentaje de batería que cada vehículo reportará con una frecuencia de 60 segundos, también parametrizable.
REQ-005	La aplicación registrará la fecha de última revisión de cada uno de los vehículos.
REQ-006	La aplicación ofrecerá información de la cantidad de kilómetros efectuados por cada repartidor, permitiendo el filtrado por periodos temporales.
REQ-007	La aplicación ofrecerá información sobre qué vehículos están activos y cuales

	averiados.
REQ-008	El sistema registrará las actuaciones que deban producirse en cada uno de los vehículos.

2.3.2 Requisitos no funcionales

Requisito	Descripción
REQ-009	Debido a los escasos recursos de infraestructura IT de la empresa, el sistema debe estar hospedado en servicios.
REQ-010	Debe garantizarse que el acceso a la información es únicamente posible para usuarios autorizados por la empresa.
REQ-011	El sistema no debe guardar ningún tipo de información protegida por la LOPD.
REQ-004	El sistema debe poder recibir y registrar el kilometraje y porcentaje de batería que cada vehículo reportará con una frecuencia de 60 segundos, también parametrizable.

2.4 Actores

Según el análisis de los testimonios de las partes interesadas, los actores identificados son los siguientes:

- Usuario: usuario del sistema, el cual interactuará para consultar información sobre los vehículos, conductores y su ubicación.
- Administrador: se encargará de gestionar el acceso, y revocación de este, de los usuarios, vehículos y jefe/a de logística.
- Jefe de logística: además de las funcionalidades propias de usuario, también podrá gestionar la flota de vehículos.
- Vehículo: sistema que se comunicará con la aplicación para registrar diferente información sobre el vehículo en el que está ensamblado.

2.5 Diagrama de casos de uso

Se generan los casos de uso, para cada uno de los actores, extraídos de los testimonios de los *stakeholders*.

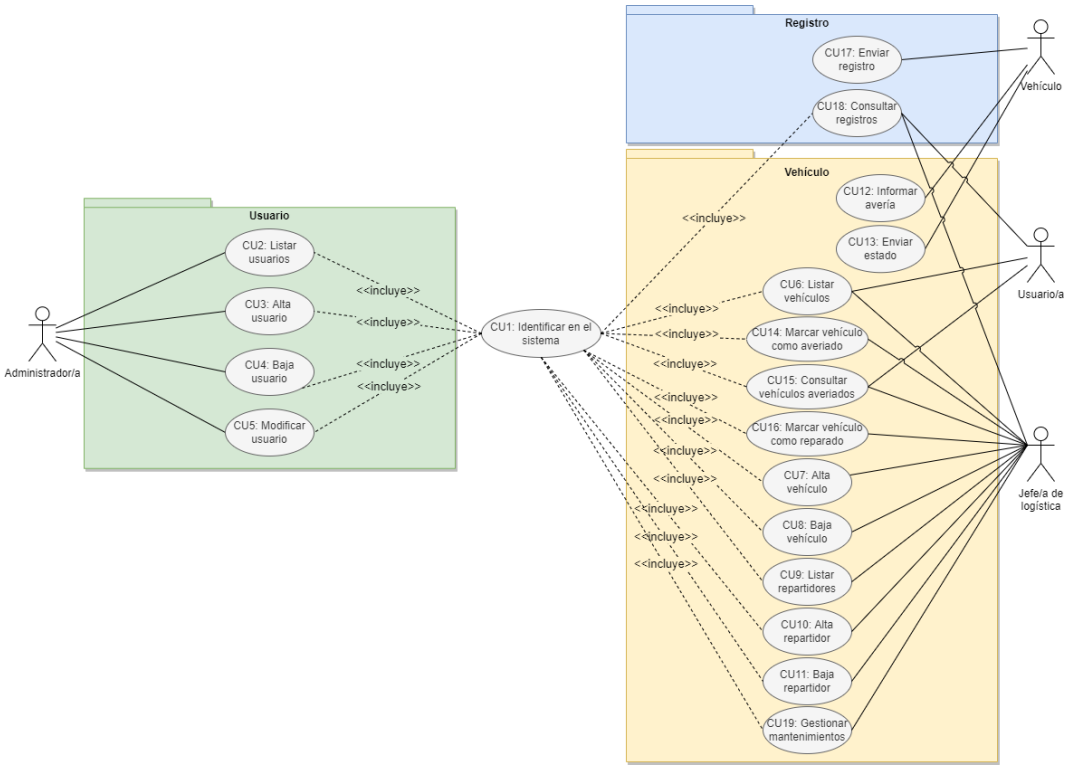


Ilustración 1 - Diagrama de casos de uso

2.6 Fichas de casos de uso

Del anterior diagrama, se desgranan los casos de uso por paquetes, especificando primero el caso de uso CU1 – identificar en el sistema, que actuará como precondición para el resto.

Caso de uso	CU1 – Identificar en el sistema	
Descripción	El usuario de la aplicación debe identificarse en el sistema para acceder a sus funcionalidades.	
Actores principales	Usuario, administrador y jefe de logística.	
Precondición	El usuario no se ha identificado previamente.	
Garantías en caso de éxito	El usuario obtiene acceso a las funcionalidades permitidas para su rol.	
Escenario principal de éxito	Paso	Descripción
	1	La aplicación solicita al usuario su nombre de usuario y contraseña.
	2	El usuario introduce los datos.
	3	El sistema valida los datos y redirige al usuario a la pantalla principal.
Escenario alternativo 1	3.1	El sistema comprueba que los datos introducidos no

		corresponden a ningún usuario. Se informa al usuario y volvemos al paso 2.
Interfaz asociada	IU1 – Interfaz de identificación en el sistema.	

2.6.1 Paquete usuario

A continuación, se detallan los casos de usuario pertenecientes al paquete usuario.

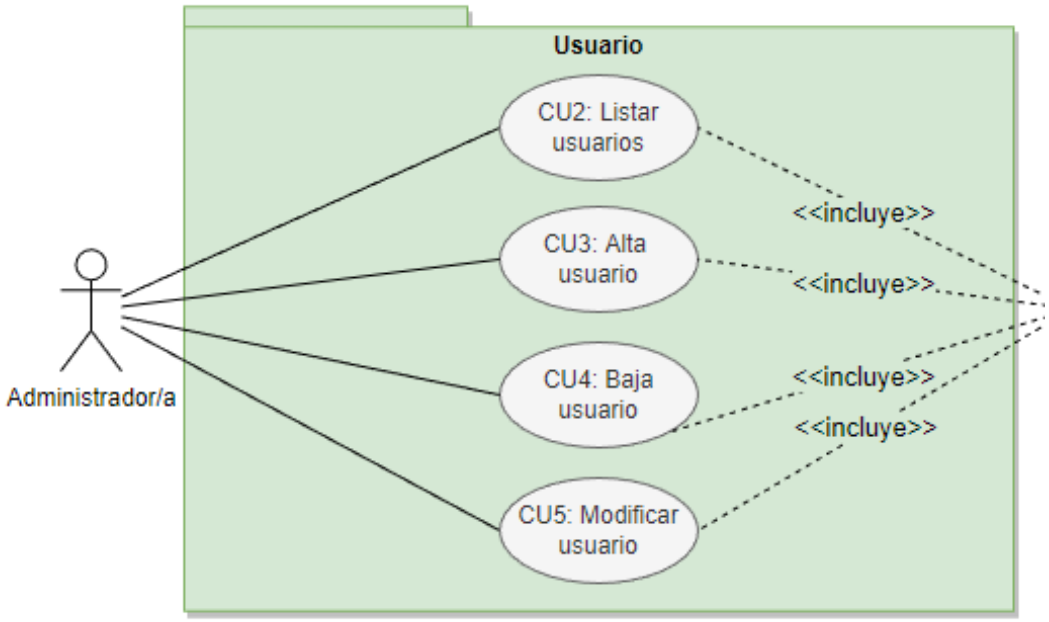


Ilustración 2 - Casos de uso, paquete usuario

Caso de uso	CU2 – Listar usuarios	
Descripción	El usuario administrador debe poder obtener un listado de los usuarios en el sistema.	
Actores principales	Administrador.	
Precondición	CU1 – Identificar en el sistema.	
Garantías en caso de éxito	El usuario administrador ha obtenido un listado de los usuarios dados de alta en el sistema actualmente.	
Escenario principal de éxito	Paso	Descripción
	1	El usuario administrador indica que quiere obtener un listado de usuarios.
	2	El sistema muestra por pantalla una tabla con los usuarios, sus respectivos roles y si se encuentran activos o no. Además, proporciona al usuario filtros que le permitirá acotar el listado por nombre de usuario, rol y activo o no. También, el sistema proporciona al usuario la opción de borrar o modificar el usuario de una fila específica, además de la posibilidad de dar de alta un usuario.
Interfaz asociada	IU2 – Interfaz de listado de usuarios.	

Caso de uso	CU3 – Alta usuario	
Descripción	El usuario administrador debe poder dar de alta nuevos usuarios de tipo usuario, administrador y jefe de almacén.	
Actores principales	Administrador.	
Precondición	CU1 – Identificar en el sistema.	
Garantías en caso de éxito	El usuario administrador ha dado de alta un usuario en el sistema satisfactoriamente.	
Escenario principal de éxito	Paso	Descripción
	1	El usuario administrador indica que quiere dar de alta un nuevo usuario.
	2	El sistema muestra la plantilla para introducir los datos del nuevo usuario.
	3	El usuario administrador introduce el nombre de usuario, contraseña y rol (usuario, administrador o jefe de logística) y envía el formulario.
	4	El sistema recibe los datos, comprueba que son válidos y los guarda.
Escenario alternativo 1	4.1	El sistema comprueba que los datos introducidos no son válidos. Volvemos al paso 2.
Interfaz asociada	IU3 – Interfaz de creación de nuevo usuario.	

Caso de uso	CU4 – Baja usuario	
Descripción	El usuario administrador debe poder dar de baja usuarios de tipo usuario, administrador y jefe de almacén.	
Actores principales	Administrador.	
Precondición	CU1 – Identificar en el sistema CU2 – Listar usuarios	
Garantías en caso de éxito	El usuario administrador ha dado de baja un usuario en el sistema.	
Escenario principal de éxito	Paso	Descripción
	1	El usuario administrador presiona el botón de borrado en la fila correspondiente al usuario que desea borrar.
	2	El sistema solicita confirmación para borrar el usuario seleccionado.
	3	El usuario administrador confirma la baja.
	4	El sistema realiza la baja del usuario.
Interfaz asociada	IU4 – Interfaz de baja usuario.	

Caso de uso	CU5 – Modificar usuario	
Descripción	El usuario administrador debe poder modificar los datos de usuarios de tipo usuario, administrador y jefe de almacén.	
Actores principales	Administrador.	
Precondición	CU1 – Identificar en el sistema CU2 – Listar usuarios	
Garantías en caso de éxito	El usuario administrador ha modificado los datos de un usuario en el sistema satisfactoriamente.	
Escenario principal de	Paso	Descripción

éxito	1	El usuario administrador selecciona qué usuario quiere modificar.
	2	El sistema muestra campos para realizar la modificación (contraseña y/o rol).
	3	El usuario administrador introduce los datos que desea modificar y envía el formulario.
	4	El sistema modifica los datos del usuario correspondientes.
Escenario alternativo 1	4.1	El sistema comprueba que los datos introducidos no son válidos y volvemos al paso 3.
Interfaz asociada	IU5 – Interfaz de modificación de usuario.	

2.6.2 Paquete vehículo

A continuación, se detallan los casos de usuario pertenecientes al paquete vehículo.

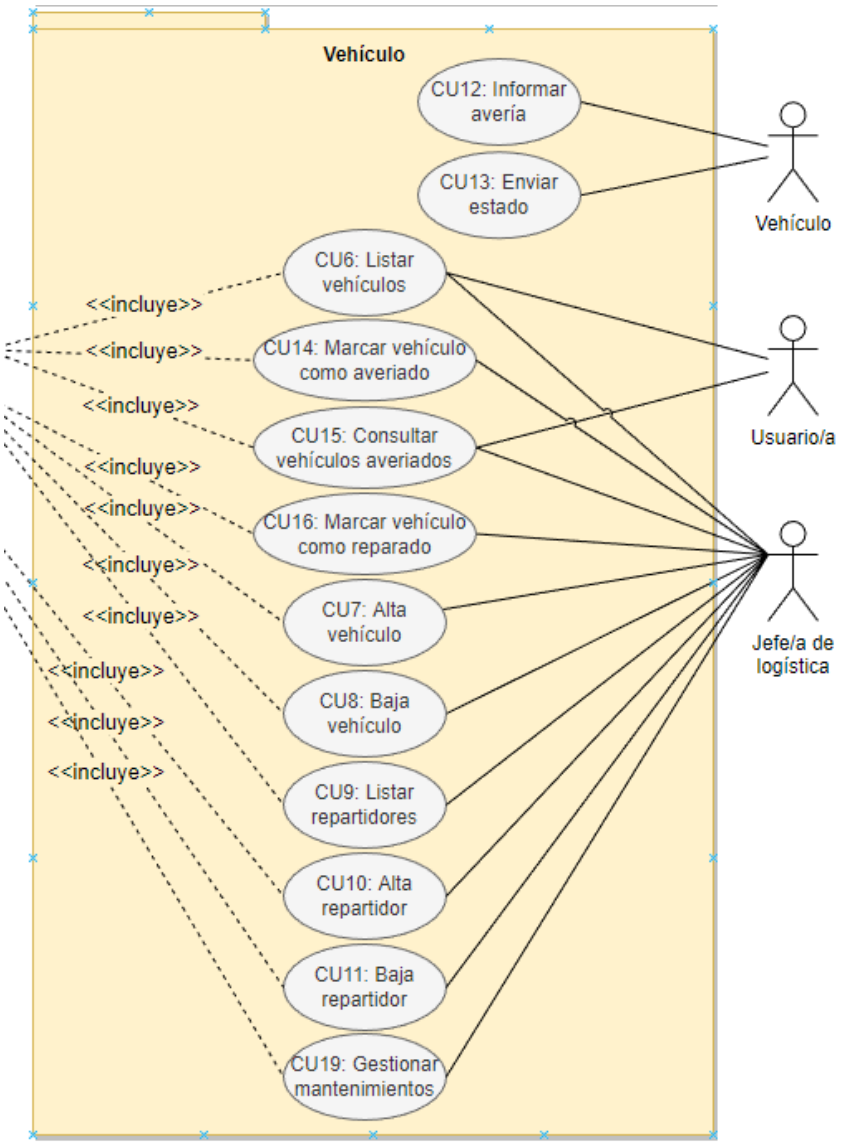


Ilustración 3 – Casos de uso paquete Vehículo

Caso de uso	CU6 – Listar vehículos	
Descripción	El usuario debe poder obtener un listado de los vehículos activos en el sistema.	
Actores principales	Usuario y jefe de logística.	
Precondición	CU1 – Identificar en el sistema.	
Garantías en caso de éxito	El usuario ha obtenido un listado de los vehículos registrados en el sistema.	
Escenario principal de éxito	Paso	Descripción
	1	El usuario indica que quiere obtener un listado de los vehículos registrados.
	2	El sistema muestra por pantalla una tabla con los datos de todos los vehículos ingresados en el sistema. Además, proporciona al usuario filtros que le permitirá acotar el listado por: número de matrícula, el kilometraje, el último nivel de batería registrado, si está activo, si está averiado o no, la fecha de último mantenimiento y el secreto. Activando el filtro de mostrar solo vehículos averiados, se cumpliría con el CU15: consultar vehículos averiados.
Interfaz asociada	IU6 – Interfaz de listado de vehículos.	

Caso de uso	CU7 – Alta vehículo	
Descripción	El usuario con rol de jefe de logística debe poder dar de alta nuevos vehículos.	
Actores principales	Jefe de logística.	
Precondición	CU1 – Identificar en el sistema.	
Garantías en caso de éxito	El usuario con rol de jefe de logística ha dado de alta un vehículo en el sistema.	
Escenario principal de éxito	Paso	Descripción
	1	El jefe de logística indica que quiere dar de alta un nuevo vehículo.
	2	El sistema muestra la plantilla para introducir los datos del nuevo vehículo (matrícula, kilometraje, nivel de batería y secreto).
	3	El usuario jefe de logística introduce los datos del vehículo.
	4	El sistema recibe los datos, comprueba que son válidos y los guarda.
Escenario alternativo 1	4.1	El sistema comprueba que los datos introducidos no son válidos. Volvemos al paso 2.
Interfaz asociada	IU7 – Interfaz de creación de nuevo vehículo.	

Caso de uso	CU8 – Baja vehículo
Descripción	El usuario con rol de jefe de logística debe poder dar de baja un vehículo.
Actores principales	Jefe de logística.
Precondición	CU1 – Identificar en el sistema
Garantías en caso de	El usuario jefe de logística ha dado de baja un vehículo en el

éxito	sistema.	
Escenario principal de éxito	Paso	Descripción
	1	El jefe de logística indica que quiere dar de baja un vehículo.
	2	El sistema solicita la matrícula del vehículo.
	3	El jefe de logística introduce la matrícula del vehículo que quiere dar de baja.
	4	El sistema solicita confirmación al usuario.
	5	El usuario confirma.
	6	El sistema confirma la baja del vehículo.
Escenario alternativo 1	4.1	El sistema informa al usuario que la matrícula no existe. Volvemos al paso 2.
Interfaz asociada	IU8 – Interfaz de baja vehículo	

Caso de uso	CU9 – Listar repartidores	
Descripción	El usuario debe poder obtener un listado de los repartidores activos en el sistema.	
Actores principales	Jefe de logística.	
Precondición	CU1 – Identificar en el sistema.	
Garantías en caso de éxito	El usuario ha obtenido un listado de los repartidores registrados en el sistema.	
Escenario principal de éxito	Paso	Descripción
	1	El usuario indica que quiere obtener un listado de los repartidores registrados.
	2	El sistema muestra por pantalla una tabla con los datos de todos los repartidores ingresados en el sistema. Además, proporciona al usuario filtros que le permitirá acotar el listado por id y si está activo o no.
Interfaz asociada	IU9 – Interfaz de listado de repartidores.	

Caso de uso	CU10 – Alta repartidor	
Descripción	El usuario con rol de jefe de logística debe poder dar de alta nuevos repartidores.	
Actores principales	Jefe de logística.	
Precondición	CU1 – Identificar en el sistema.	
Garantías en caso de éxito	El usuario con rol de jefe de logística ha dado de alta un repartidor en el sistema.	
Escenario principal de éxito	Paso	Descripción
	1	El jefe de logística indica que quiere dar de alta un nuevo repartidor.
	2	El sistema muestra la plantilla para introducir los datos del nuevo repartidor (id).
	3	El usuario jefe de logística introduce el id del repartidor.
	4	El sistema recibe los datos, comprueba que son válidos y los guarda.
Escenario alternativo 1	4.1	El sistema comprueba que los datos introducidos no son válidos. Volvemos al paso 2.
Interfaz asociada	IU10 – Interfaz de creación de nuevo repartidor.	

Caso de uso	CU11 – Baja repartidor	
Descripción	El usuario con rol de jefe de logística debe poder dar de baja un repartidor.	
Actores principales	Jefe de logística.	
Precondición	CU1 – Identificar en el sistema.	
Garantías en caso de éxito	El usuario jefe de logística ha dado de baja un repartidor en el sistema.	
Escenario principal de éxito	Paso	Descripción
	1	El jefe de logística indica que quiere dar de baja un repartidor.
	2	El sistema solicita el id del repartidor.
	3	El jefe de logística introduce la id del repartidor que quiere dar de baja.
	4	El sistema solicita confirmación al usuario.
	5	El usuario confirma.
	6	El sistema confirma la baja del repartidor.
Escenario alternativo 1	4.1	El sistema informa al usuario que el id de repartidor no existe. Volvemos al paso 2.
Interfaz asociada	IU11 – Interfaz de baja repartidor.	

Caso de uso	CU12 – Informar avería	
Descripción	El sistema vehículo comunica al sistema que tiene una avería.	
Actores principales	Vehículo.	
Precondición	-	
Garantías en caso de éxito	El sistema ha registrado que el vehículo tiene una avería.	
Escenario principal de éxito	Paso	Descripción
	1	Uno de los sistemas vehículo comunica al sistema que ha sufrido una avería.
	2	El sistema recibe la comunicación y registra la avería.
Interfaz asociada	-	

Caso de uso	CU13 – Enviar estado	
Descripción	El sistema vehículo comunica al sistema un registro de estado, incluyendo nivel de batería y kilometraje.	
Actores principales	Vehículo.	
Precondición	-	
Garantías en caso de éxito	El sistema ha registrado el estado del vehículo.	
Escenario principal de éxito	Paso	Descripción
	1	Uno de los sistemas vehículo comunica al sistema sus datos relativos a kilometraje y nivel de batería.
	2	El sistema recibe la comunicación y registra los datos.
Interfaz asociada	-	

Caso de uso	CU14 – Marcar vehículo como averiado
Descripción	El usuario con rol jefe de logística debe poder marcar un vehículo

	concreto como averiado.	
Actores principales	Jefe de logística.	
Precondición	CU1 – Identificar en el sistema.	
Garantías en caso de éxito	El sistema ha registrado la avería del vehículo.	
Escenario principal de éxito	Paso	Descripción
	1	El usuario jefe de logística indica al sistema que desea marcar un vehículo como averiado.
	2	El sistema solicita la matrícula del vehículo.
	3	El usuario jefe de logística introduce la matrícula del vehículo que quiere marcar como averiado.
	4	El sistema registra los datos.
Escenario alternativo 1	4.1	La matrícula introducida no coincide con ningún vehículo registrado. Volvemos al paso 2.
Interfaz asociada	IU12 – Interfaz marcar vehículo como averiado.	

Caso de uso	CU16 – Marcar vehículo como reparado	
Descripción	El usuario con rol jefe de logística debe poder marcar un vehículo concreto como reparado.	
Actores principales	Jefe de logística.	
Precondición	CU1 – Identificar en el sistema.	
Garantías en caso de éxito	El sistema ha registrado la reparación del vehículo.	
Escenario principal de éxito	Paso	Descripción
	1	El usuario jefe de logística indica al sistema que desea marcar un vehículo como reparado.
	2	El sistema solicita la matrícula del vehículo.
	3	El usuario jefe de logística introduce la matrícula del vehículo que quiere marcar como reparado.
	4	El sistema registra los datos.
Escenario alternativo 1	4.1	La matrícula introducida no coincide con ningún vehículo registrado. Volvemos al paso 2.
Interfaz asociada	IU13 – Interfaz marcar vehículo como reparado.	

Caso de uso	CU19 – Gestionar mantenimientos	
Descripción	El usuario con rol jefe de logística debe poder visualizar los mantenimientos de un vehículo, crear nuevos, borrar y marcar como completados.	
Actores principales	Jefe de logística.	
Precondición	CU1 – Identificar en el sistema.	
Garantías en caso de éxito	El sistema ha registrado cambios en los mantenimientos de un vehículo.	
Escenario principal de éxito	Paso	Descripción
	1	El usuario jefe de logística indica al sistema que desea consultar los mantenimientos relativos a un vehículo.
	2	El sistema solicita la matrícula del vehículo.
	3	El usuario jefe de logística introduce la matrícula del vehículo el cual quiere consultar los mantenimientos.

	4	El sistema muestra un listado con los mantenimientos registrados de un vehículo, indicando: • Descripción del mantenimiento. • Fecha concertada para el mantenimiento. • Campo indicando si se ha realizado o no. El sistema permitirá al usuario modificar los datos de un mantenimiento, borrarlo o crear uno nuevo. Una vez realizado los cambios, el sistema registrará las modificaciones.
Escenario alternativo 1	3.1	La matrícula introducida no coincide con ningún vehículo registrado. Volvemos al paso 2.
Interfaz asociada	IU15 – Interfaz gestión de mantenimientos vehículo.	

2.6.3 Paquete Registro

A continuación, se detallan los casos de usuario pertenecientes al paquete registro.

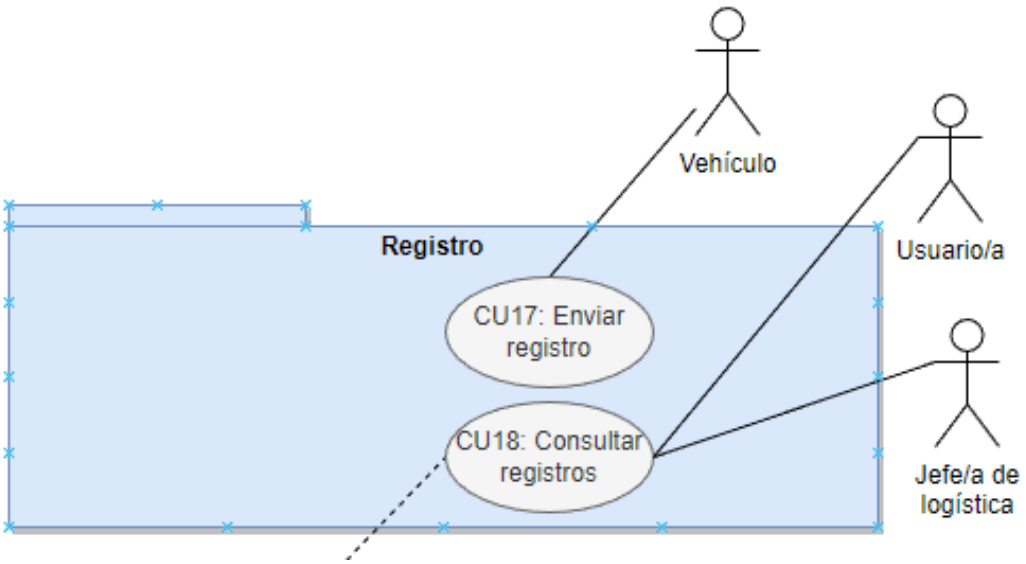


Ilustración 4 – Casos de uso paquete Registro

Caso de uso	**CU17 – Enviar registro de ubicación**	
Descripción	Cualquiera de los múltiples sistemas vehículo deben poder enviar al sistema los datos de su ubicación actual.	
Actores principales	Vehículo.	
Precondición	-	
Garantías en caso de éxito	El sistema ha registrado la comunicación del vehículo.	
Escenario principal de éxito	**Paso**	**Descripción**
	1	El sistema vehículo envía al sistema las coordenadas de su ubicación (latitud y longitud), su matrícula y el id del repartidor que la conduce.
	2	El sistema valida y registra las coordenadas, la matrícula del vehículo, el id del repartidor y una marca de tiempo

		del momento del envío.
Interfaz asociada	-	

Caso de uso	CU18 – Consultar registros	
Descripción	Cualquier usuario o jefe de logística debe poder visualizar las ubicaciones registradas de cada uno de los vehículos.	
Actores principales	Usuario y jefe de logística.	
Precondición	CU1 – Identificar en el sistema.	
Garantías en caso de éxito	El sistema ha proporcionado los registros de ubicación del vehículo que haya consultado.	
Escenario principal de éxito	Paso	Descripción
	1	El usuario indica al sistema que desea consultar los registros de un vehículo.
	2	El sistema muestra una serie de filtros para acotar la búsqueda: • Matrícula: desplegable que contiene las matrículas activas. • Repartidor: desplegable que contiene los id de repartidor. • Selector de fecha, diaria. Además, el sistema recuerda al usuario que, para obtener un listado, debe indicar un número de matrícula o un id de repartidor.
	3	El usuario selecciona los filtros que desea y presiona el botón consultar.
	4	El sistema devuelve los datos registrados en base a los filtros del usuario en forma de un mapa interactivo donde se marcan las ubicaciones registradas.
Interfaz asociada	IU14 – Interfaz registros de ubicación.	

3. Diseño

3.1 Introducción

En esta sección, se detalla el diseño que será aplicado en la fase de implementación y las decisiones tomadas a la hora de especificar la tipología de arquitectura a seguir y el *stack* tecnológico.

3.2 Arquitectura

3.2.1 Planteamiento

Según lo estudiado en la fase de análisis, la aplicación ofrece dos tipologías de servicio claramente identificados:

- Sistema de información: los usuarios usarán el sistema para obtener información sobre la ubicación y estado de los vehículos, así como para la gestión de usuarios del sistema.

- Servicio para registro de datos: los vehículos enviarán información de manera constante con periodos parametrizables, lo que generará gran afluencia de conexiones e intercambio de información. Para cuantificar la cantidad de peticiones que recibirá la aplicación por parte de los vehículos, si consideramos que el cliente mantiene los parámetros de frecuencia por defecto, podemos hacer el siguiente cálculo:

    - Número de vehículos: 12
    - Frecuencia de comunicación ubicación del vehículo: 30 segundos
    - Frecuencia de comunicación estado del vehículo: 60 segundos
    - Total: 36 peticiones/minuto.

Teniendo en cuenta las dos tipologías de uso que puede tener la aplicación, es conveniente plantearse una arquitectura distribuida en el que un componente o servicio cumpla la función de sistema de información, y otro componente se encargue del registro de datos. De esta manera, garantizamos que, ante una ampliación de flota o una disminución de los intervalos de envío de información de los vehículos al sistema, el sistema de información sigue sirviendo las solicitudes de los usuarios y en tiempo razonable.

Por estos motivos, se descarta una arquitectura de aplicación monolítica y se opta por considerar una arquitectura basada en microservicios.

3.2.2 Arquitectura de microservicios

La arquitectura de microservicios permite aplicar la división de una aplicación en múltiple aplicaciones, más pequeñas e independientes, donde cada una tiene un

conjunto de responsabilidades. Para aplicar este tipo de diseño, se crearán dos microservicios:

- *Fleet-manager-app*: su responsabilidad será servir como sistema de información. Servirá a los frontales de los datos que los usuarios requieran. La comunicación será mediante arquitectura REST sobre protocolo de transporte HTTP(s), ofreciendo una interfaz estándar permitiendo integraciones con cualquier otro sistema.
- *Fleet-manager-receiver*: la responsabilidad de este microservicio será el registro de los datos que los vehículos envíen. Compartirá la tipología de arquitectura con la aplicación *fleet-manager-app*.

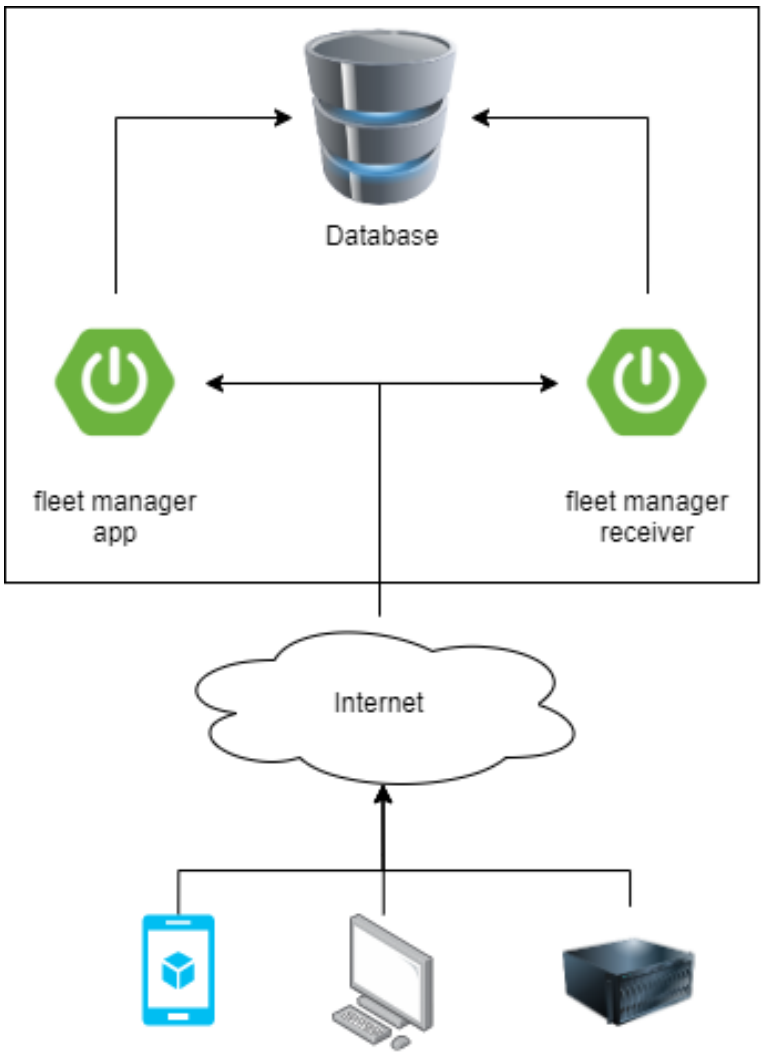


Ilustración 5 - Diagrama arquitectura

3.2.3 Arquitectura por capas

El modelo de arquitectura escogido para la implementación de las aplicaciones *spring boot* será el modelo por capas. Concretamente, serán tres capas, las siguientes:

- Controlador: en esta capa se gestionarán las peticiones a la interfaz REST para, posteriormente, ser traspasadas a la capa de servicio.
- Servicio: capa en la que se implementará la lógica de negocio.
- Repositorio: encargada de gestionar las operaciones de persistencia sobre los objetos del modelo (o dominio).

3.3 Clases principales

El conjunto de clases principales, que formarán parte del dominio (modelo), vendrá especificado por el siguiente diagrama:

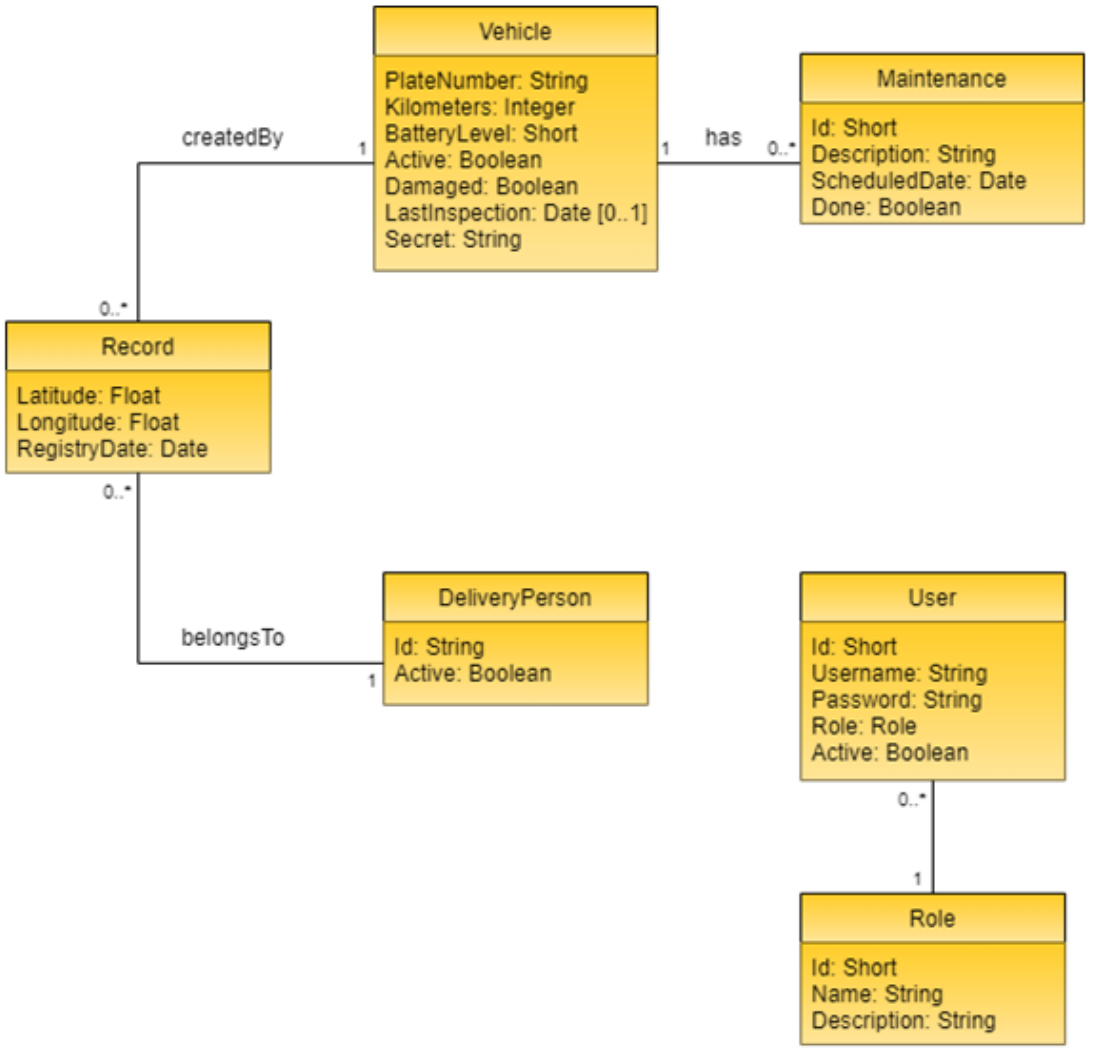


Ilustración 6 - Diagrama de clases principales

3.4 *Stack* tecnológico

En los siguientes puntos se nombran y detallan cada una de las tecnologías que se usarán para llevar a cabo el proyecto. Todas las herramientas y tecnologías propuestas

son de código libre. Por claridad y para garantizar un orden correcto, se dividen por el ámbito al que pertenecen: desarrollo, sistemas y herramientas auxiliares.

3.4.1 Ámbito: desarrollo

En este punto se enumeran y detallan las tecnologías que se usarán para el desarrollo de la aplicación (en el más estricto sentido).

3.4.1.1 Java versión 17

Lenguaje de programación, de propósito general, alto nivel, orientado a objetos y basado en clases. Desarrollado por *Sun Microsystems* y cuya primera aparición fue en Mayo del 1995. Actualmente, según el índice de *TIOBE Programming Community*, Java es el tercer lenguaje de programación más usado.

Ilustración 7 - Java

3.4.1.2 *Spring framework*

Framework de aplicación y contenedor de inversión de dependencias, de código abierto, para plataformas Java. Puede utilizarse sobre cualquier versión de java, no impone ningún modelo de programación específico y su popularidad ha crecido exponencialmente ya que ofrece valor adicional sobre el modelo EJB (*Enterprise JavaBeans*). Licenciado por *Apache License* 2.0.

Ilustración 8 - Spring framework

3.4.1.3 *JPA (Java Persistence API)*

API de persistencia desarrollado por *Sun Microsystems* para plataformas Java EE. Se encarga del manejo de datos relacionales en aplicaciones Java. Su objetivo es no perder las ventajas de la orientación a objetos al interactuar con bases de datos, mediante el uso de POJOs (*Plain Old Java Objects*).

3.4.1.3 Hibernate

Herramienta de mapeo objeto-relacional para Java. Proporciona un *framework* para el mapeado del dominio orientado a objetos hacia una base de datos relacional. En parte, implementa y extiende funcionalidades de JPA.

Ilustración 9 - Hibernate

3.4.1.4 PostgreSQL

Sistema gestor de base de datos relacional de código abierto bajo su propia licencia PostgreSQL License.

Ilustración 10 - PostgreSQL

3.4.1.5 JUnit

Framework de pruebas unitarias para aplicaciones Java, con licencia EPL 2.0 y que, actualmente, se encuentra en su versión 5.

Ilustración 11 - JUnit

3.4.1.6 Mockito

Framework que ofrece la posibilidad de crear objetos que simulan a objetos reales con la finalidad de testear el código y simular los casos de uso que se implementan en el código. Junto a JUnit, forman una herramienta muy potente orientada al *testing*.

Ilustración 12 - Mockito

3.4.1.7 ReactJS

Biblioteca de *Javascript*, de código abierto (licencia MIT), desarrollada por Facebook y la comunidad, diseñada para crear interfaces web de usuario con el objetivo de crear aplicaciones de una sola página.

Ilustración 13 - ReactJS

3.4.2 Ámbito: *DevOps*

En este punto se listan las herramientas que se usarán en el proyecto y que están más ligadas a la disciplina de sistemas y operaciones.

3.4.2.1 Docker

Conjunto de herramientas que se apoya en la virtualización a nivel de sistema operativo para la entrega de software en paquetes (comúnmente referidos como contenedores). Cada contenedor está compuesto por su propio software, librerías y archivos de configuración que, junto a una capa de abstracción, proporcionan un uso del container en la mayoría de sistemas operativos.

Ilustración 14 - Docker

3.4.2.2 Kubernetes

Plataforma portable y extensible, de código abierto, para administrar cargas de trabajo y servicios. En este proyecto, se usará para el despliegue de contenedores y la creación de servicios relacionados con la aplicación.

Ilustración 15 - Kubernetes

3.4.2.3 Google Cloud Platform (GCP)

Plataforma de servicios que ofrece un conjunto de herramientas y servicios, que son ejecutados en la misma red que Google usa internamente en sus propios productos, para la computación en la nube. En este proyecto, nos servirá para el despliegue en la nube, usando *Google Kubernetes Engine*, de nuestra aplicación.

Ilustración 16 - Google Cloud

3.4.3 Ámbito: herramientas auxiliares

En este punto se listan las herramientas auxiliares que se utilizarán y enriquecerán el proyecto en arras de trabajar en una aplicación rica en instrumentos estándar de la industria.

3.4.3.1 IntelliJ IDEA

Entorno de desarrollo integrado (del inglés *Integrated Development Environment*, IDE), para aplicaciones escritas en Java, Kotlin o Groovy. El fabricante es JetBrains y, aunque la versión *Ultimate* no es accesible gratuitamente, dispone de una versión *Community* licenciada por *Apache License* 2.0.

Ilustración 17 - IntelliJ IDEA

3.4.3.2 Git y GitLab

Sistema distribuido de control de versiones diseñado para soportar proyectos de cualquier magnitud. Sus funcionalidades más reseñables son el sistema de ramas y la posibilidad de crear múltiples flujos de trabajo. De código libre y abierto.
Para explotar al máximo las posibilidades de Git, usaremos específicamente la plataforma GitLab (www.gitlab.com).

Ilustración 18 – Git

Ilustración 19 - GitLab

3.4.3.3 Maven

Herramienta para la automatización de construcción de aplicaciones y la gestión de dependencias. Licencia *Apache License* 2.0.

Ilustración 20 - Maven

3.4.3.4 SonarQube

Plataforma, de código abierto, para la continua revisión e inspección de la calidad del código. Disponible para 29 lenguajes de programación diferente. Mediante el análisis estático, el *software* detecta bugs y *code smells*, ofreciendo avisos, consejos y documentación sobre las posibles carencias de nuestro código.

Ilustración 21 - SonarQube

3.4 Componentes

A continuación, se especifica el diseño de los componentes para cada uno de los microservicios.

3.4.1 *Fleet manager receiver*

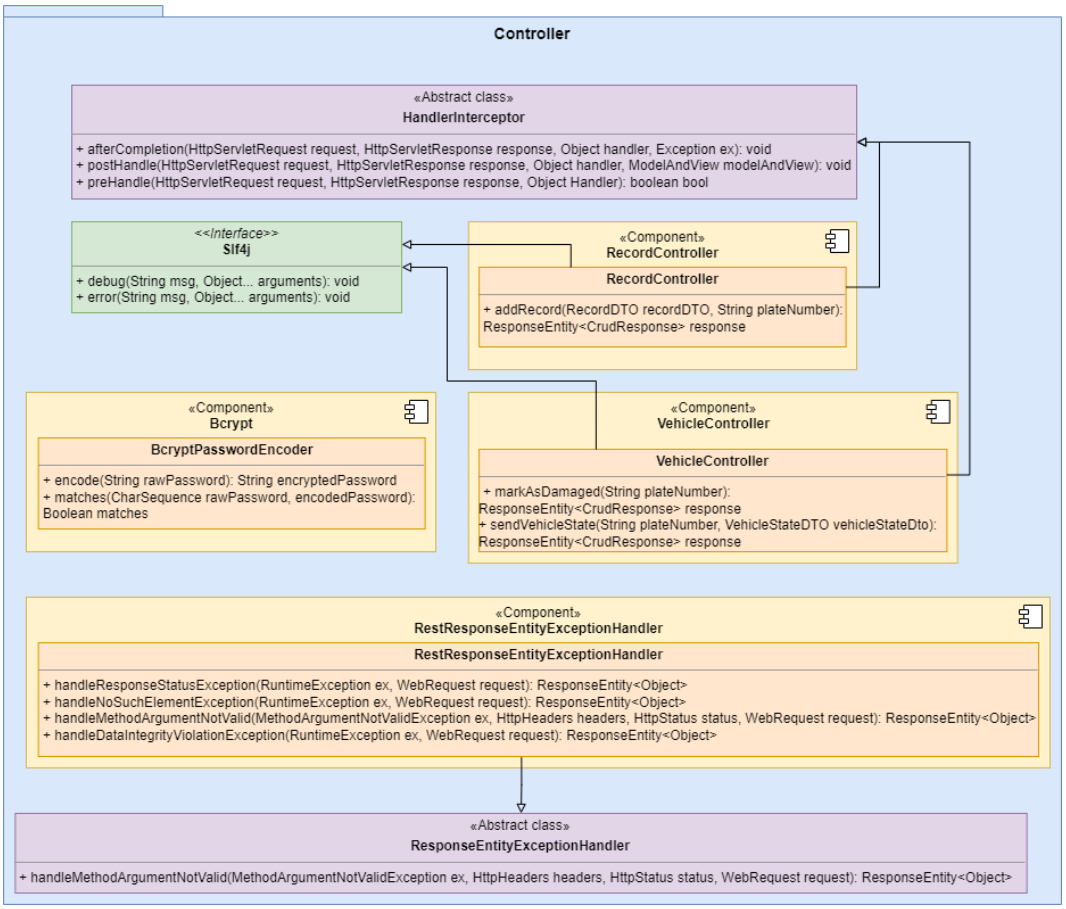


Ilustración 22 - Componentes capa controlador

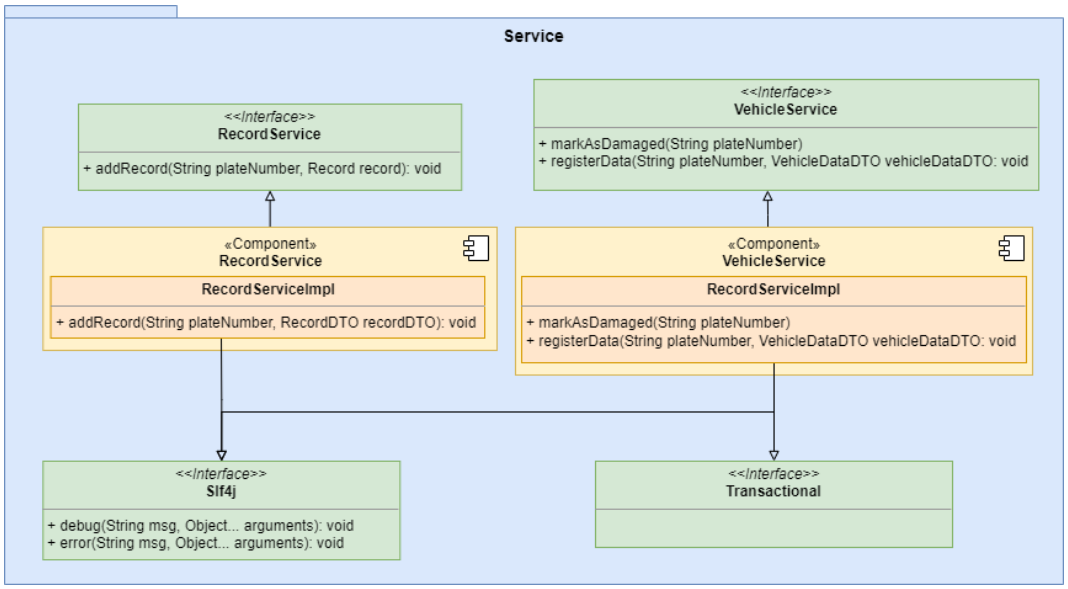


Ilustración 23 - Componentes capa servicio

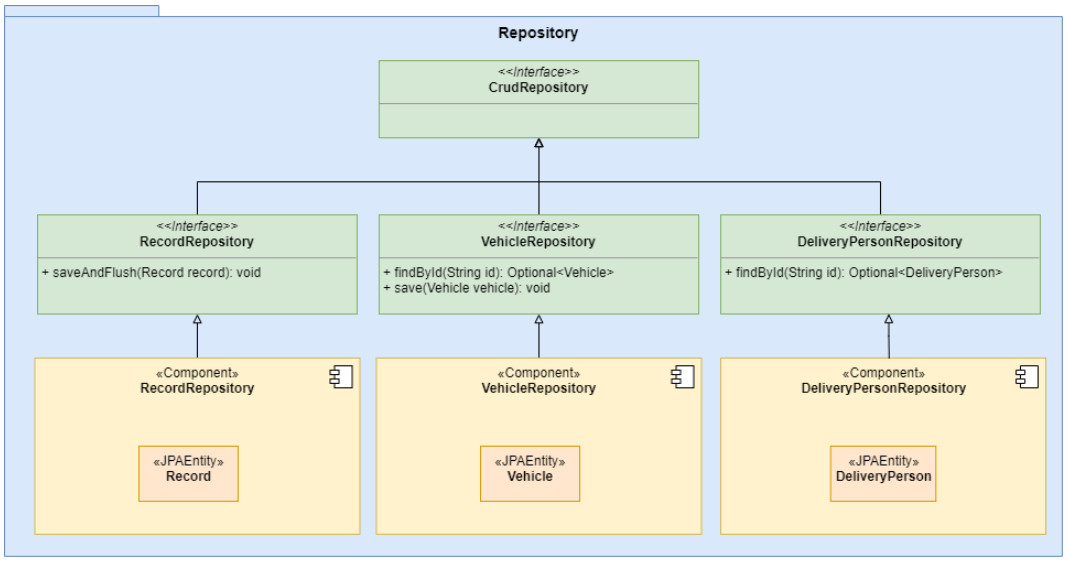


Ilustración 24 - Componentes capa repositorio

3.4.2 *Fleet manager application*

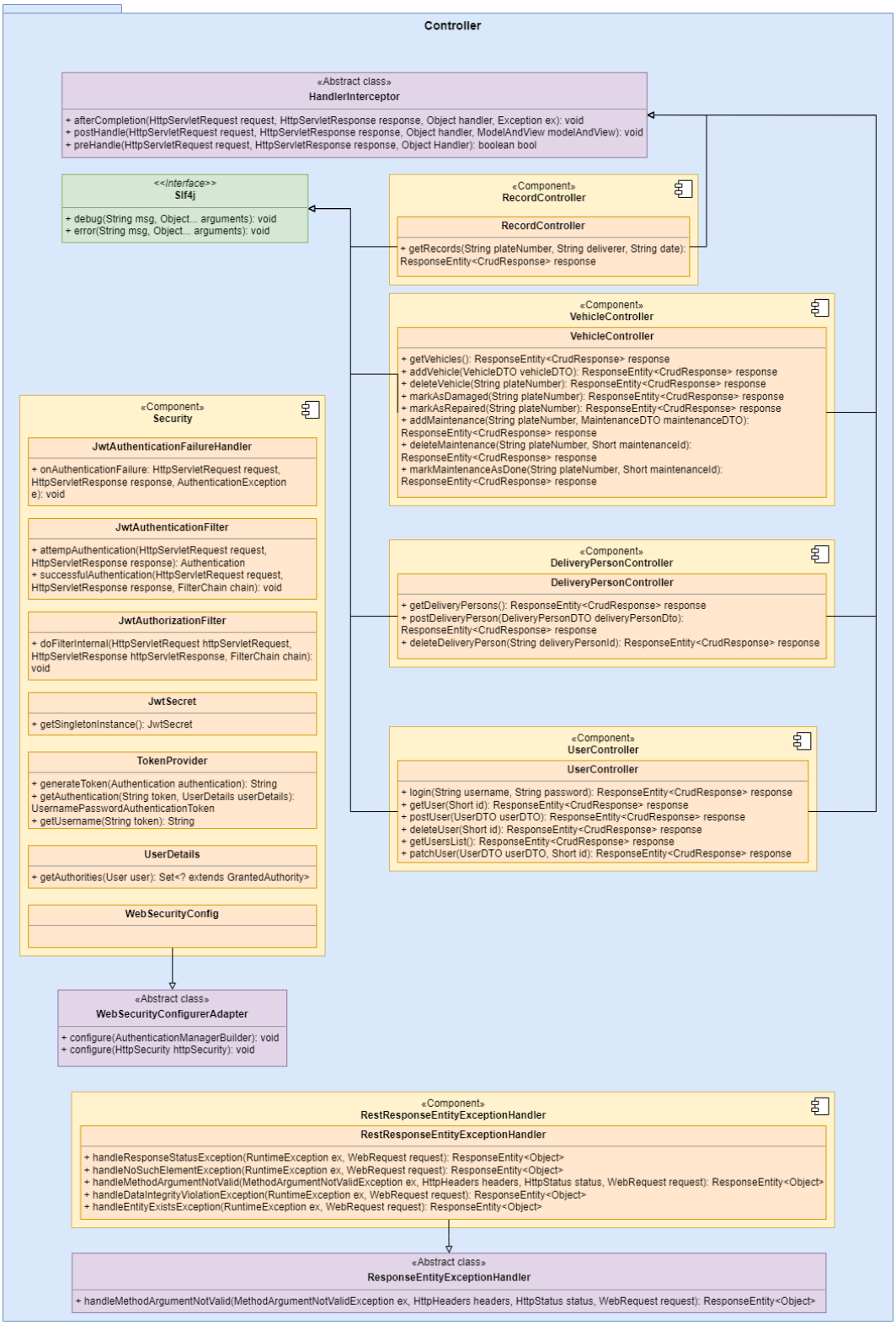


Ilustración 25 - Componentes capa controlador

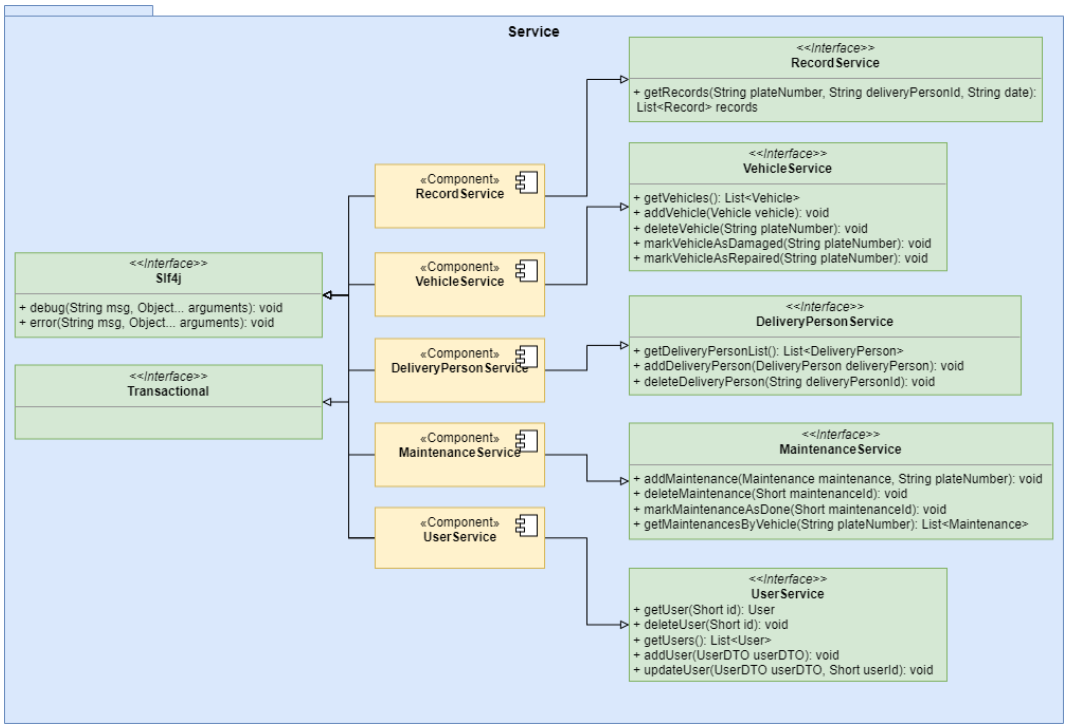


Ilustración 26 - Componentes capa servicio

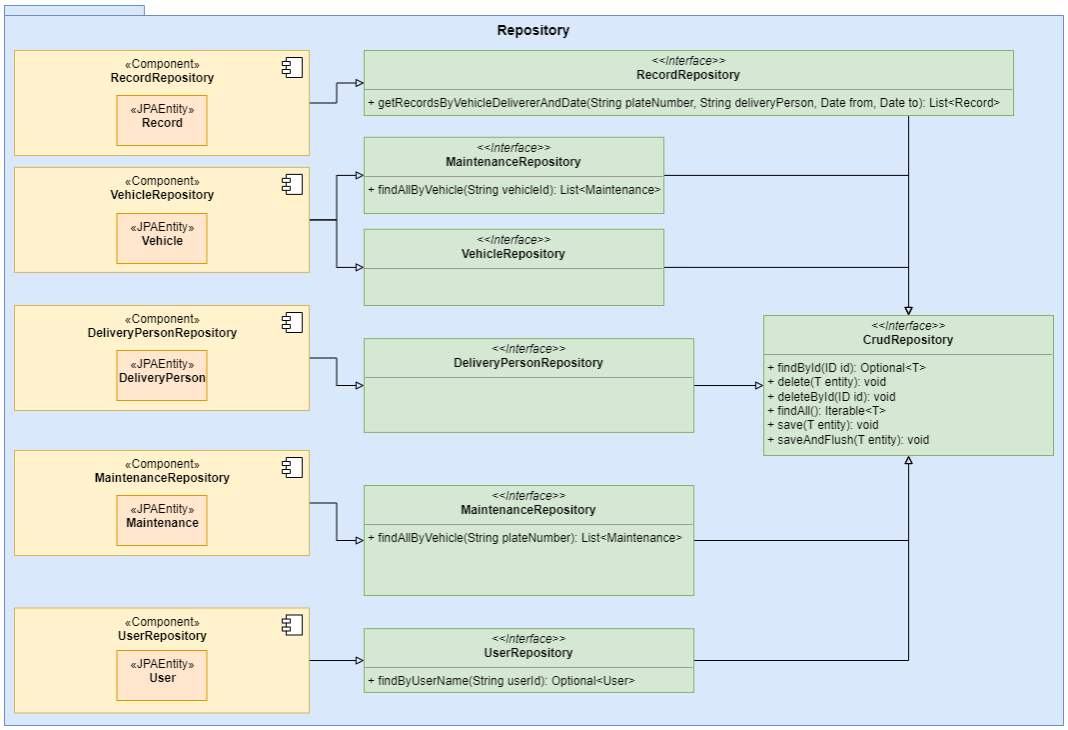


Ilustración 27 - Componentes capa repositorio

3.5 Persistencia de datos

PostgreSQL será el sistema encargado de la persistencia de los datos. El diseño conceptual se muestra en la ilustración 2. La relación entre las tablas *Record* y las múltiples tablas con patrón *Record_mesaño* se detalla en la siguiente sección.

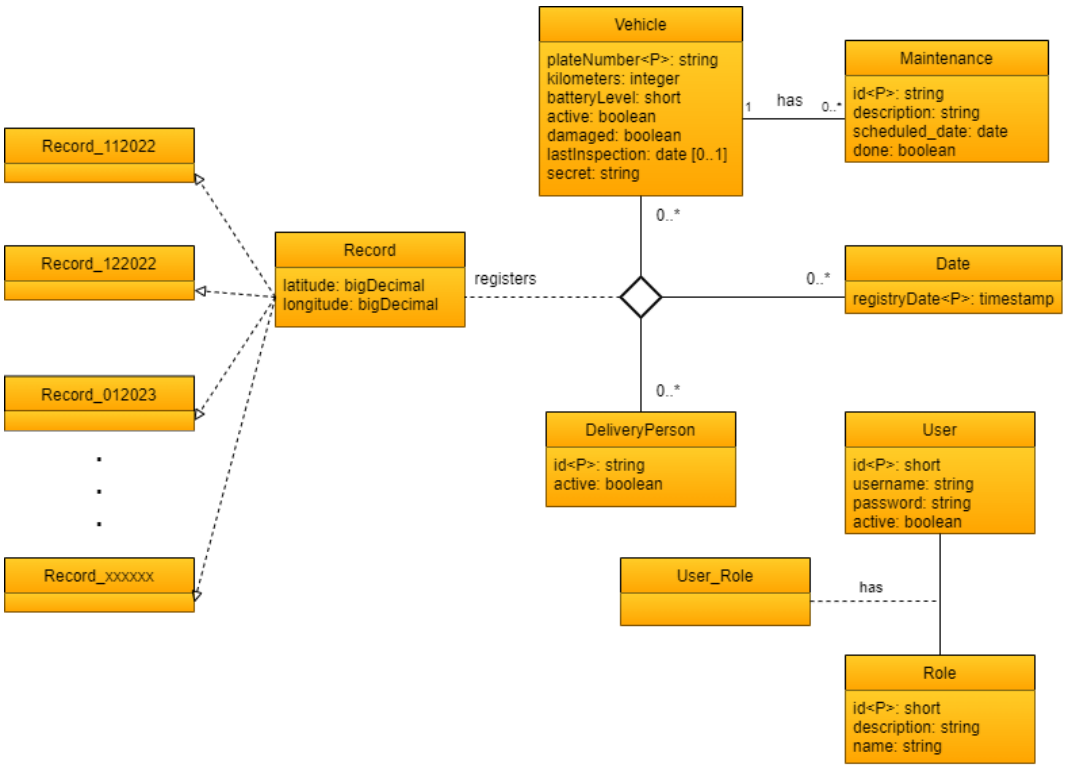


Ilustración 28 - Diseño conceptual base de datos

3.5.1 Particionamiento

En la tabla *Record* se nos presenta una casuística en la que, a la hora de la implementación, debemos considerar en esta fase de diseño. Esta tabla, que almacenará los registros de ubicación que envíen los vehículos puede, con el tiempo, alcanzar un volumen que merme el rendimiento del sistema. En base a los siguientes cálculos:

Número de vehículos actual: 12
Envío de registros: 1 por minuto durante 8 horas (según valor estándar parametrizado, puede ser más frecuente, lo que incrementaría el volumen de almacenamiento requerido)
Cantidad total de registros diarios:
1 registro · 60 minutos · 8 horas · 12 número de vehículos = 5.760 registros

Si multiplicamos ese número de registros por veinte (número de jornadas laborales al mes de media) obtenemos que, cada mes, la cantidad de registros acumulados

incrementa en 115.200 registros. En un año tendríamos, aproximadamente, 1.382.400 registros.

Como medida de prevención utilizaremos una técnica que nos ofrece el motor PostgreSQL, el particionamiento de tablas. La partición será por rango de fechas, con periodicidad mensual. PostgreSQL creará una tabla mensual en la que guardará los registros generados en ese rango temporal.
El particionamiento de tablas es un mecanismo abstracto, es decir, no requiere de procedimientos específicos a la hora de consultar, insertar o leer de la tabla. Mediante una consulta directa a la tabla *Record*, el motor ya se encarga de hacer la operación en la tabla particionada respectiva.
Más información sobre particionamiento de tablas en: https://www.postgresql.org/docs/current/ddl-partitioning.html .

3.6 Despliegue

El despliegue de la aplicación debe hacerse completamente en *cloud* ya que la empresa no dispone de recursos IT. Esto, añade complejidad al proyecto y supone un reto a nivel de seguridad y acceso a los datos. Se detalla sobre la seguridad en la siguiente sección (2.7 - seguridad).

Para cumplir con el requisito, debe considerarse un proveedor de servicios *cloud*. Haciendo un estudio de los proveedores disponibles y sus respectivos costes, las opciones disponibles son: *Amazon Web Services*, *Google Cloud Platform*, Microsoft Azure, Linode, Heroku, entre otros. Aunque todos ofrecen los mismos servicios de una manera muy similar, me decantaré por el proveedor Google y su plataforma *Google Cloud Platform* (en adelante, GCP).

3.6.1 *Google cloud platform* (GCP)

GCP nos ofrece varios tipos de servicios para el despliegue de aplicaciones: máquinas virtuales o *compute engines* (IaaS, del inglés *Infrastructure as a Service*), un *cluster* de Kubernetes denominado Google Kubernetes Engine (SaaS, del inglés *Software as a Service*).
Me decantaré por el *cluster* de kubernetes de Google, ya que simplifica, en cierta medida, las tareas relacionadas con configuración de sistemas y nos proporciona funcionalidades a nivel de seguridad como estándar.

El diseño del *cluster* se especifica en la ilustración 7.

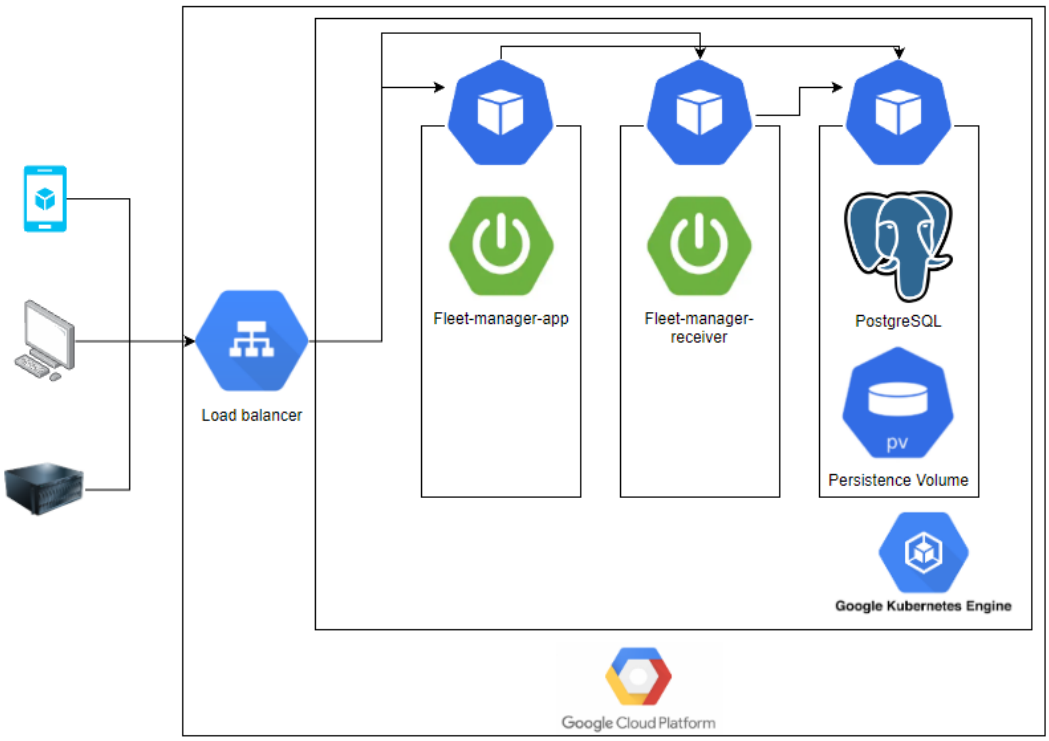


Ilustración 29 - Cluster

3.7 Seguridad

En este apartado, se especifica los diferentes aspectos relativos a la seguridad que deberán cumplir los diferentes componentes que forman el servicio completo.

3.7.1 Base de datos

La base de datos deberá ser accesible únicamente desde direcciones y/o recursos ubicados dentro del mismo *cluster*, imposibilitando en toda medida su accesibilidad desde recursos fuera del mismo.

Se usará un mecanismo de control de usuario. Los únicos usuarios que tendrán acceso a los datos serán:

- *fleet_manager_owner*: propietario de la base de datos, permisos completos.
- *fleet_manager_app*: usuario utilizado por la aplicación *fleet-manager-app*. Permisos de lectura y escritura en todas las tablas.
- *fleet_manager_receiver_app*: usuario utilizado por la aplicación *fleet-manager-receiver*. Permisos de escritura sobre la tabla *Record*, lectura y escritura en la tabla *Vehicle* y lectura en *DeliveryPerson*.

3.7.2 Componente: *Fleet manager receiver*

La API expuesta por este componente es de arquitectura REST sobre protocolo HTTP/TLSv1.2, lo que garantiza un canal de comunicación encriptado. Únicamente ofrece métodos de escritura:

- Insertar *record*: método POST.
- Marcar vehículo como averiado: método PATCH.

Teniendo en cuenta los puntos anteriormente detallados, se considera que el intercambio del secreto, después de un proceso *hashing*, servirá para autenticar cada uno de los vehículos en la aplicación. Por lo tanto, se idea el siguiente procedimiento para garantizar la autenticidad de la comunicación:

1. El vehículo formará una cadena de caracteres con la siguiente estructura: {matrícula}{secreto}{fecha en formato dd/MM/aaaa}

2. El vehículo procesará la cadena con el algoritmo de cifrado hash Bcrypt (https://en.wikipedia.org/wiki/Bcrypt) con fuerza 4.

3. El vehículo incluirá el valor hash como secreto en cabecera de la petición HTTP.

4. El servidor extraerá el secreto de la cabecera, recuperará de la base de datos el secreto del vehículo en base a su matrícula, formará la cadena de caracteres que corresponda, y comprobará si el secreto enviado en la cabecera corresponde con el calculado.

3.7.3 Componente: *Fleet manager app*

Al igual que el componente *fleet manager receiver*, este componente también expondrá una API REST sobre protocolo HTTP/TLSv1.2. Además, deberemos de implementar un sistema de autorización y autenticación por roles. Para conseguirlo, haré uso del estándar JSON Web Token (https://jwt.io/) con la ayuda de la librería específica de *Spring security*.

3.8 Prueba de concepto

En esta prueba de concepto, se implementarán, de manera preliminar, dos de los casos de uso de la aplicación. Con esto, quiero confirmar la viabilidad de la implementación del proyecto utilizando las tecnologías propuestas.

Los casos de uso que se implementarán serán:

- CU14 – Marcar vehículo como averiado
- CU17 – Enviar registro de ubicación

Para el caso de usuario CU14 se implementará el algoritmo de cifrado propuesto en la fase de diseño, apartado "2.8.2 Componente: *Fleet manager receiver*", para demostrar su viabilidad.

3.8.1 Desarrollo

3.8.1.1 Repositorio

El código utilizado es accesible, de manera pública, en el siguiente repositorio:
https://gitlab.com/uoc-tfg-jarubioca/fleet-manager-data-receiver-spring
La rama que se desplegará será la nombrada "poc".

3.8.2 Despliegue

El despliegue de la aplicación, según requerimientos del cliente, debe ser en entorno *cloud*. El proveedor de servicios *cloud* elegido es *Google Cloud Platform*. Antes de desplegar la aplicación, he tenido que diversidad de acciones:

- Creación del proyecto:

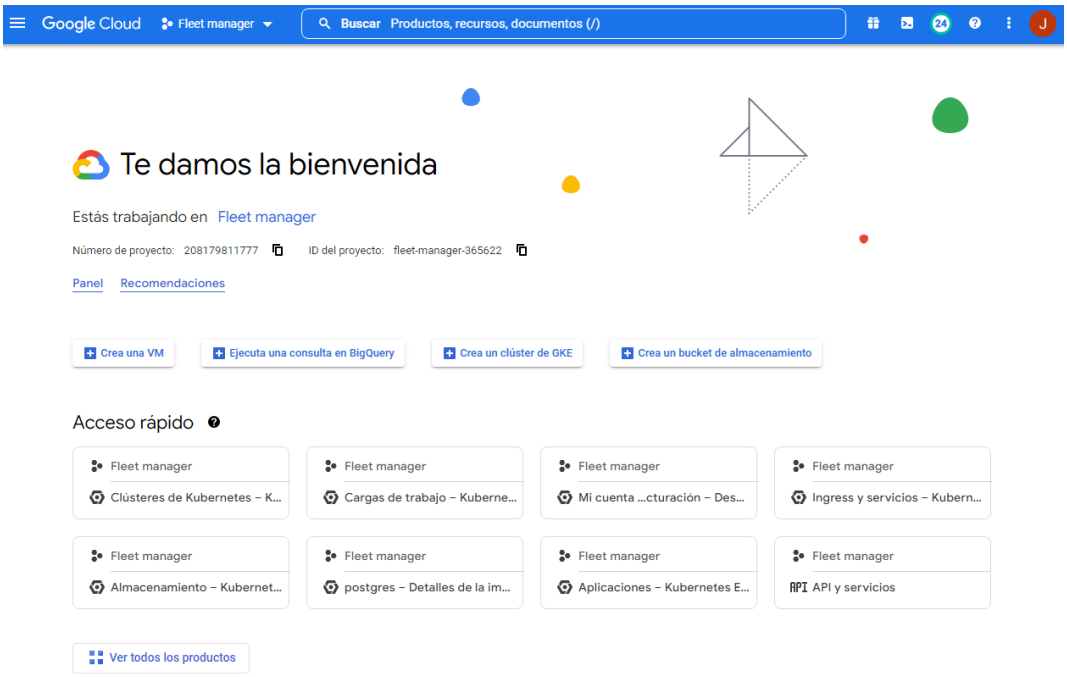


Ilustración 30 - Proyecto GCP

- Creación del motor kubernetes de Google:

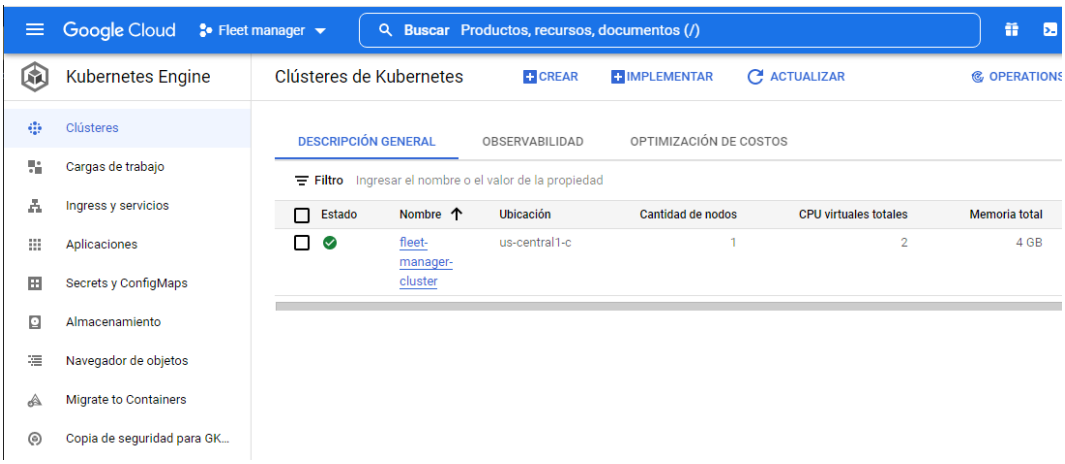


Ilustración 31 - Google Kubernetes Engine

Una vez creado el proyecto y el *cluster*, ya puedo crear los recursos que necesitaré para la puesta en marcha de la aplicación.

3.8.2.1 Base de datos

Una de las decisiones que he tomado es la creación de un volumen de persistencia al que accederé desde una instancia de PostgreSQL desde el *cluster*.

3.8.2.1.1 Volumen de persistencia

Las especificaciones del volumen de persistencia vienen marcados por el siguiente deployment.yaml:

```yaml
kind: StorageClass
apiVersion: storage.k8s.io/v1
metadata:
  name: regionalpd-storageclass
provisioner: pd.csi.storage.gke.io
parameters:
  type: pd-standard
  replication-type: regional-pd
volumeBindingMode: WaitForFirstConsumer
allowedTopologies:
  - matchLabelExpressions:
      - key: topology.gke.io/zone
        values:
          - europe-west1-b
          - europe-west1-c
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: regional-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 200Gi
  storageClassName: regionalpd-storageclass

```

Ilustración 32 – Deployment.yml Volumen de persistencia

Enlace de gitlab al *yaml* de despliegue: https://gitlab.com/uoc-tfg-jarubioca/fleet-manager-data-receiver-spring/-/blob/poc/postgres-pv.yml

Aunque el almacenamiento parezca sobredimensionado (200Gi, Gibibytes), es el tamaño mínimo que permite asignar GCP.

Confirmo que el volumen de persistencia se ha creado en la ilustración 4.

```
PS C:\Users\Javi\Desktop\Google Drive\UOC\TFG\Software\git\fleet-manager-data-receiver-spring> kubectl get pv
NAME                                       CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS   CLAIM                      STORAGECLASS              REASON   AGE
pv-fleet-manager                           1Gi        RWO            Retain           Bound    default/pvc-fleet-manager  local-storage                      8d
pvc-aa07ce86-b17f-427f-b64f-bb20f30ad4ea   200Gi      RWO            Delete           Bound    default/regional-pvc       regionalpd-storageclass            8d
PS C:\Users\Javi\Desktop\Google Drive\UOC\TFG\Software\git\fleet-manager-data-receiver-spring> $
```

Ilustración 33 - Volumen de persistencia

3.8.2.1.2 Instancia de PostgreSQL

Se crea una instancia de PostgreSQL, versión 15.0.1, en el *cluster*. Para ello, creamos el siguiente deployment.yaml, accesible en el repositorio desde:

https://gitlab.com/uoc-tfg-jarubioca/fleet-manager-data-receiver-spring/-/blob/poc/postgres-deployment.yml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: postgres
spec:
  strategy:
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 1
    type: RollingUpdate
  replicas: 1
  selector:
    matchLabels:
      app: postgres
  template:
    metadata:
      labels:
        app: postgres
    spec:
      containers:
        - name: postgres
          image: postgres
          resources:
            limits:
              cpu: 50m
              memory: "0.5Gi"
            requests:
              cpu: 50m
              memory: "0.5Gi"
          ports:
            - containerPort: 5432
          env:
            - name: POSTGRES_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: postgressecret
                  key: password
            - name: PGDATA
              value: /var/lib/postgresql/data/pgdata
          volumeMounts:
            - mountPath: /var/lib/postgresql/data
              name: postgredb
      volumes:
        - name: postgredb
          persistentVolumeClaim:
            claimName: regional-pvc
```

Ilustración 34 – Deployment.yml postgreSQL

Confirmamos que se ha creado un pod en el *cluster* con la instancia de postgreSQL y levanta sin errores en la siguiente ilustración.

```
PS C:\Users\Javi\Desktop\Google Drive\UOC\TFG\Software\git\fleet-manager-data-receiver-spring> kubectl get pods | Select-String -Pattern "postgres"
postgres-666f4547b-vzfnn                1/1     Running   0          8d
```

Ilustración 35 - Pod PostgreSQL

El siguiente paso sería exponer un servicio al pod de postgreSQL para que sea accesible desde otros pods del mismo cluster. Para ello, generamos el siguiente deployment.yml:

```yaml
apiVersion: v1
kind: Service
metadata:
  name: postgres
spec:
  ports:
    - port: 5432
      targetPort: 5432
  selector:
    app: postgres
```

Ilustración 36 - Deployment.yml servicio PostgreSQL

3.8.2.1.3 Creación del esquema de la base de datos

El siguiente paso es la creación de la base de datos. El script de creación está accesible en el repositorio mediante el siguiente enlace:

https://gitlab.com/uoc-tfg-jarubioca/fleet-manager-data-receiver-spring/-/blob/poc/scripts/db_create_script.sql

En la siguiente captura, se muestra la conexión al pod, el *login* a psql y la confirmación de que el esquema se ha creado. No se incluye el proceso de creación ya que la consola ha creado cientos de líneas.

```
PS C:\Users\Javi\Desktop\Google Drive\UOC\TFG\Software\git\fleet-manager-data-receiver-spring> kubectl exec -it postgres-666f454f7b-vzfnn sh
kubectl exec [POD] [COMMAND] is DEPRECATED and will be removed in a future version. Use kubectl exec [POD] -- [COMMAND] instead.
# psql -U postgres
psql (15.0 (Debian 15.0-1.pgdg110+1))
Type "help" for help.

postgres=# \dn
        List of schemas
      Name      |       Owner
----------------+-------------------
 fleet_manager | fleet_manager_own
 public        | pg_database_owner
(2 rows)
```

Ilustración 37 - Creación esquema

Listo las tablas:

```
postgres=# select * from pg_tables where schemaname='fleet_manager';
  schemaname   |       tablename        | tableowner | tablespace | hasindexes | hasrules | hastriggers | rowsecurity
---------------+------------------------+------------+------------+------------+----------+-------------+-------------
 fleet_manager | user                   | postgres   |            | t          | f        | f           | f
 fleet_manager | delivery_person        | postgres   |            | t          | f        | t           | f
 fleet_manager | vehicle                | postgres   |            | t          | f        | t           | f
 fleet_manager | maintenance            | postgres   |            | f          | f        | t           | f
 fleet_manager | vehicle_record         | postgres   |            | t          | f        | t           | f
 fleet_manager | vehicle_record_202210  | postgres   |            | t          | f        | t           | f
 fleet_manager | vehicle_record_202211  | postgres   |            | t          | f        | t           | f
 fleet_manager | vehicle_record_202212  | postgres   |            | t          | f        | t           | f
(8 rows)
```

Ilustración 38 - Listado de tablas

Ahora, confirmo que el script ha introducido el *record* de muestra incluido en el script:

```
postgres=# set search_path to fleet_manager;
SET
postgres=# select * from vehicle_record;
 vehicle_id | delivery_person_id |        record_date         | latitude  | longitude
------------+--------------------+----------------------------+-----------+-----------
 1518FPJ    | u12345             | 2022-11-13 10:51:43.337435 | 41.422180 |  2.190590
(1 row)
```

Ilustración 39 - registro de test insertado

Ahora que tengo el volumen de persistencia creado, la instancia de postgreSQL funcionando y el esquema de la aplicación creado, puedo considerar que la base de datos está operativa. En los siguientes puntos, se despliega la aplicación y se comprueba que es posible la conexión de la aplicación a la base de datos.

3.8.2.2 Componente: *Fleet manager receiver*

En esta sección, desplegaré la aplicación en el *cluster* de *google kubernetes* haciendo uso de maven y docker.

3.8.2.2.1 Creación del contenedor

El primer paso es hacer la *build* de la aplicación haciendo uso de Maven con el comando "mvn package". Una vez hecho esto, puedo crear el contenedor docker con el comando "docker build -t eu.gcr.io/fleet-manager-365622/fleet-manager-data-receiver:v0.0.1-SNAPSHOT .".

```
PS C:\Users\Javi\Desktop\Google Drive\UOC\TFG\Software\git\fleet-manager-data-receiver-spring> docker build -t eu.gcr.io/fleet-manager-36562
2/fleet-manager-data-receiver:v0.0.1-SNAPSHOT .
Sending build context to Docker daemon  46.27MB
Step 1/3 : FROM openjdk:17-jdk-slim
 ---> 37cb44321d04
Step 2/3 : COPY target/fleet-manager-data-receiver-spring-0.0.1-SNAPSHOT.jar fleet-manager-data-receiver-spring-0.0.1-SNAPSHOT.jar
 ---> f417f39becb9
Step 3/3 : ENTRYPOINT ["java", "-jar", "/fleet-manager-data-receiver-spring-0.0.1-SNAPSHOT.jar"]
 ---> Running in c23d4ea686d1
Removing intermediate container c23d4ea686d1
 ---> cf9e7f5aeda9
Successfully built cf9e7f5aeda9
Successfully tagged eu.gcr.io/fleet-manager-365622/fleet-manager-data-receiver:v0.0.1-SNAPSHOT
SECURITY WARNING: You are building a Docker image from Windows against a non-Windows Docker host. All files and directories added to build c
ontext will have '-rwxr-xr-x' permissions. It is recommended to double check and reset permissions for sensitive files and directories.
```

Ilustración 40 - Creación del contenedor

El siguiente paso, es subir la imagen del contenedor al registro de GCP.

```
PS C:\Users\Javi\Desktop\Google Drive\UOC\TFG\Software\git\fleet-manager-data-receiver-spring> docker push eu.gcr.io/fleet-manager-365622/fl
eet-manager-data-receiver:v0.0.1-SNAPSHOT
The push refers to repository [eu.gcr.io/fleet-manager-365622/fleet-manager-data-receiver]
5f81f83cc32f: Pushed
6be690267e47: Layer already exists
13a34b6fff78: Layer already exists
9c1b6dd6c1e6: Layer already exists
v0.0.1-SNAPSHOT: digest: sha256:834271fb31a55e6824a458f63ee5fd46c48689c08c5c59b71271407b070cdd4f size: 1165
```

Ilustración 41 - Subida del contenedor

Se puede comprobar que el contenedor se ha subido al registro correctamente.

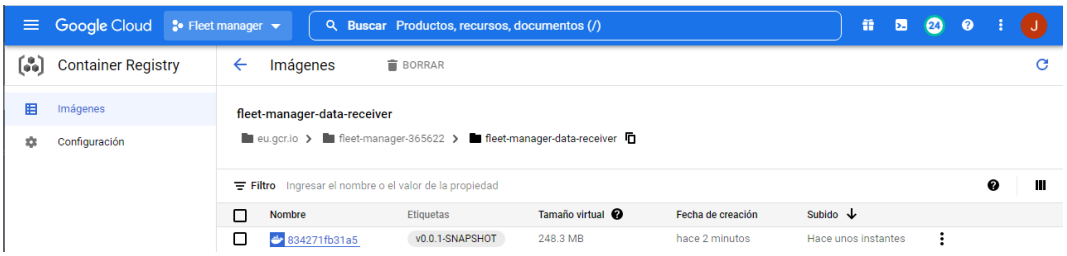


Ilustración 42 – GCP container registry

3.8.2.2.2 Despliegue de la aplicación

Ahora, desplegaré la aplicación en un pod de kubernetes haciendo uso del archivo fmr-deployment.yml, que se puede encontrar en la siguiente ruta del repositorio git:

https://gitlab.com/uoc-tfg-jarubioca/fleet-manager-data-receiver-spring/-/blob/poc/fmr-deployment.yml

```
PS C:\Users\Javi\Desktop\Google Drive\UOC\TFG\Software\git\fleet-manager-data-receiver-spring> kubectl apply -f .\fmr-deployment.yml
deployment.apps/fleet-manager-receiver configured
service/fleet-manager-receiver created
```

Ilustración 43 - Despliegue pod y servicio

Ahora, si ejecuto “kubectl get all” nos listará el pod de la aplicación y el servicio. En este último, indicará la IP externa para acceder al servicio (104.197.121.169).

```
PS C:\Users\Javi\Desktop\Google Drive\UOC\TFG\Software\git\fleet-manager-data-receiver-spring> kubectl get all
NAME                                          READY   STATUS    RESTARTS   AGE
pod/fleet-manager-receiver-75c5647b85-5ph6t   1/1     Running   0          56s
pod/postgres-666f4547b-vzfnn                  1/1     Running   0          9d

NAME                               TYPE           CLUSTER-IP     EXTERNAL-IP       PORT(S)          AGE
service/fleet-manager-receiver     LoadBalancer   10.48.15.28    104.197.121.169   8081:30313/TCP   56s
service/kubernetes                 ClusterIP      10.48.0.1      <none>            443/TCP          9d
service/postgres                   ClusterIP      10.48.4.178    <none>            5432/TCP         9d

NAME                                       READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/fleet-manager-receiver     1/1     1            1           58s
deployment.apps/postgres                   1/1     1            1           9d

NAME                                                DESIRED   CURRENT   READY   AGE
replicaset.apps/fleet-manager-receiver-75c5647b85   1         1         1       58s
replicaset.apps/postgres-666f4547b                  1         1         1       9d
```

Ilustración 44 - pods y servicios

Confirmo que la aplicación se ha ejecutado correctamente revisando el log del pod.

```
PS C:\Users\Javi\Desktop\Google Drive\UOC\TFG\Software\git\fleet-manager-data-receiver-spring> kubectl logs pod/fleet-manager-receiver-75c5647b85-5ph6t

  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v2.7.4)

2022-11-14 16:43:58.148  INFO 1 --- [           main] leetManagerDataReceiverSpringApplication : Starting FleetManagerDataReceiverSpringApplication v0.0.1-SNAPSHOT using Java 17.
0.2 on fleet-manager-receiver-75c5647b85-5ph6t with PID 1 (/fleet-manager-data-receiver-spring-0.0.1-SNAPSHOT.jar started by root in /)
2022-11-14 16:43:58.153  INFO 1 --- [           main] leetManagerDataReceiverSpringApplication : No active profile set, falling back to 1 default profile: "default"
2022-11-14 16:44:09.146  INFO 1 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
2022-11-14 16:44:09.739  INFO 1 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 497 ms. Found 2 JPA repository interf
aces.
2022-11-14 16:44:17.350  INFO 1 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port(s): 8081 (http)
2022-11-14 16:44:17.450  INFO 1 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2022-11-14 16:44:17.451  INFO 1 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet engine: [Apache Tomcat/9.0.65]
2022-11-14 16:44:18.645  INFO 1 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2022-11-14 16:44:18.646  INFO 1 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 19800 ms
2022-11-14 16:44:20.547  INFO 1 --- [           main] o.hibernate.jpa.internal.util.LogHelper  : HHH000204: Processing PersistenceUnitInfo [name: default]
2022-11-14 16:44:21.046  INFO 1 --- [           main] org.hibernate.Version                    : HHH000412: Hibernate ORM core version 5.6.11.Final
2022-11-14 16:44:21.048  INFO 1 --- [           main] org.hibernate.cfg.Environment            : HHH000205: Loaded properties from resource hibernate.properties: {hibernate.bytec
ode.use_reflection_optimizer=false, hibernate.hbm2ddl.extra_physical_table_types=PARTITIONED TABLE}
2022-11-14 16:44:22.750  INFO 1 --- [           main] o.hibernate.annotations.common.Version   : HCANN000001: Hibernate Commons Annotations {5.1.2.Final}
2022-11-14 16:44:24.145  INFO 1 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Starting...
2022-11-14 16:44:26.344  INFO 1 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Start completed.
2022-11-14 16:44:26.641  INFO 1 --- [           main] org.hibernate.dialect.Dialect            : HHH000400: Using dialect: org.hibernate.dialect.PostgreSQLDialect
2022-11-14 16:44:39.338  INFO 1 --- [           main] o.h.e.t.j.p.i.JtaPlatformInitiator       : HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jt
a.platform.internal.NoJtaPlatform]
2022-11-14 16:44:39.348  INFO 1 --- [           main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2022-11-14 16:44:47.241  WARN 1 --- [           main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database queries may be
 performed during view rendering. Explicitly configure spring.jpa.open-in-view to disable this warning
2022-11-14 16:44:52.950  INFO 1 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port(s): 8081 (http) with context path ''
2022-11-14 16:44:55.448  INFO 1 --- [           main] leetManagerDataReceiverSpringApplication : Started FleetManagerDataReceiverSpringApplication in 63.205 seconds (JVM running
for 68.487)
PS C:\Users\Javi\Desktop\Google Drive\UOC\TFG\Software\git\fleet-manager-data-receiver-spring>
```

Ilustración 45 - Log aplicación

3.8.3 Pruebas

Este apartado corresponde a las pruebas de los dos casos de uso utilizados para la prueba de concepto.

3.8.3.1 Caso de uso: CU17 – Enviar registro de ubicación

Para simplificar las pruebas, este caso no tiene implementado la autenticación mediante secreto explicado en el documento de diseño, apartado "2.8.2 Componente: *Fleet manager receiver*".

Los registros en la tabla *vehicle_record* de la base de datos antes registrar una nueva ubicación son:

```
postgres=# select * from vehicle_record;
 vehicle_id | delivery_person_id |        record_date         | latitude  | longitude
------------+--------------------+----------------------------+-----------+-----------
 1518FPJ    | U12345             | 2022-11-14 08:52:25.294081 | 41.422180 |  2.190590
 1518FPJ    | U12345             | 2022-11-14 10:39:05.945    | 50.387466 | 10.168768
(2 rows)
```

Ilustración 46 - Tabla vehicle_record

Procedo a insertar un nuevo registro:

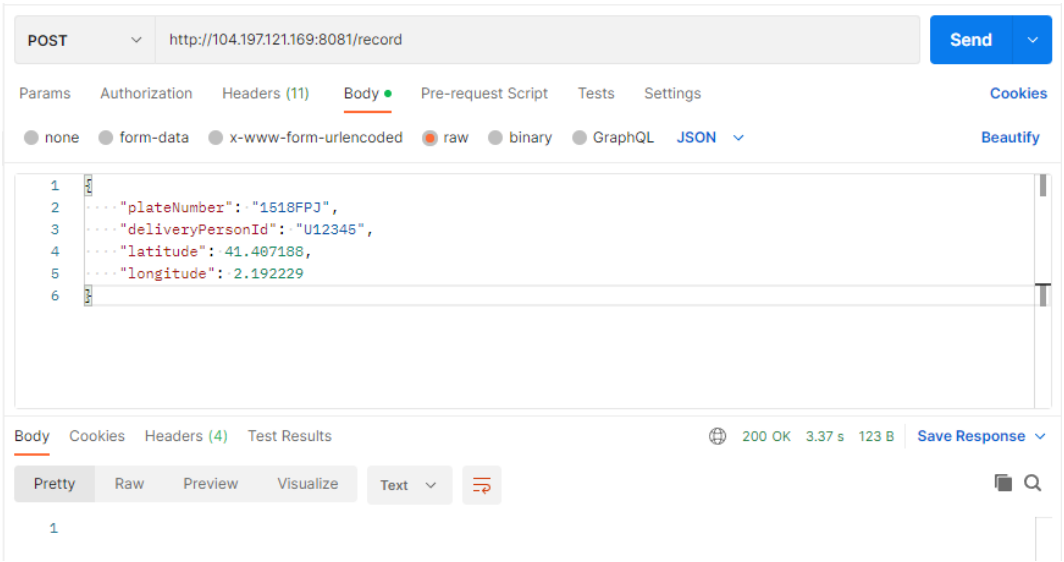


Ilustración 47 - Añadir registro

Según se puede ver en la ilustración 18, el servidor nos ha contestado con un código 200. Reviso la tabla *vehicle_record* de nuevo, ilustración 19.

```
postgres=# select * from vehicle_record order by record_date desc;
 vehicle_id | delivery_person_id |        record_date         | latitude  | longitude
------------+--------------------+----------------------------+-----------+-----------
 1518FPJ    | U12345             | 2022-11-14 16:57:59.248    | 41.407188 |  2.192229
 1518FPJ    | U12345             | 2022-11-14 10:39:05.945    | 50.387466 | 10.168768
 1518FPJ    | U12345             | 2022-11-14 08:52:25.294081 | 41.422180 |  2.190590
(3 rows)
```

Ilustración 48 - tabla vehicle_record después de la petición

2.8.3.2 Caso de uso: CU14 – Marcar vehículo como averiado

Las pruebas de este caso son ligeramente más complicadas ya que he habilitado la autenticación según lo explicado en el documento de diseño, apartado "2.8.2 Componente: Fleet manager receiver".

Primero, compruebo que el vehículo no está marcado como averiado antes de hacer la prueba. En la ilustración 20 se puede ver que el vehículo tiene el valor *booleano false* en la columna *damaged*.

```
postgres=# select * from vehicle;
 plate_number | kilometers | battery_level | active | damaged | last_inspection |           secret
--------------+------------+---------------+--------+---------+-----------------+----------------------------
 1518FPJ      |        100 |            50 | t      | f       | 2022-10-10      | ADMWermss34nsldWERmsmkbndf
(1 row)
```

Ilustración 49 - Tabla vehicle

Ahora, haremos la petición para marcar el vehículo con matrícula 1518FPJ como averiado, pero el servidor debería devolvernos un código *unauthorized*. Ilustración 21.

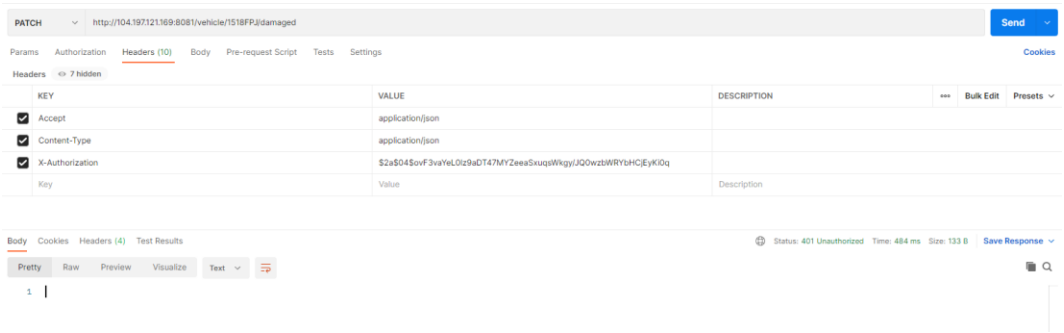


Ilustración 50 - Petición sin autenticar

Si reviso el log de la aplicación, se puede ver cómo el interceptor inyectado en el controlador ha interceptado la petición y la ha rechazado puesto que no ha sido validado por el algoritmo de cifrado. Ilustración 22.

```
2022-11-14 17:10:24.549 ERROR 1 --- [nio-8081-exec-3] u.t.j.f.c.interceptor.CustomInterceptor  : Authorization failed for authorization intercept. Plate number: 1518FPJ
```

Ilustración 51 - Log

Procedo a calcular el valor correcto que debemos enviar en la cabecera para que el servidor autorice nuestra petición. Para ello, usaré una clase auxiliar que he creado que calcula el valor usando el algoritmo que el servidor luego usa para comprobar el valor. Ilustración 22.

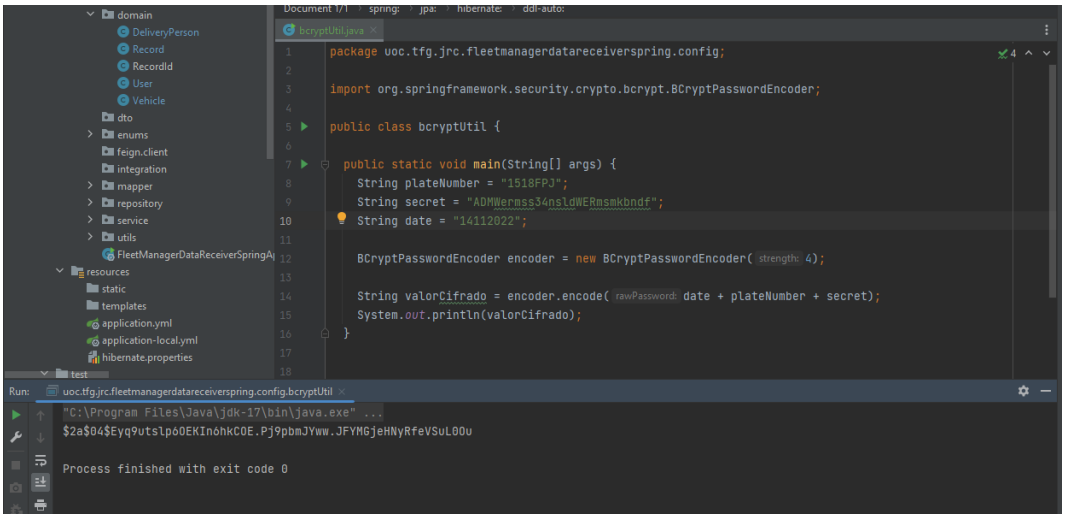


Ilustración 52 - Cálculo del secreto

Cojo el valor de salida del algoritmo y lo incluyo en la cabecera para volver a hacer la petición.

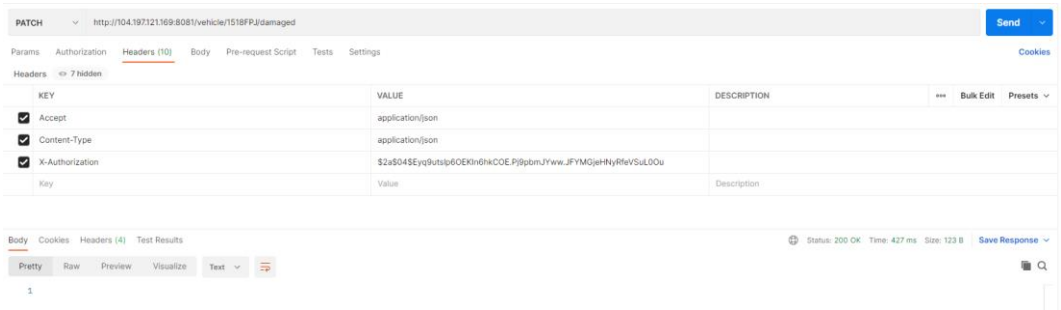


Ilustración 53 - Petición correcta

Ahora el servidor ha devuelto un código 200, esto quiere decir que el valor pasado en la cabecera es correcto y la petición ha sido autenticada.

El log de la aplicación se muestra en la ilustración 25.

```
2022-11-14 17:16:07.337  INFO 1 --- [nio-8081-exec-5] u.t.j.f.service.impl.VehicleServiceImpl  : Marking vehicle 1518FPJ as damaged
2022-11-14 17:16:07.343  INFO 1 --- [nio-8081-exec-5] u.t.j.f.service.impl.VehicleServiceImpl  : Car with plate 1518FPJ was marked as damaged
```

Ilustración 54 - Log aplicación

Reviso la base de datos para comprobar que el vehículo se ha marcado como averiado. Ilustración 26.

```
postgres=# select * from vehicle;
 plate_number | kilometers | battery_level | active | damaged | last_inspection |          secret
--------------+------------+---------------+--------+---------+-----------------+--------------------------
 1518FPJ      |        100 |            50 | t      | t       | 2022-10-10      | ADMWermss34nsldWERmsmkbndf
(1 row)
```

Ilustración 55 - Tabla vehicle

4. Implementación

4.1 Introducción

En esta sección se especifica todo el proceso de implementación en base a lo estudiado en las fases de análisis y diseño. Además de información directamente relacionada con la implementación, también se detallan conceptos, estándares y códigos de buenas prácticas que ayudarán a construir una aplicación consistente, eficiente, de código inteligible y abierta al desarrollo concurrente por parte de múltiples desarrolladores/as.

4.2 Convenios generales

En este apartado se detallarán las buenas prácticas y acuerdos que se llevarán a cabo, durante la fase de implementación, con el propósito de generar un código mantenible, limpio y que esté abierto a la modificación y extensión por otros/as desarrolladores/as.

4.2.1 Guía de estilo

Las guías de estilo ayudan a aplicar una estructura heterogénea del código en equipos de varios/as desarrolladores/as, proporcionando consistencia de estilo y facilitando el entendimiento del código.

La guía que se usará para el desarrollo de este proyecto es de código abierto y proporcionada por Google y está disponible para los IDE Eclipse e IntelliJ IDE. Yo usaré la versión de IntelliJ IDEA.

Enlace al recurso: https://github.com/google/styleguide/blob/gh-pages/intellij-java-google-style.xml

En la ilustración 1 se muestra la aplicación de la hoja de estilo en el entorno de desarrollo.

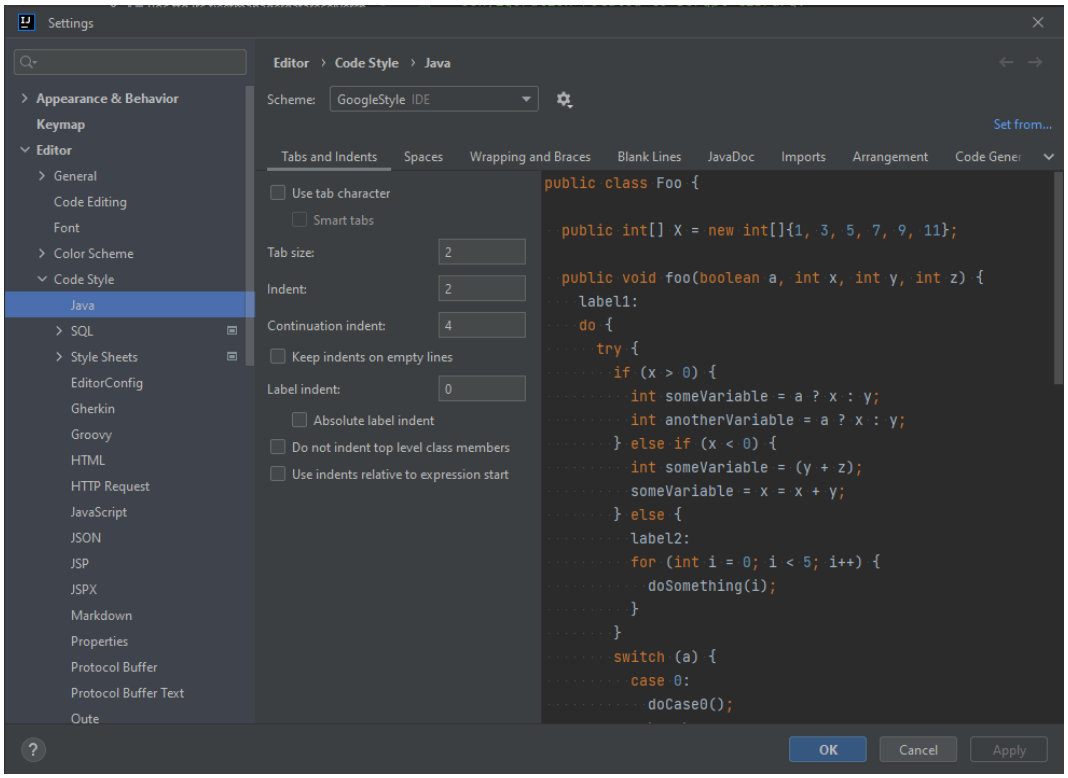


Ilustración 56 - Aplicación de la guía de estilo en el IDE

4.2.2 Principios SOLID

Los principios SOLID consisten en un conjunto de principios a seguir para garantizar que nuestro código generado cumple con los siguientes objetivos:

- Construir un código limpio, eficaz, de gran robustez y estable.
- Flexibilidad. El código está abierto a ser modificado, reutilizado y mantenible.
- Permite la escalabilidad, es decir, puede ser ampliado con nuevas funcionalidades.

Los cinco conceptos aplicados al código que nos ayudarán a conseguir los objetivos anteriores son:

- Principio de única responsabilidad (*Single responsibility*): una clase solo debe tener una sola responsabilidad. Ejemplo: si la responsabilidad de un POJO (*Plain Old Java Object*) de la capa de dominio, por ejemplo, la clase Vehicle.java, es representar un objeto del mundo real, añadir métodos de impresión en log o cualquier otra funcionalidad que vaya más allá de representar ese objeto como entidad, estaríamos incumpliendo este principio.
- Principio abierto/cerrado (*Open/close*): las clases deben estar abiertas a la extensión, pero cerradas a la modificación. Ejemplo: si en una versión posterior de la aplicación, se quiere añadir más atributos a la clase vehículo, pero conservando la compatibilidad del código existente, se podría considerar la herencia y crear una nueva clase que herede de vehículo.

- Substitución de *Liskov*: si una clase A es subtipo de una clase B, se debería poder sustituir una instancia de B por una de A sin influir en el funcionamiento de la aplicación.
- Segregación de interfaces: en el caso de que alguna interfaz empiece a disponer de un número elevado de métodos, debe considerarse la división en múltiples interfaces.
- Inversión de dependencias: especialmente enfocado en propiciar el desacoplamiento en módulos de *software*. Consiste en hacer depender todos los módulos en abstracciones en vez de en módulos de más alto nivel.

4.2.3 Patrones de diseño

Durante la implementación, se ha tenido muy en cuenta la necesidad e importancia de utilizar patrones de diseño en cualquier tipo de proyecto. Algunos de los múltiples patrones que se han utilizado son:

- Patrón *singleton*: se ha utilizado para garantizar que, durante la ejecución de la aplicación, únicamente se instanciará (y existirá) un objeto de la clase JwtSecret. De esta manera, se garantiza que la *secret key* utilizada por el programa será siempre la misma.
  Enlace a la clase específica en el repositorio:
  https://gitlab.com/uoc-tfg-jarubioca/fleet-manager/-/blob/develop/src/main/java/uoc/tfg/jrc/fleetmanagerappspring/config/security/JwtSecret.java
- Patrón DTO (*Data Transfer Object*): serializan las entidades del dominio y su principal cometido es el paso de entidades entre capas (o bien en controladores). Enlace al repositorio: https://gitlab.com/uoc-tfg-jarubioca/fleet-manager/-/tree/develop/src/main/java/uoc/tfg/jrc/fleetmanagerappspring/controller/dto
- Patrón *mapper*: se utiliza para el mapeado entre entidades del dominio y DTOs. Enlace al repositorio: https://gitlab.com/uoc-tfg-jarubioca/fleet-manager/-/blob/develop/src/main/java/uoc/tfg/jrc/fleetmanagerappspring/mapper/UserMapper.java
- Patrón constructor (*builder*): se utiliza para delegar la construcción de objetos complejos. Enlace al repositorio: https://gitlab.com/uoc-tfg-jarubioca/fleet-manager/-/blob/develop/src/main/java/uoc/tfg/jrc/fleetmanagerappspring/domain/Record.java

4.2.4 Repositorio

El código de los dos componentes, juntamente al script de la base de datos y el resto de los archivos que forman parte del proyecto pueden encontrarse en el repositorio git de gitlab, que es de acceso público.

Enlace directo al componente *fleet manager receiver* en el repositorio:
https://gitlab.com/uoc-tfg-jarubioca/fleet-manager-data-receiver-spring

Enlace directo al componente *fleet manager app* en el repositorio:

https://gitlab.com/uoc-tfg-jarubioca/fleet-manager

4.2.5 Patrón de nombre en *tests* unitarios: *given-when-then*

Para aportar un extra de información y claridad, y conseguir unos *tests* con nombre más descriptivos, se considera usar el patrón *given-when-then*. Se puede encontrar más información en la siguiente dirección: https://dzone.com/articles/a-new-approach-to-given-when-then

4.2.6 Reducción de código *boilerplate*: *lombok*

La librería Lombok (https://projectlombok.org/) proporciona herramientas, concretamente mediante el uso de anotaciones, que generarán código considerado como *boilerplate* (constructores, *setters*, *getters*, etc.) en tiempo de compilación.

Prácticamente en la totalidad de las clases de ambos componentes que conforman el proyecto, pueden encontrarse anotaciones de esta librería.

4.3 Despliegue en entorno *cloud*

Según se requería en la fase de análisis y, posteriormente, se tuvo en cuenta en la fase de diseño, la aplicación debe estar desplegada en la nube. Para demostrar la viabilidad de este requisito, se realizó la prueba de concepto.
Para la fase de implementación, se aprovecharán los siguientes elementos para lograr la implementación de la aplicación:

- *Cluster* de GKE (*Google Kubernetes Engine*).
- Volumen de persistencia.
- Instancia (*pod*) de PosgreSQL.

No obstante, puesto que en la fase de implementación se ha tenido que actualizar el diseño de la base de datos para incorporar las entidades y relaciones necesarias para dotar a la aplicación de autorización y autenticación, se procede al borrado del esquema y los datos contenidos durante la fase de prueba de concepto para proceder a la creación del modelo final, el cual hará uso la aplicación en su versión *release*.

4.3.1 Prerrequisitos

Para aprobar el despliegue de una nueva versión a entorno productivo, deben cumplirse los siguientes requisitos:

- Porcentaje de código cubierto por test unitario: el código debe estar cubierto en un 85% mínimo. Se comprobará este cumplimiento mediante la herramienta jacoco (https://www.jacoco.org/jacoco/), que está incluido en todos los componentes a nivel de dependencia maven.
- Calidad del código: se medirá según la herramienta SonarQube según parámetros estándar. Se considera que el código es correcto y de calidad cuando la herramienta reporta que los bugs, vulnerabilidades, errores de seguridad y *code smells* son todos cero.
- Generación de versión: se seguirá un esquema de versionado según el esquema XX.YY.ZZ-VERSION, donde:
    - XX: especifica el número de versión mayor. Se incrementa cuando el componente recibe cambios que lo hacen incompatible con integraciones anteriores.
    - YY: especifica el número de versión menor. Se incrementa cuando el componente recibe cambios de funcionalidades y/o servicios sin provocar alteraciones o malfuncionamientos con integraciones ya existentes.
    - ZZ: se incrementa cuando se corrige un malfuncionamiento de funcionalidades y/o servicios incluidos en incrementos de versiones mayores o menores.
    - VERSION: indica el objetivo de la versión. Sus posibles valores son:
        - *SNAPSHOT*: versión de desarrollo, no apta para entorno productivo.
        - *RELEASE*: versión de producción.
        - *RC (RELEASE CANDIDATE)*: versión candidata a ser productiva.
- Empaquetado: se utilizará la herramienta docker para el empaquetado del componente y, posteriormente, se registrará el contenedor en el registro de contenedores del *cluster* de kubernetes.

Request for change (RFC): consiste en un documento en el que se propone la aplicación de un cambio, en este caso aplicado a un producto de *software*. En el documento se especifica qué cambios se harán, cómo, el criterio de aceptación, las personas involucradas y sus roles, además de un plan para dar marcha atrás y volver a la versión del producto estable anterior en caso de que la nueva versión no cumpla con los requerimientos.

4.3.2 Componente: Fleet manager receiver

En este subapartado se detalla el proceso de validación de los prerrequisitos por tal de desplegar versión de la aplicación en entorno productivo. Así como su proceso de despliegue.

4.3.2.1 Testeo unitario: cobertura

Para comprobar que el código cumple el requisito de cobertura de código, debe utilizarse el comando "*mvn -U clean install*" desde la herramienta de construcción *maven*. Con este comando, maven eliminará el código previamente construido y lo volverá a construir, actualizando, además, las dependencias. Al encontrar en el pom la dependencia de jacoco, calculará el porcentaje de cobertura.

```
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 16, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO]
[INFO] --- jacoco-maven-plugin:0.8.8:report (report) @ fleet-manager-data-receiver-spring ---
[INFO] Loading execution data file C:\Users\Javi\Desktop\Google Drive\UOC\TFG\Software\git\fleet-manager-data-receiver-spring\target\jacoco.exec
[INFO] Analyzed bundle 'fleet-manager-data-receiver-spring' with 10 classes
[INFO]
[INFO] --- maven-jar-plugin:3.2.2:jar (default-jar) @ fleet-manager-data-receiver-spring ---
[INFO] Building jar: C:\Users\Javi\Desktop\Google Drive\UOC\TFG\Software\git\fleet-manager-data-receiver-spring\target\fleet-manager-data-receiver-spring-0.0.1-SNAPSHOT.jar
[INFO]
[INFO] --- spring-boot-maven-plugin:2.7.4:repackage (repackage) @ fleet-manager-data-receiver-spring ---
[INFO] Replacing main artifact with repackaged archive
[INFO]
[INFO] --- maven-install-plugin:2.5.2:install (default-install) @ fleet-manager-data-receiver-spring ---
[INFO] Installing C:\Users\Javi\Desktop\Google Drive\UOC\TFG\Software\git\fleet-manager-data-receiver-spring\target\fleet-manager-data-receiver-spring-0.0.1-SNAPSHOT.jar to C:\Users\Javi\.m2\repository\uoc\tfg\jrc\fleet-manager-data-receiver-spring\0.0.1-SNAPSHOT\fleet-manager-data-receiver-spring-0.0.1-SNAPSHOT.jar
[INFO] Installing C:\Users\Javi\Desktop\Google Drive\UOC\TFG\Software\git\fleet-manager-data-receiver-spring\pom.xml to C:\Users\Javi\.m2\repository\uoc\tfg\jrc\fleet-manager-data-receiver-spring\0.0.1-SNAPSHOT\fleet-manager-data-receiver-spring-0.0.1-SNAPSHOT.pom
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  19.504 s
[INFO] Finished at: 2022-12-22T18:53:12+01:00
[INFO] ------------------------------------------------------------------------
```

Ilustración 57 - Resultado ejecución comando maven

En la carpeta *target* del proyecto, la herramienta de jacoco habrá creado una serie de archivos con extensión .html donde podremos consultar el dato de cobertura.

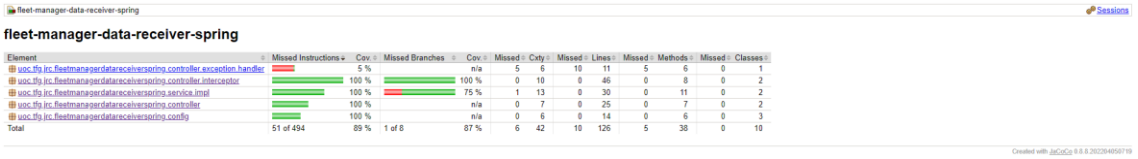


Ilustración 58 - Reporte jacoco

Como se puede comprobar en la ilustración 3, el porcentaje es del 89%, por lo que se cumple el requisito de cobertura de pruebas.

4.3.2.2 Calidad del código: SonarQube

El primer análisis del código, cuyo resultado se muestra en la ilustración 4, muestra numerosos *code smells*. Antes de generar versión, deben subsanarse todos estos avisos.

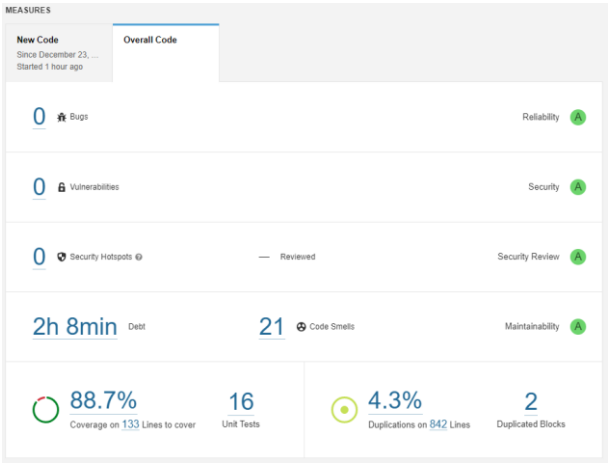


Ilustración 59 - Primer análisis SonarQube

Después de subsanar los errores reportados, se vuelve a analizar el código y, esta vez, el proyecto marca cero en todas las métricas (ilustración 5).

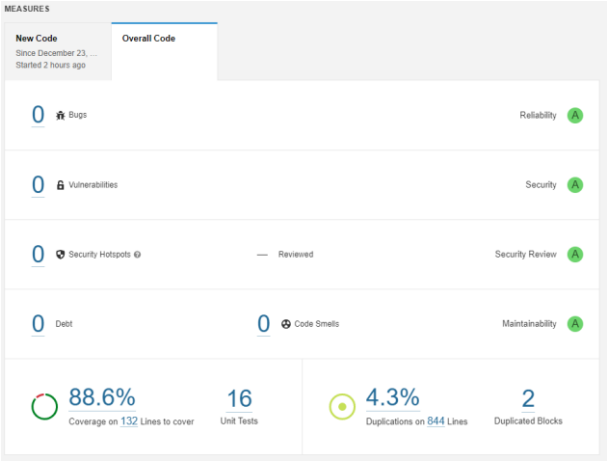


Ilustración 60 - Segundo análisis SonarQube

4.3.2.3 Generación de versión

Puesto que la versión que se generará esta vez será la que pondrá en marcha el servicio por primera vez, es lógico considerar que deberá ser la 1.0.0-RELEASE. Para llevar a cabo la generación de versión, se siguen los siguientes pasos:

- Se modifica la versión en el pom según la ilustración 6.

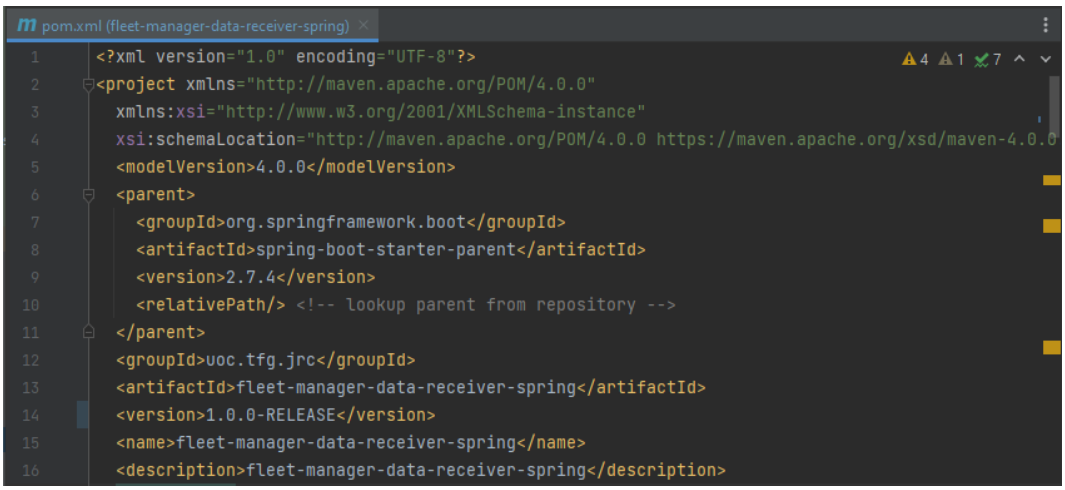


Ilustración 61 - Incremento versión pom

- Creación de un *tag* que especificará sobre qué *commit* específico se ha generado una versión. Ilustración 7.

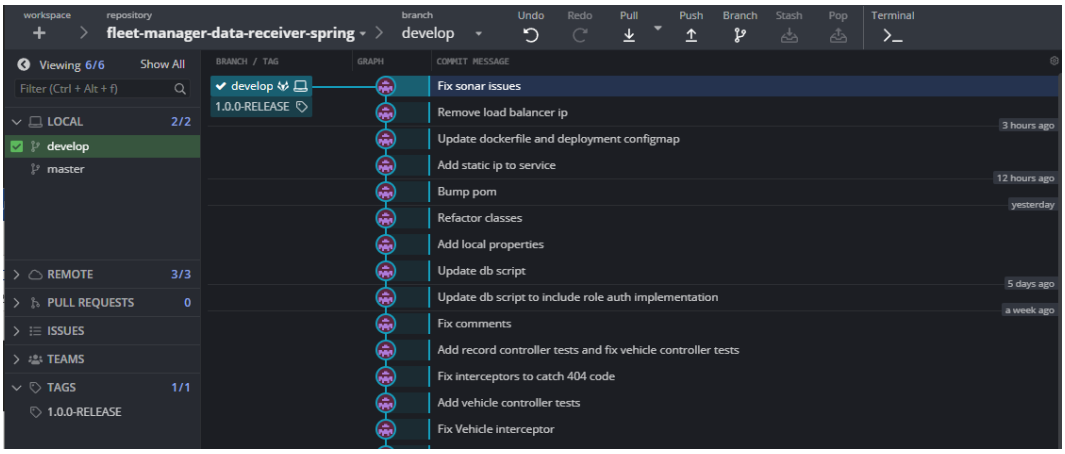


Ilustración 62 - Creación tag 1.0.0-RELEASE

- Fusión (*merge*) a la rama master, que contendrá únicamente código destinado a producción. Ilustración 8.

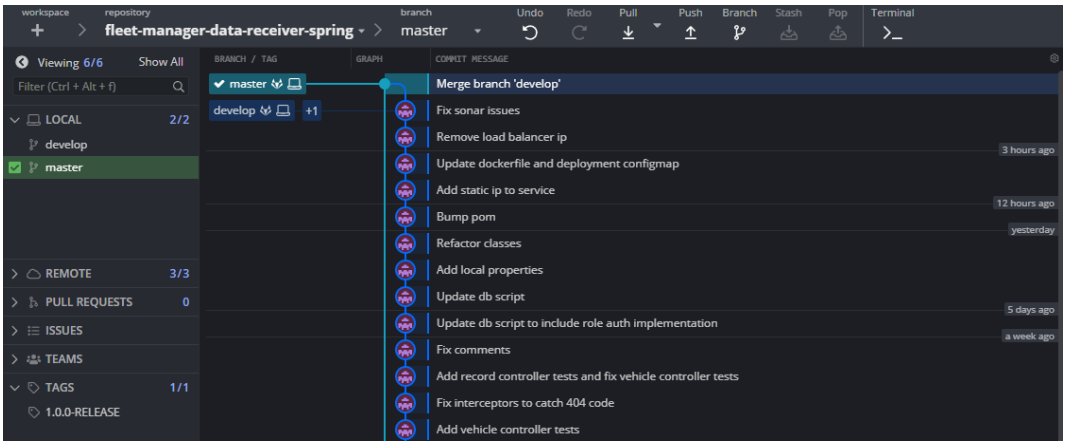


Ilustración 63 - Merge a la rama master

4.3.2.4 Empaquetado y subida al registro de contenedores *Google Container Registry* (GCR)

El primer paso es realizar el empaquetado del componente, que se hace con la herramienta docker (ilustración 9).

```
PS C:\Users\Javi\Desktop\Google Drive\UOC\TFG\Software\git\fleet-manager-data-receiver-spring> docker build -t eu.gcr.io/fleet-manager-365622/fleet-manager-data-receiver:v1.0.0-RELEASE

Sending build context to Docker daemon  47.49MB
Step 1/3 : FROM openjdk:17-jdk-slim
 ---> 37cb44321d04
Step 2/3 : COPY target/fleet-manager-data-receiver-spring-1.0.0-RELEASE.jar fleet-manager-data-receiver-spring-1.0.0-RELEASE.jar
 ---> b0938ddf87c3
Step 3/3 : ENTRYPOINT ["java", "-jar", "/fleet-manager-data-receiver-spring-1.0.0-RELEASE.jar"]
 ---> Running in 8b86b027fbcf
Removing intermediate container 8b86b027fbcf
 ---> 2edc7894cbe8
Successfully built 2edc7894cbe8
Successfully tagged eu.gcr.io/fleet-manager-365622/fleet-manager-data-receiver:v1.0.0-RELEASE
```

Ilustración 64 - Empaquetado del componente

Una vez hecho el empaquetado, el siguiente paso es subir el container creado al repositorio de google container registry (ilustración 10).

```
PS C:\Users\Javi\Desktop\Google Drive\UOC\TFG\Software\git\fleet-manager-data-receiver-spring> docker push eu.gcr.io/fleet-manager-365622/fleet-manager-data-receiver:v1.0.0-RELEASE
The push refers to repository [eu.gcr.io/fleet-manager-365622/fleet-manager-data-receiver]
2dd19cfe01cd: Pushed
6be690267e47: Layer already exists
13a34b6fff78: Layer already exists
9c1b6dd6c1e6: Layer already exists
v1.0.0-RELEASE: digest: sha256:caa95fdc4a5d31ffff373a612da324a174bbf14730498aff0547fff81fc71b0f size: 1165
```

Ilustración 65 - Subida del container al registro

En la ilustración 11 se puede comprobar, en la interfaz de GCP, como el container está disponible y accesible por el cluster de kubernetes.

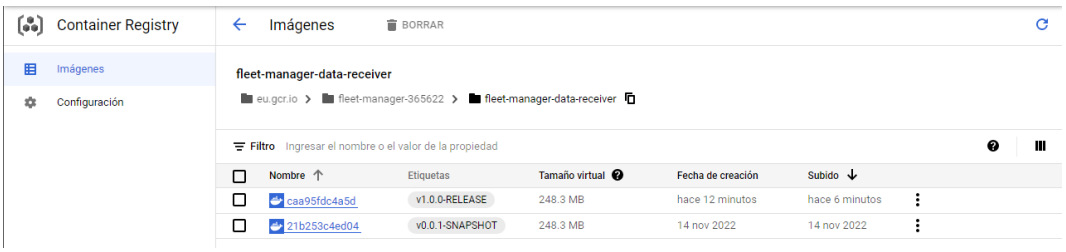


Ilustración 66 - Google container registry

4.3.2.5 *Request for changes* (RFC)

A continuación, el contenido de la RFC.

a) Datos generales

Producto	Fleet manager receiver
Versión	1.0.0
Funcionalidades (casos de uso)	Envío de registros vehículos. Envío de estado vehículo averiado. Envío de estado de vehículos.
Responsable	Javier Rubio
Fecha solicitada	23/12/2022 – 10:00 GMT+1
Duración esperada	30min
Autoriza	Javier Rubio
Entorno	Producción
Estado	Satisfactorio

b) Equipo de despliegue

Nombre	Rol	Contacto	Responsabilidades
Javier Rubio	Desarrollador	jarubioca@uoc.edu	Incremento de versionado, despliegue, pruebas de aceptación.

c) Descripción de cambios

Primera puesta en marcha del servicio.

d) Impacto

El impacto es general, ya que coincide con la puesta en marcha del servicio en su primera versión.

e) Descripción del despliegue

1. Generación de versión y subida a la rama *master* de la versión *release* generada.

2. Empaquetado de la aplicación y subida al registro de contenedores de GCP.
3. Despliegue en el *cluster* de kubernetes del proyecto GCP.

f) Rollback

En caso de inestabilidad o malfuncionamiento del servicio, al ser primera versión del servicio y no haber una versión estable a la que volver, deberá de cerrarse el servicio inmediatamente.

g) Pruebas de aceptación

Las pruebas de aceptación se encuentran en el punto 3.2.3, pruebas de aceptación, de este documento.

4.3.2.6 Despliegue

Para realizar el despliegue, nos conectaremos al *cluster* de kubernetes y aplicaremos el *configmap* de la aplicación que se encuentra en el repositorio. Enlace: https://gitlab.com/uoc-tfg-jarubioca/fleet-manager-data-receiver-spring/-/blob/master/fmr-deployment.yml

Ejecutamos el comando para ejecutar el despliegue del servicio según la siguiente ilustración.

```
PS C:\Users\Javi\Desktop\Google Drive\UOC\TFG\Software\git\fleet-manager-data-receiver-spring> kubectl apply -f .\fmr-deployment.yml
deployment.apps/fleet-manager-receiver created
service/fleet-manager-receiver created
```

Ilustración 67 - Ejecución del despliegue

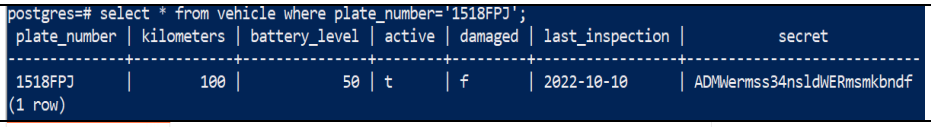
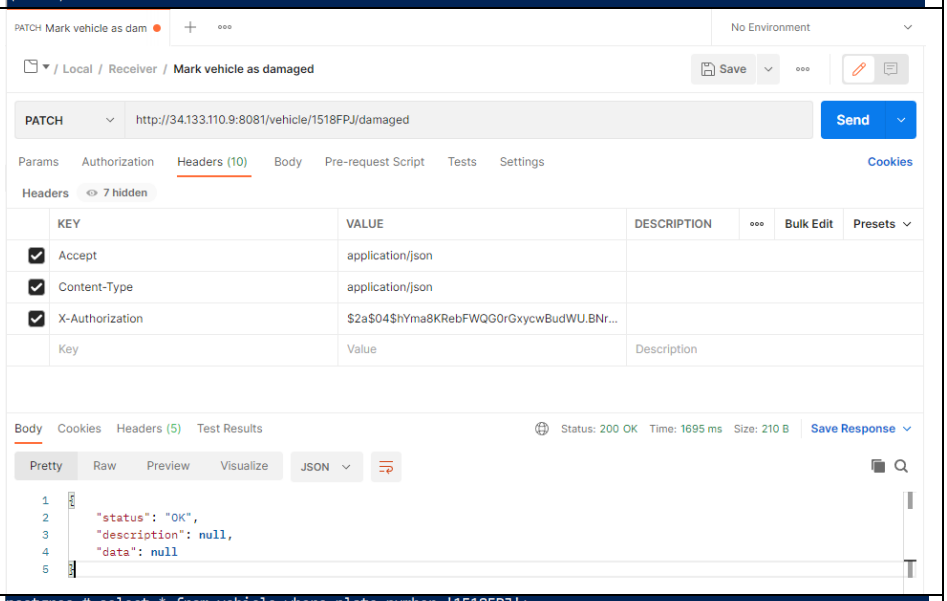
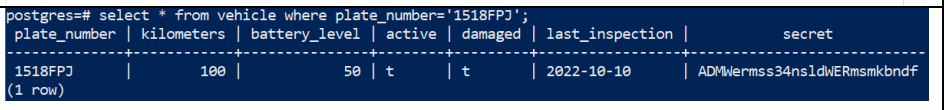
Ahora, se comprueba que se ha creado el despliegue, *pods* de la aplicación y servicio según el *configmap* ejecutado. En la siguiente ilustración se muestra el estado de del *cluster* donde pueden verse los *pods* y servicios creados.

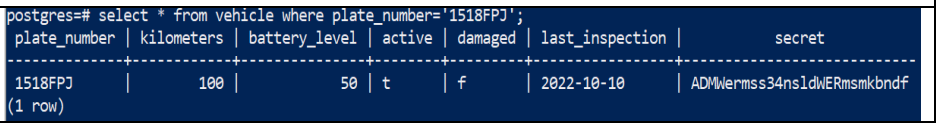
```
PS C:\Users\Javi\Desktop\Google Drive\UOC\TFG\Software\git\fleet-manager-data-receiver-spring> kubectl get all
NAME                                          READY   STATUS    RESTARTS   AGE
pod/fleet-manager-receiver-6c5c4d5fbb-zt5dq   1/1     Running   0          41s
pod/postgres-666f4547b-tglxf                  1/1     Running   0          11d

NAME                               TYPE           CLUSTER-IP     EXTERNAL-IP    PORT(S)          AGE
service/fleet-manager-receiver     LoadBalancer   10.48.1.134    34.133.110.9   8081:31896/TCP   42s
service/kubernetes                 ClusterIP      10.48.0.1      <none>         443/TCP          48d
service/postgres                   ClusterIP      10.48.4.178    <none>         5432/TCP         48d

NAME                                       READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/fleet-manager-receiver     1/1     1            1           42s
deployment.apps/postgres                   1/1     1            1           48d

NAME                                                DESIRED   CURRENT   READY   AGE
replicaset.apps/fleet-manager-receiver-6c5c4d5fbb   1         1         1       42s
replicaset.apps/postgres-666f4547b                  1         1         1       48d
```

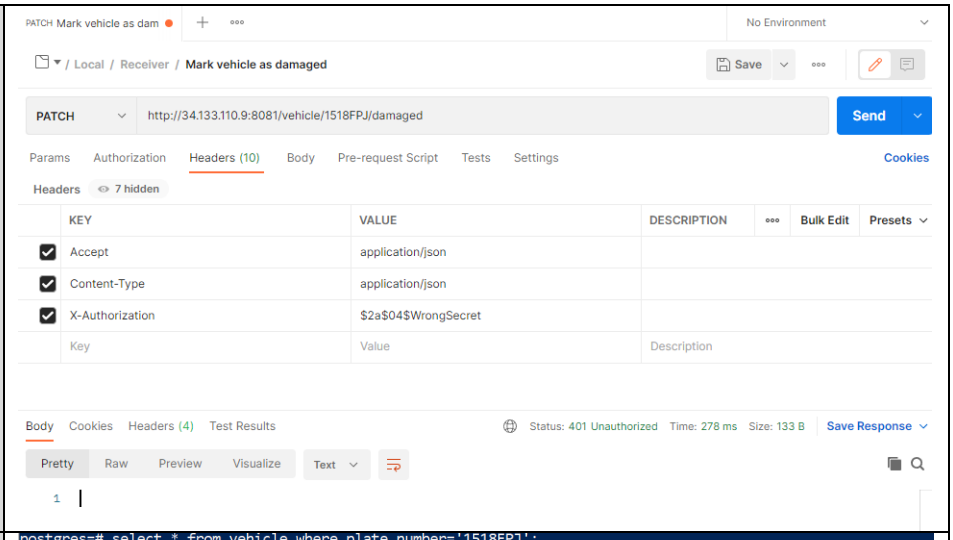
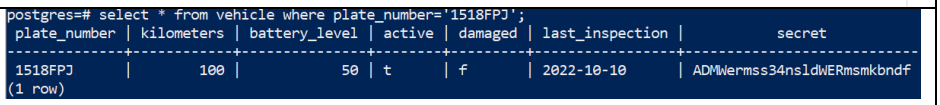
Ilustración 68 - Cluster kubernetes

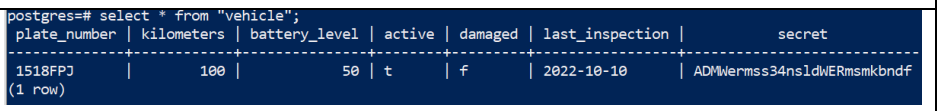
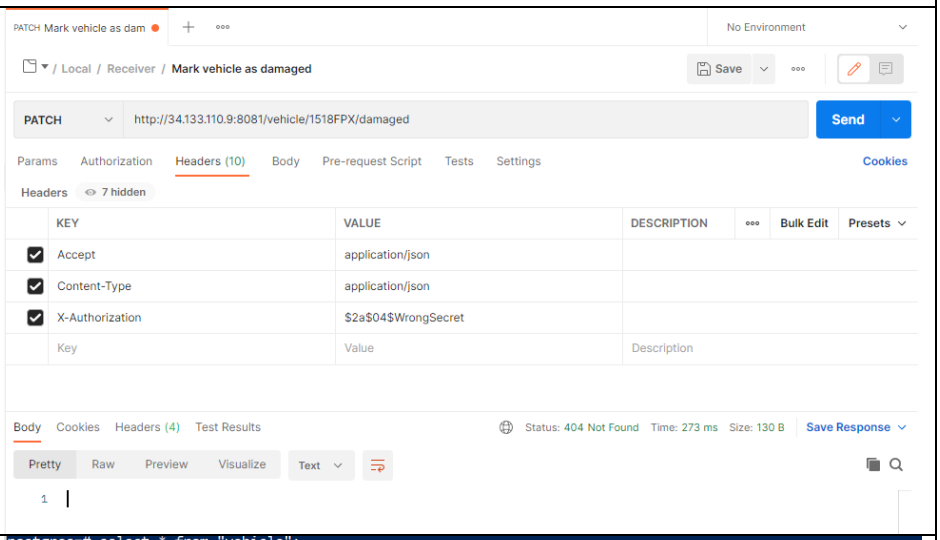
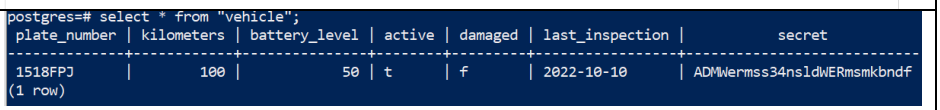
4.3.2.7 Pruebas de aceptación

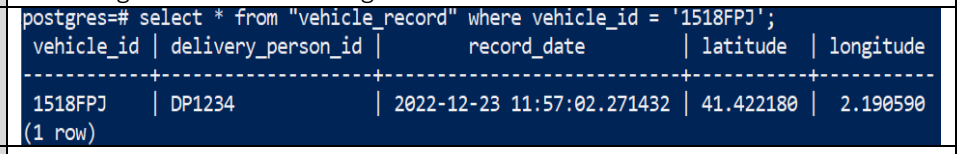
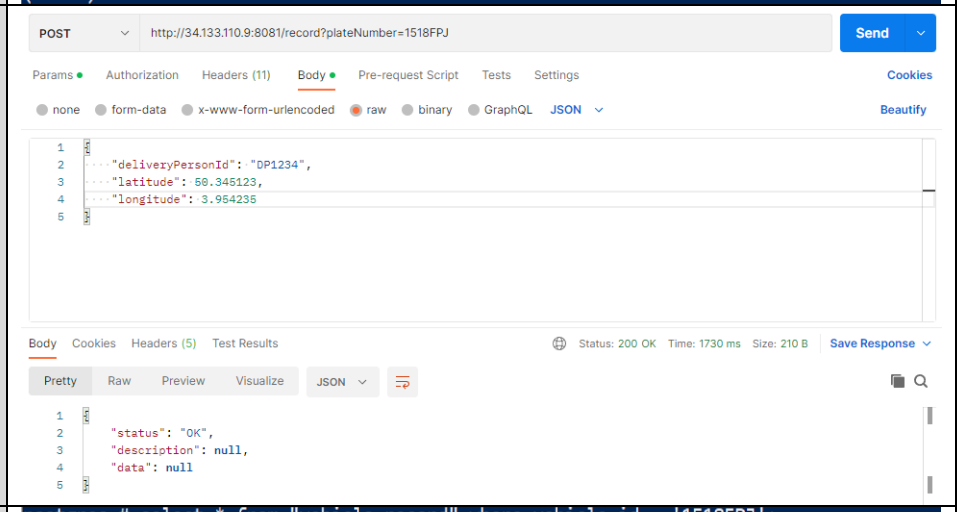
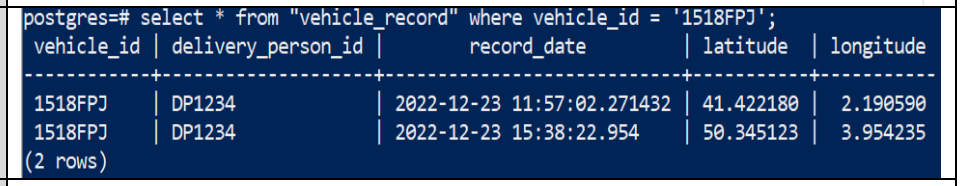
A continuación, se detallan todas las pruebas realizadas por tal de validar el mínimo entregable.

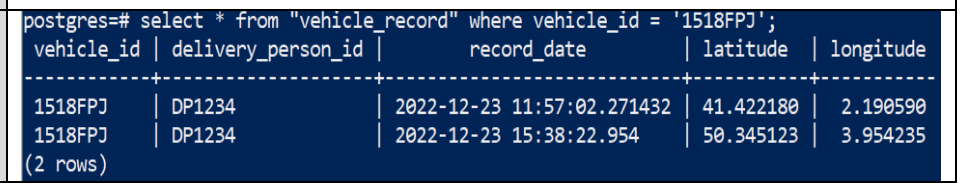
Descripción	Marcar vehículo 1518FPJ como averiado.
Precondición	Se envía como *header* "X-Authorization" el secreto encriptado del vehículo.
Resultado esperado	Respuesta código HTTP 200. Vehículo 1518FPJ marcado como averiado.
Estado BDD antes de la operación	
Petición	
Estado BDD después de la operación	
Resultado satisfactorio	Sí

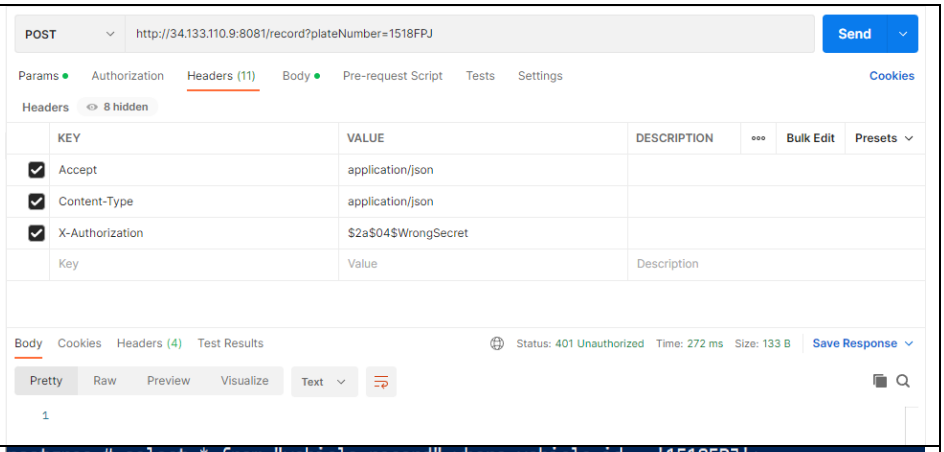
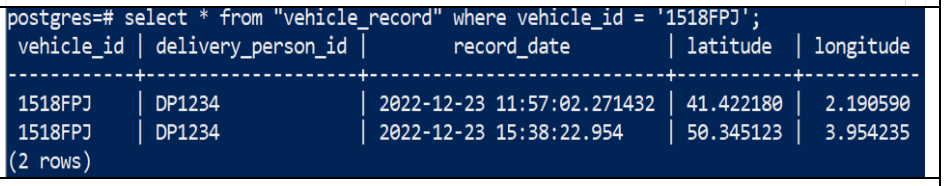
Descripción	Marcar vehículo 1518FPJ como averiado.
Precondición	Se envía como *header* "X-Authorization" un valor que no corresponde con el secreto encriptado del vehículo.
Resultado esperado	Respuesta código HTTP 401.
Estado BDD antes de la operación	

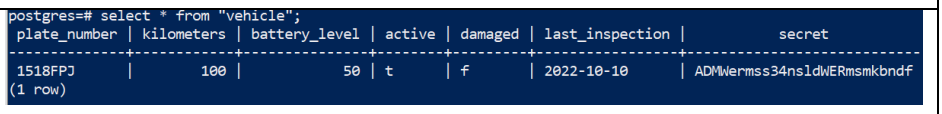
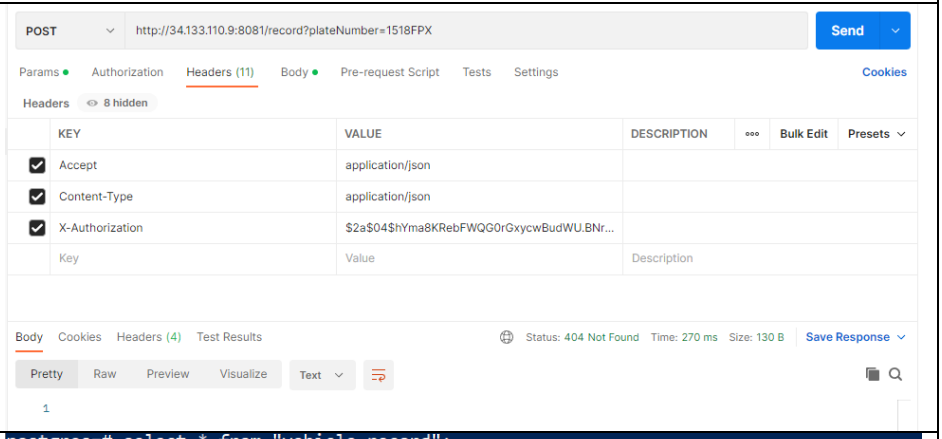
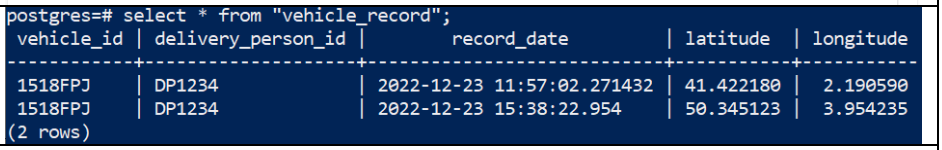
Petición	
Estado BDD después de la operación	
Resultado satisfactorio	Sí

Descripción	Marcar vehículo no existente como averiado.
Precondición	-
Resultado esperado	Respuesta código HTTP 404.
Estado BDD antes de la operación	
Petición	
Estado BDD después de la operación	
Resultado satisfactorio	Sí

Descripción	Insertar registro para vehículo 1518FPJ.
Precondición	Se envía como *header* "X-Authorization" el secreto encriptado del vehículo.
Resultado esperado	Respuesta código HTTP 200. Se ha registrado un nuevo registro.
Estado BDD antes de la operación	
Petición	
Estado BDD después de la operación	
Resultado satisfactorio	Sí

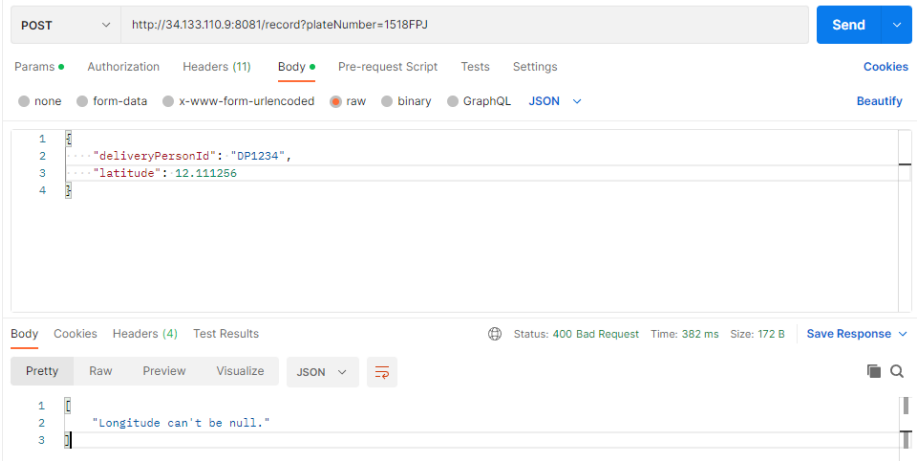
Descripción	Insertar registro para vehículo 1518FPJ.
Precondición	Se envía como *header* "X-Authorization" un valor incorrecto de secreto encriptado del vehículo.
Resultado esperado	Respuesta código HTTP 401.
Estado BDD antes de la operación	

Petición	
Estado BDD después de la operación	
Resultado satisfactorio	Sí

Descripción	Insertar registro para vehículo 1518FPX.
Precondición	-
Resultado esperado	Respuesta código HTTP 404.
Estado BDD antes de la operación	
Petición	
Estado BDD después de la operación	
Resultado satisfactorio	Sí

Descripción	Insertar registro para vehículo 1518FPJ.
Precondición	Se envía como *header* "X-Authorization" el secreto encriptado del vehículo. No obstante, no se informa la longitud.
Resultado esperado	Respuesta código HTTP 400. El sistema avisa de que la longitud no puede ser nula.

Estado BDD antes de la operación	`postgres=# select * from "vehicle_record" where vehicle_id = '1518FPJ';`
	` vehicle_id | delivery_person_id |  record_date  | latitude | longitude`
	`---+---+---+---+---`
	` 1518FPJ | DP1234 | 2022-12-23 11:57:02.271432 | 41.422180 | 2.190590`
	` 1518FPJ | DP1234 | 2022-12-23 15:38:22.954 | 50.345123 | 3.954235`
	`(2 rows)`

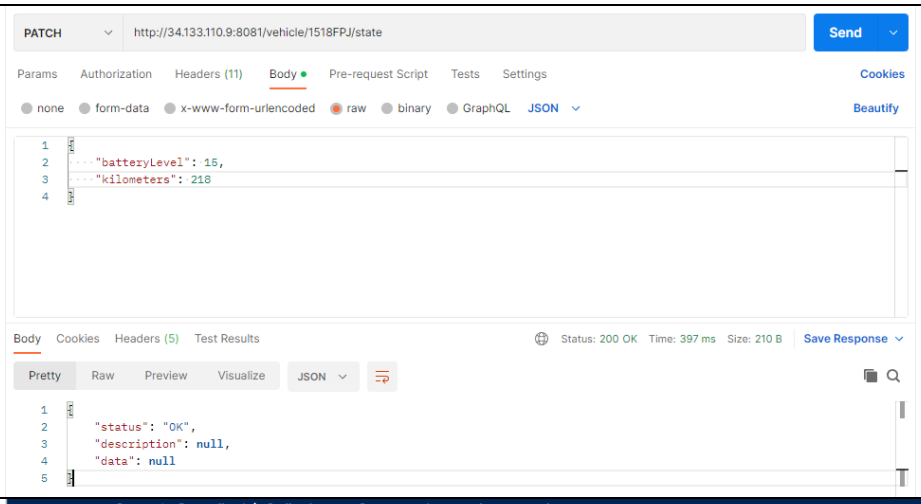
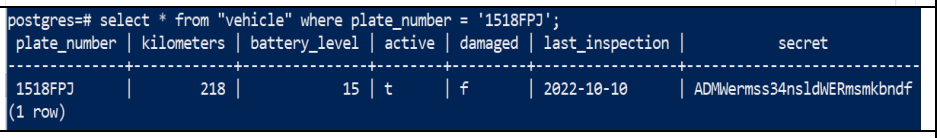
Petición	

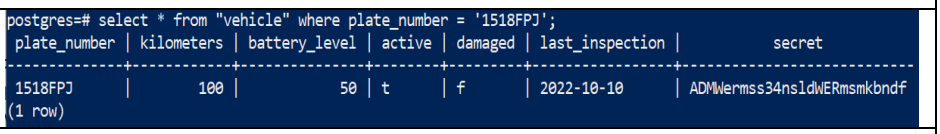
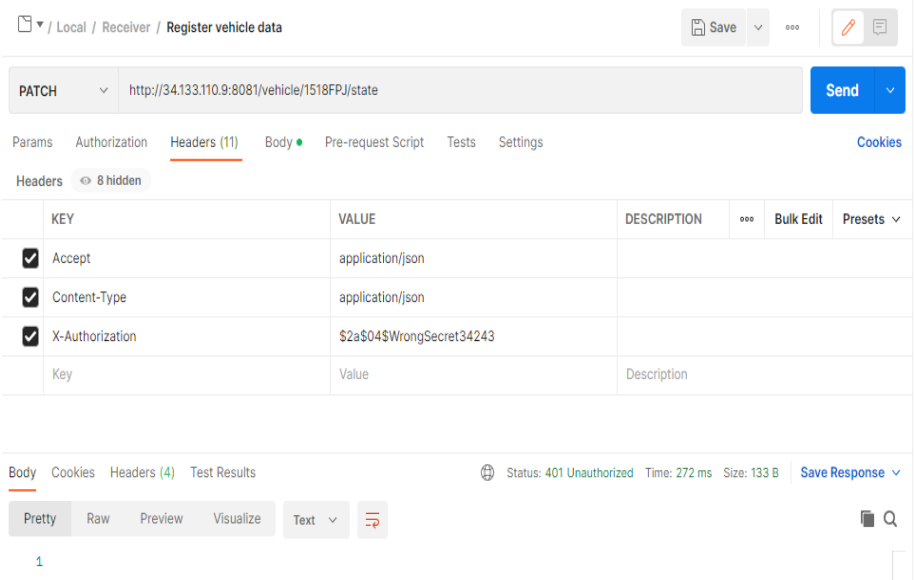
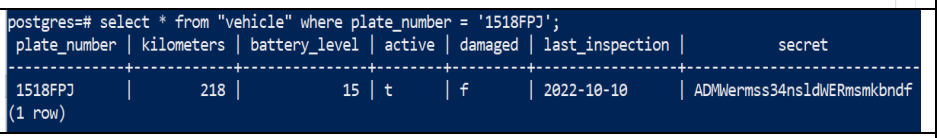
Estado BDD después de la operación	`postgres=# select * from "vehicle_record" where vehicle_id = '1518FPJ';`
	` vehicle_id | delivery_person_id |  record_date  | latitude | longitude`
	`---+---+---+---+---`
	` 1518FPJ | DP1234 | 2022-12-23 11:57:02.271432 | 41.422180 | 2.190590`
	` 1518FPJ | DP1234 | 2022-12-23 15:38:22.954 | 50.345123 | 3.954235`
	`(2 rows)`

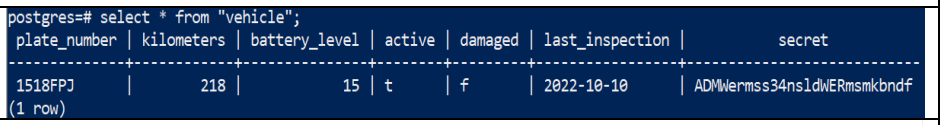
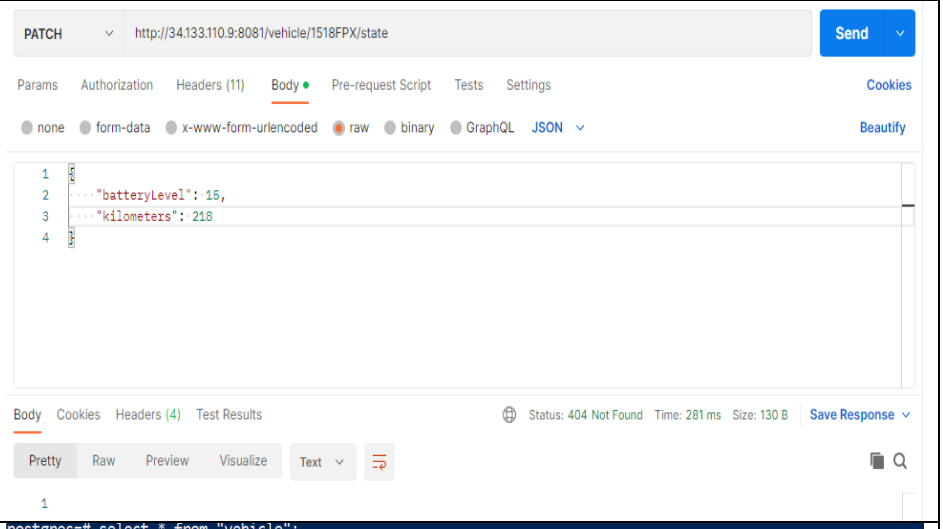
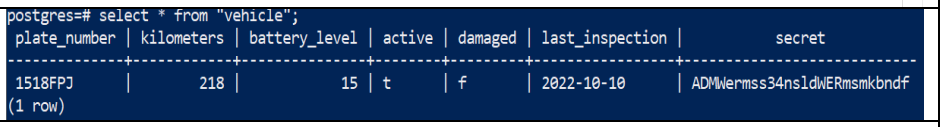
Resultado satisfactorio	Sí

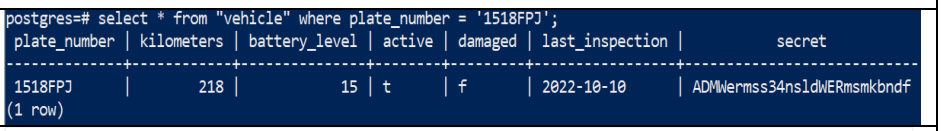
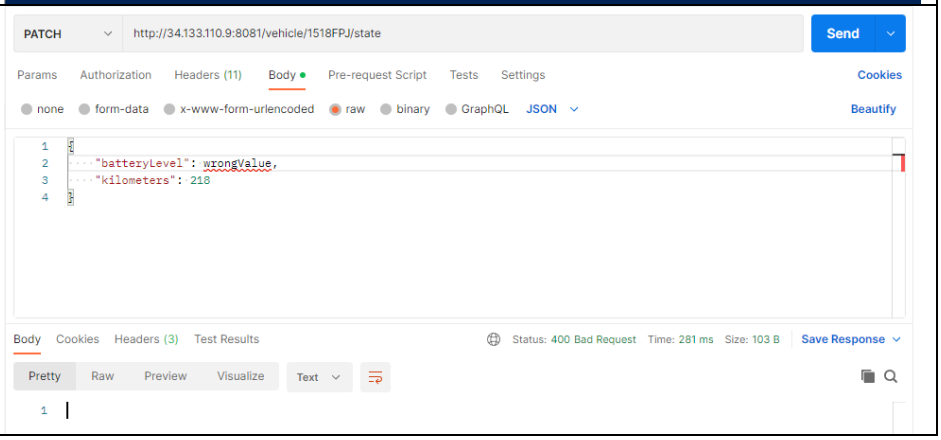
Descripción	Insertar datos de kilometraje y nivel de batería del vehículo 1518FPJ.
Precondición	Se envía como *header* "X-Authorization" el secreto encriptado del vehículo.
Resultado esperado	Respuesta código HTTP 200. Se han actualizado los valores de kilometraje y nivel de batería del vehículo.

Estado BDD antes de la operación	`postgres=# select * from "vehicle" where plate_number = '1518FPJ';`
	` plate_number | kilometers | battery_level | active | damaged | last_inspection |  secret`
	`---+---+---+---+---+---+---`
	` 1518FPJ | 100 | 50 | t | f | 2022-10-10 | ADMWermss34nsldWERmsmkbndf`
	`(1 row)`

Petición	
Estado BDD después de la operación	
Resultado satisfactorio	Sí

Descripción	Insertar datos de kilometraje y nivel de batería del vehículo 1518FPJ.
Precondición	Se envía como *header* "X-Authorization" un valor incorrecto del secreto encriptado del vehículo.
Resultado esperado	Respuesta código HTTP 401.
Estado BDD antes de la operación	
Petición	
Estado BDD después de la operación	
Resultado satisfactorio	Sí

Descripción	Insertar datos de kilometraje y nivel de batería del vehículo 1518FPX (no existente).
Precondición	-
Resultado esperado	Respuesta código HTTP 404.
Estado BDD antes de la operación	
Petición	
Estado BDD después de la operación	
Resultado satisfactorio	Sí

Descripción	Insertar datos de kilometraje y nivel de batería del vehículo 1518FPJ, indicando un valor incorrecto para kilometraje
Precondición	Se envía como *header* "X-Authorization" el secreto encriptado del vehículo.
Resultado esperado	Respuesta código HTTP 400.
Estado BDD antes de la operación	
Petición	

Estado BDD después de la operación	postgres=# select * from "vehicle" where plate_number = '1518FPJ'; plate_number | kilometers | battery_level | active | damaged | last_inspection | secret ---+---+---+---+---+---+--- 1518FPJ | 218 | 15 | t | f | 2022-10-10 | ADMWermss34nsldWERmsmkbndf (1 row)
Resultado satisfactorio	Sí

4.3.3 Componente: *Fleet manager app*

En este subapartado repetiremos el procedimiento llevado a cabo con el componente *fleet manager receiver* pero, esta vez, sobre el componente *fleet manager app*. A continuación, se detalla el proceso de validación de los prerrequisitos por tal de desplegar versión de la aplicación en entorno productivo.

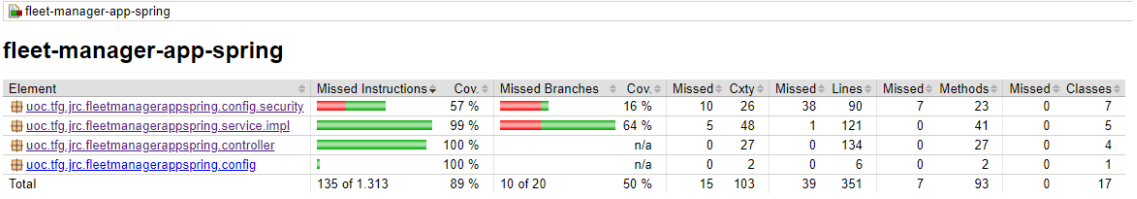
4.3.3.1 Testeo unitario: cobertura

Al igual que con el componente *fleet manager receiver*, explicado en el punto 3.2.1.1 de este documento, ejecutamos el comando "mvn -U clean install" para limpiar el proyecto, verificarlo y crear el JAR.

```
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 69, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO]
[INFO] --- jacoco-maven-plugin:0.8.8:report (report) @ fleet-manager-app-spring ---
[INFO] Loading execution data file C:\Users\Javi\Desktop\Google Drive\UOC\TFG\Software\git\fleet-manager-app-spring\targ
et\jacoco.exec
[INFO] Analyzed bundle 'fleet-manager-app-spring' with 17 classes
[INFO]
[INFO] --- maven-jar-plugin:3.2.2:jar (default-jar) @ fleet-manager-app-spring ---
[INFO] Building jar: C:\Users\Javi\Desktop\Google Drive\UOC\TFG\Software\git\fleet-manager-app-spring\target\fleet-manag
er-app-spring-0.0.1-SNAPSHOT.jar
[INFO]
[INFO] --- spring-boot-maven-plugin:2.7.4:repackage (repackage) @ fleet-manager-app-spring ---
[INFO] Replacing main artifact with repackaged archive
[INFO]
[INFO] --- maven-install-plugin:2.5.2:install (default-install) @ fleet-manager-app-spring ---
[INFO] Installing C:\Users\Javi\Desktop\Google Drive\UOC\TFG\Software\git\fleet-manager-app-spring\target\fleet-manager-
app-spring-0.0.1-SNAPSHOT.jar to C:\Users\Javi\.m2\repository\uoc\tfg\jrc\fleet-manager-app-spring\0.0.1-SNAPSHOT\fleet-
manager-app-spring-0.0.1-SNAPSHOT.jar
[INFO] Installing C:\Users\Javi\Desktop\Google Drive\UOC\TFG\Software\git\fleet-manager-app-spring\pom.xml to C:\Users\J
avi\.m2\repository\uoc\tfg\jrc\fleet-manager-app-spring\0.0.1-SNAPSHOT\fleet-manager-app-spring-0.0.1-SNAPSHOT.pom
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  19.910 s
[INFO] Finished at: 2022-12-31T10:34:55+01:00
[INFO] ------------------------------------------------------------------------
```

Ilustración 69 - Empaquetado maven

El paso anterior habrá ejecutado el plugin de jacoco, generando los html que indicarán el porcentaje de cobertura de *tests*.


fleet-manager-app-spring

fleet-manager-app-spring

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
uoc.tfg.jrc.fleetmanagerappspring.config.security		57 %		16 %	10	26	38	90	7	23	0	7
uoc.tfg.jrc.fleetmanagerappspring.service.impl		99 %		64 %	5	48	1	121	0	41	0	5
uoc.tfg.jrc.fleetmanagerappspring.controller		100 %		n/a	0	27	0	134	0	27	0	4
uoc.tfg.jrc.fleetmanagerappspring.config		100 %		n/a	0	2	0	6	0	2	0	1
Total	135 of 1.313	89 %	10 of 20	50 %	15	103	39	351	7	93	0	17

Ilustración 70 - Reporte jacoco

Según podemos ver en la ilustración 15, el 89% de cobertura de *tests* está por encima de lo requerido, por lo que puede darse como cumplido este prerrequisito.

4.3.3.2 Calidad del código: SonarQube

El primer análisis del código nos informa de múltiples irregularidades en el código que deberán subsanarse (ilustración 15).

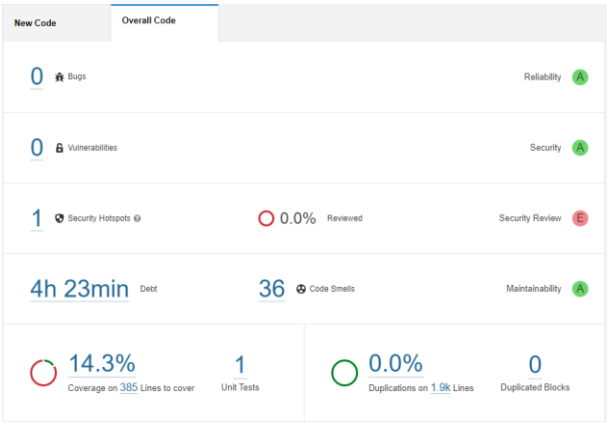


Ilustración 71 - Análisis SonarQube

Después de trabajar en las *issues*, se consigue subsanarlas todas según se muestra en la ilustración 16.

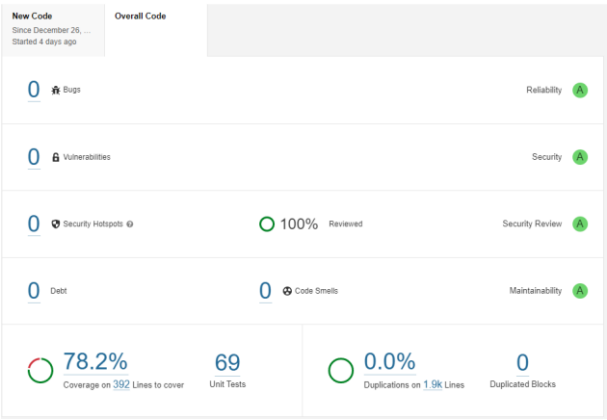


Ilustración 72 - Segundo análisis SonarQube

4.3.3.3 Generación de versión

Para la generación de versión de este componente, se seguirán los pasos seguidos en el componente previo (apartado 3.2.1.3).

- Se modifica la versión en el pom según la ilustración 7.

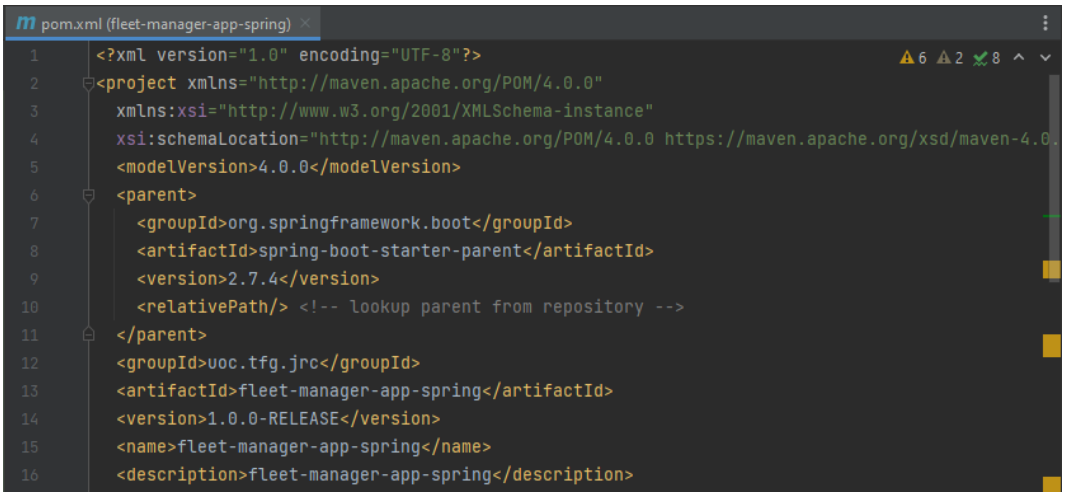


Ilustración 73 - Incremento de versión pom

- Creación de un *tag* que especificará sobre qué *commit* específico se ha generado una versión. Ilustración 18.

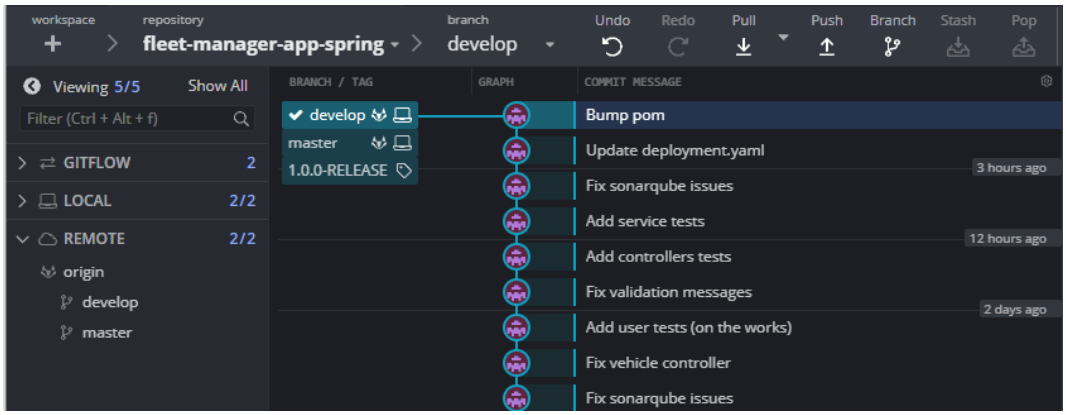


Ilustración 74 - Creación tag 1.0.0-RELEASE

En este componente, la rama *master* no estaba creada, por lo que he decidido crearla en base al último *commit* de la rama *develop*. Al hacer esto, no se requiere un paso extra de *merge* de *develop* a *master*.

4.3.3.4 Empaquetado y subida al registro de contenedores Google Container Registry (GCR)

Al igual que con el componente *fleet manager receiver*, seguiremos los pasos del punto 3.2.1.4 para el empaquetado del componente.

```
PS C:\Users\Javi\Desktop\Google Drive\UOC\TFG\Software\git\fleet-manager-app-spring> docker build -t eu.gcr.io/fleet-man
ager-365622/fleet-manager-app:v1.0.0-RELEASE .
Sending build context to Docker daemon  51.31MB
Step 1/3 : FROM openjdk:17-jdk-slim
 ---> 37cb44321d04
Step 2/3 : COPY target/fleet-manager-app-spring-1.0.0-RELEASE.jar fleet-manager-app-spring-1.0.0-RELEASE.jar
 ---> 2d621ce69869
Step 3/3 : ENTRYPOINT ["java", "-jar", "/fleet-manager-app-spring-1.0.0-RELEASE.jar"]
 ---> Running in 4a30a3ec1c1c
Removing intermediate container 4a30a3ec1c1c
 ---> 235573eacdc0
Successfully built 235573eacdc0
Successfully tagged eu.gcr.io/fleet-manager-365622/fleet-manager-app:v1.0.0-RELEASE
```

Ilustración 75 - Build docker

El siguiente paso es la subida de la imagen al registro de contenedores de google, según la siguiente ilustración.

```
PS C:\Users\Javi\Desktop\Google Drive\UOC\TFG\Software\git\fleet-manager-app-spring> docker push eu.gcr.io/fleet-manager
-365622/fleet-manager-app:v1.0.0-RELEASE
The push refers to repository [eu.gcr.io/fleet-manager-365622/fleet-manager-app]
444244f7739c: Pushed
6be690267e47: Layer already exists
13a34b6fff78: Layer already exists
9c1b6dd6c1e6: Layer already exists
v1.0.0-RELEASE: digest: sha256:079d3b4b4c9a5772a8655a17c0fcd81b4415c526715c4baf6d53daecca56d479 size: 1165
```

Ilustración 76 - Subida de la imagen

Se puede comprobar, en el container *registry* de google cloud platform, como la imagen se ha subido al registro (ilustración x).

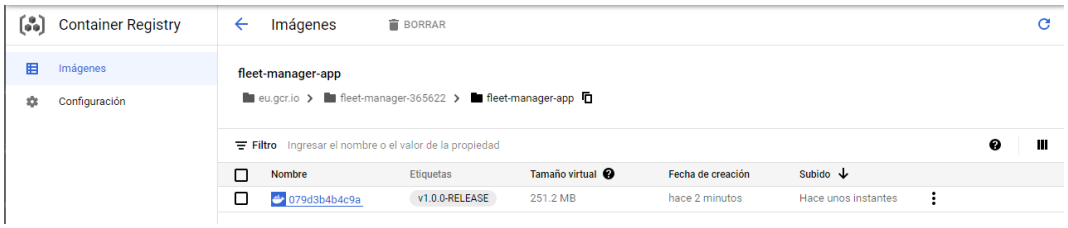


Ilustración 77 - Google container registry

4.3.3.5 *Request for change* (RFC)

A continuación, el contenido de la RFC.

a) Datos generales

Producto	Fleet manager app
Versión	1.0.0
Funcionalidades (casos de uso)	Casos de uso usuario (alta, baja, modificación y *login*). Casos de uso vehículo. Casos de uso registros. Casos de uso mantenimientos. Casos de uso repartidores.
Responsable	Javier Rubio
Fecha solicitada	31/12/2022 – 10:00 GMT+1
Duración esperada	30min
Autoriza	Javier Rubio
Entorno	Producción

Estado	Pendiente finalización pruebas de aceptación

b) Equipo de despliegue

Nombre	Rol	Contacto	Responsabilidades
Javier Rubio	Desarrollador	jarubioca@uoc.edu	Incremento de versionado, despliegue, pruebas de aceptación.

c) Descripción de cambios

Primera puesta en marcha del servicio.

d) Impacto

El impacto es general, ya que coincide con la puesta en marcha del servicio en su primera versión.

e) Descripción del despliegue

1. Generación de versión y subida a la rama master de la versión *release* generada.
2. Empaquetado de la aplicación y subida al registro de contenedores de GCP.
3. Despliegue en el *cluster* de kubernetes del proyecto GCP.

f) Rollback

En caso de inestabilidad o malfuncionamiento del servicio, al ser primera versión del servicio y no haber una versión estable a la que volver, deberá de cerrarse el servicio inmediatamente.

g) Pruebas de aceptación

Las pruebas de aceptación se encuentran en el punto 3.3.3, pruebas de aceptación, de este documento.

4.3.3.6 Despliegue

Para realizar el despliegue, nos conectaremos al *cluster* de kubernetes y aplicaremos el *configmap* de la aplicación que se encuentra en el repositorio. Enlace: https://gitlab.com/uoc-tfg-jarubioca/fleet-manager/-/blob/master/fma-deployment.yml

Ejecutamos el comando para ejecutar el despliegue del servicio según la siguiente ilustración.

```
PS C:\Users\Javi\Desktop\Google Drive\UOC\TFG\Software\git\fleet-manager-app-spring> kubectl apply -f .\fma-deployment.y
ml
deployment.apps/fleet-manager-app created
service/fleet-manager-app created
```

Ilustración 78 - Ejecución del despliegue

Ahora, se comprueba que se ha creado el despliegue, *pods* de la aplicación y servicio según el *configmap* ejecutado. En la siguiente ilustración se muestra el estado de del cluster donde pueden verse los pods y servicios creados.

```
PS C:\Users\Javi\Desktop\Google Drive\UOC\TFG\Software\git\fleet-manager-app-spring> kubectl get all
NAME                                           READY   STATUS    RESTARTS   AGE
pod/fleet-manager-app-6fd88895d5-rrp7j         1/1     Running   0          32s
pod/fleet-manager-receiver-cdf4f8b46-qq6pn     1/1     Running   0          92s
pod/postgres-666f4547b-js7m9                   1/1     Running   0          95m

NAME                               TYPE           CLUSTER-IP     EXTERNAL-IP     PORT(S)          AGE
service/fleet-manager-app          LoadBalancer   10.48.5.22     34.69.160.44    8082:31668/TCP   31s
service/fleet-manager-receiver     LoadBalancer   10.48.8.90     35.225.132.82   8081:32614/TCP   92s
service/kubernetes                 ClusterIP      10.48.0.1      <none>          443/TCP          56d
service/postgres                   ClusterIP      10.48.4.178    <none>          5432/TCP         56d

NAME                                       READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/fleet-manager-app          1/1     1            1           33s
deployment.apps/fleet-manager-receiver     1/1     1            1           94s
deployment.apps/postgres                   1/1     1            1           56d

NAME                                                 DESIRED   CURRENT   READY   AGE
replicaset.apps/fleet-manager-app-6fd88895d5         1         1         1       33s
replicaset.apps/fleet-manager-receiver-cdf4f8b46     1         1         1       94s
replicaset.apps/postgres-666f4547b                   1         1         1       56d
```

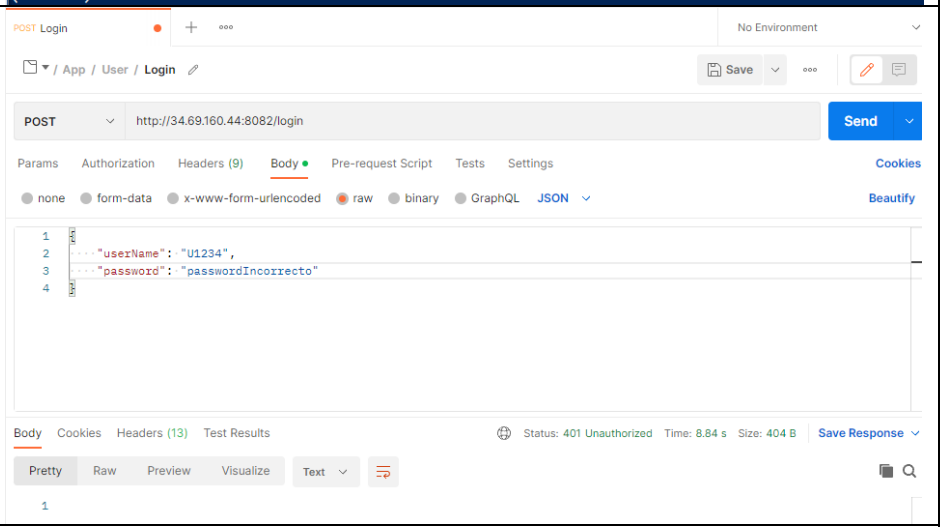
Ilustración 79 - Cluster kubernetes

4.3.3.7 Pruebas de aceptación

A continuación, se detallan todas las pruebas realizadas por tal de validar el mínimo entregable.

Descripción	*Login* como *admin* introduciendo credenciales incorrectas
Precondición	-
Resultado esperado	Respuesta código HTTP 401 (*Unauthorized*).

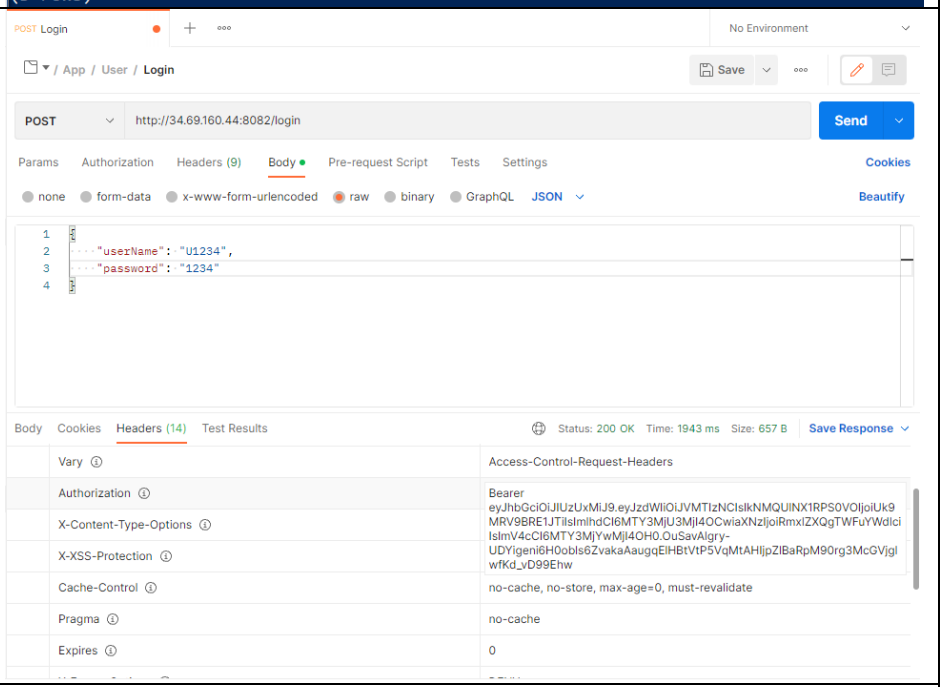
Estado BDD antes de la operación	`postgres=# select * from "user";`
	` id | username | password | active`
	`----+----------+--------------------------------------------------------------+--------`
	`  1 | U1234 | $2a$10$5l5weJICznuTPhM83r0RNecws2tIRdUU94gdUQtLz9Tpz3wR/GcIm | t`
	`  2 | U1235 | $2a$10$z9SSUgxqkRCDOkrxbNn3zeSxKWEnCrmrLXYFbuOKZRYlngw9Ch6gy | t`
	`  3 | U1236 | $2a$10$I3Cf5rY16hPzYrnnXLrZd.iCC8qKJvxrPNUktAvv8.c8Khng8uMJi | t`
	`(3 rows)`

Petición	
Estado BDD después de la operación	N/A
Resultado	Sí

satisfactorio	

Descripción	*Login* como *admin* introduciendo credenciales correctas
Precondición	-
Resultado esperado	Respuesta código HTTP 200 y respuesta del token de acceso como *header Authorization*.

Estado BDD antes de la operación	`postgres=# select * from "user";`
	` id | username |  password  | active`
	`----+----------+----------+--------`
	`  1 | U1234  | $2a$10$5l5weJICznuTPhM83r0RNecws2tIRdUU94gdUQtLz9Tpz3wR/GcIm | t`
	`  2 | U1235  | $2a$10$z9SSUgxqkRCDOkrxbNn3zeSxKWEnCrmrLXYFbuOKZRYlngw9Ch6gy | t`
	`  3 | U1236  | $2a$10$I3Cf5rY16hPzYrnnXLrZd.iCC8qKJvxrPNUktAvv8.c8Khng8uMJi | t`
	`(3 rows)`

Petición	
Estado BDD después de la operación	N/A
Resultado satisfactorio	Sí

Nota: punto incompleto debido al alto volumen de trabajo que requiere en el plazo requerido. Se propone como punto de mejora la creación de un script que automatice estas pruebas.

5. Puntos de mejora y/o pendientes

En la siguiente tabla se enumeran, detallan y se ordenan, por criterios como prioridad y criticidad, los puntos de mejora o pendientes de la aplicación.

N.	Descripción	Prioridad	Criticidad
1	En las pruebas de aceptación del componente *fleet manager app* se detecta que no se ha implementado el *endpoint* de	Muy alta	Crítico

	consultar los mantenimientos de un vehículo.		
2	Activar uso de certificado SSL para HTTPS.	Alta	Crítico
3	Aplicación frontal para navegadores.	Alta	Alta
4	Automatizado de pruebas de aceptación.	Alta	Media
5	Aumento de recursos del *cluster* de cara a la puesta en producción.	Media	Media
6	Configuración de la IP estática en el *cluster* de producción.	Media	Baja
7	*Endpoint* swagger componente *fleet manager app* requiere *header* de autenticación jwt o *endpoint* de *login* para obtenerlo.	Media	Baja
8	Entorno independiente de desarrollo para el despliegue y aceptación de nuevas versiones.	Media	Baja
9	Implementación de herramientas para el desarrollo e integración continuas (pipelines de Jenkins).	Baja	Baja
10	Configuración nombre DNS.	Baja	Baja

6. Guía para el uso de la aplicación

El código de las dos aplicaciones se encuentra subido a mi repositorio gitlab y son de acceso público. El enlace del repositorio es: https://gitlab.com/uoc-tfg-jarubioca

La aplicación se encuentra desplegada en entorno *cloud* y cada componente dispone de IP pública externa disponible según la ilustración x.

```
NAME                                TYPE           CLUSTER-IP     EXTERNAL-IP     PORT(S)          AGE
service/fleet-manager-app           LoadBalancer   10.48.2.20     34.69.160.44    8082:32599/TCP   25h
service/fleet-manager-receiver      LoadBalancer   10.48.8.90     35.225.132.82   8081:32614/TCP   25h
service/kubernetes                  ClusterIP      10.48.0.1      <none>          443/TCP          57d
service/postgres                    ClusterIP      10.48.4.178    <none>          5432/TCP         57d
```

Ilustración 80 - IP servicios

IP del componente *fleet manager receiver*: 35.225.132.82:8081.
IP del componente *fleet manager app*: 34.69.160.44:8082.

La colección de *postman* para probar los componentes puede encontrarse en: https://gitlab.com/uoc-tfg-jarubioca/fleet-manager/-/blob/develop/postman/Fleet%20Manager.postman_collection.json

Importante: según especificado anteriormente en el documento de diseño, la comunicación con el componente *fleet manager receiver* debe incluir un *header* "X-Authorization". Para conseguir un valor válido debe seguirse el siguiente procedimiento:

- Concatenar *string* fecha en formato *ddMMYYYY*, matrícula del vehículo y su secreto. Ejemplo del *string* teniendo en cuenta que queremos actuar sobre el coche con:
  - Matrícula: 1518FPJ
  - Secreto: ADMWermss34nsldWERmsmkbndf
  - Día: 010123 (1 de Enero del 2023).
  - Valor concatenado: 0101231518FPJADMWermss34nsldWERmsmkbndf

- Generar valor del hash mediante el uso del algoritmo Bcrypt con fuerza 4. Se puede conseguir ejecutando la clase BcryptUtil del proyecto *fleet manager receiver*. Enlace: https://gitlab.com/uoc-tfg-jarubioca/fleet-manager-data-receiver-spring/-/blob/master/src/main/java/uoc/tfg/jrc/fleetmanagerdatareceiverspring/utils/bcryptUtil.java

Para realizar peticiones al componente *fleet manager app*, primero debe realizarse el *login* con el rol respectivo al endpoint de lógica de negocio que quiera usarse. Como datos de muestra, existen los siguientes usuarios dados de alta:

- Usuario con rol ADMIN
  - *Username*: U1234
  - *Password*: 1234
- Usuario con rol USER
  - *Username*: U1235
  - *Password*: 1235
- Usuario con rol JEFE_LOGÍSTICA
  - *Username*: U1236
  - *Password*: 1236

7. Valoración económica

En esta sección se detallan los costes asumidos para la ejecución del proyecto.

7.1 Costes derivados de las herramientas de software

Puesto que todas las herramientas usadas durante el proyecto pertenecen a proyectos de software libre, el gasto asumido es nulo.

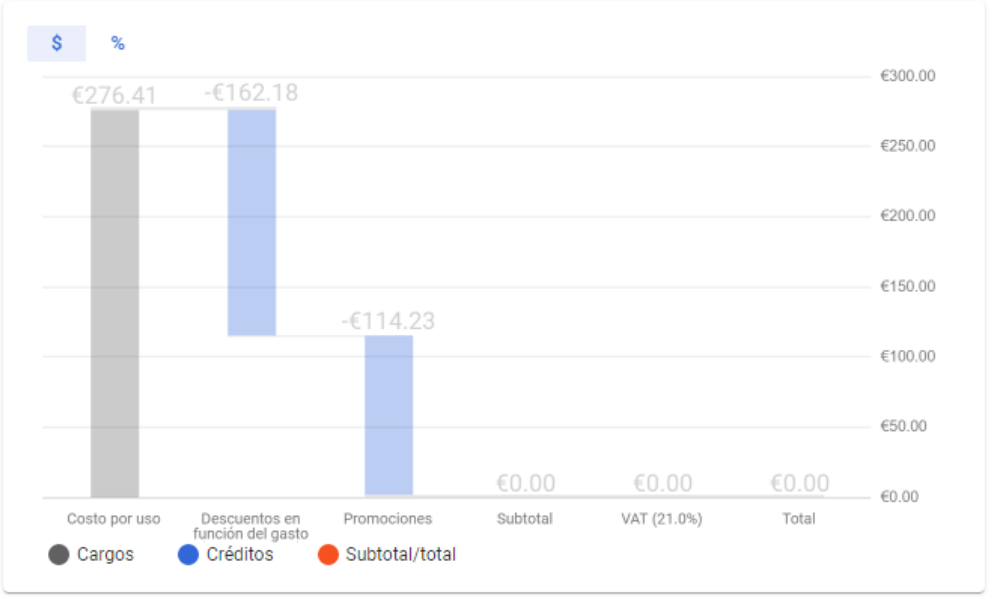
7.2 Costes *cloud*

Los costes cloud son el único gasto que se ha encontrado durante la ejecución del proyecto. Estos costes pueden calcularse fácilmente a través de la consola de facturación de Google Cloud y se especifican en la siguiente ilustración.

Desglose del costo

COMPARTIR APRENDIZAJE

La página Desglose del costo muestra el costo de tu uso básico y cómo lo afectaron los créditos, impuestos y ajustes pertinentes para llegar a tu costo total. Más información sobre los créditos



Elemento	Desglose del costo	Tasa efectiva	Importe
Costo por uso	Ver informe	100 %	€276.41
Descuentos en función del gasto (contractuales)		-58.67 %	-€162.18
Promociones		-41.33 %	-€114.23
Costo	Ver informe		€276.41
Créditos totales (descuentos, promociones y otros tipos)		-100 %	-€276.41

Ilustración 81 - Costes cloud

8. Conclusiones

Estoy muy satisfecho con el producto obtenido, ya que es funcional y abierto completamente al desarrollo de nuevas funcionalidades gracias a los diferentes métodos seguidos durante el desarrollo (librerías en versiones recientes, técnicas para un código limpio y mantenible, sistema de control de versiones, etc.). Además, seguir el proceso de ejecución del proyecto de manera ordenada y con fechas marcadas ha sido muy desafiante y enriquecedor.

En las fases de análisis y diseño he podido aplicar multitud de conocimientos adquiridos durante el grado en asignaturas como: ingeniería de requisitos, ingeniería del software, diseño de bases de datos o ingeniería del software de componentes y sistemas distribuidos. En la fase de implementación he podido aplicar los conocimientos aprendidos de asignaturas como: criptografía, fundamentos de programación, uso de bases de datos, análisis y diseño con patrones, entre otras.

Además de poner en práctica conocimientos puramente técnicos, haber cursado asignaturas como gestión de proyectos y competencias comunicativas para profesionales de las TIC, me han ayudado para entender las etapas y procesos que deben seguirse en la planificación y ejecución de un proyecto de software.

Me siento muy orgulloso de haber conseguido cumplir un objetivo específico: desplegar el producto en el entorno *cloud* de *Google Cloud Platform* usando las herramientas docker y kubernetes. El resultado ha sido muy satisfactorio, resultando en un *cluster* plenamente funcional, monitorizable, completamente accesible y estable.

Con toda probabilidad seguiré trabajando en el proyecto por mi cuenta, que serán los puntos descritos en el apartado 5 – Puntos de mejora y/o pendientes.

9. Glosario

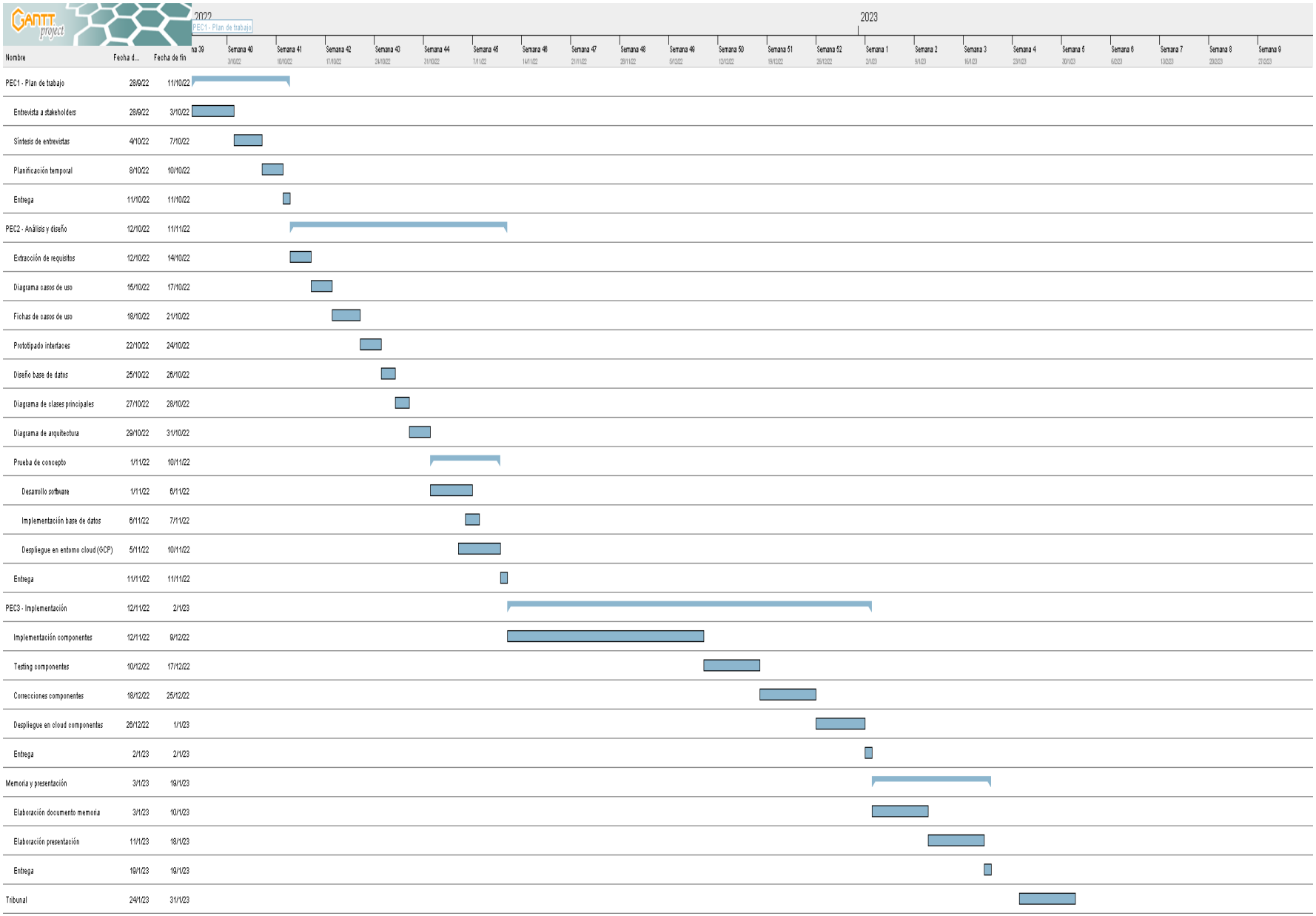
- PYME: pequeña y mediana empresa.
- *Cloud*: referido a computación en la nube, disponibilidad de recursos de sistema bajo demanda en proveedores externos.
- Container: referido específicamente a la herramienta Docker. Paquete de software ligero que incluye todo lo necesario para ejecutar una aplicación.
- *Deadline*: fecha de entrega.
- *Stakeholder*: parte interesada.
- *Stack* tecnológico: conjunto de herramientas tecnológicas utilizadas para la ejecución de un proyecto.
- Microservicio: cada uno de los servicios que forman parte de una aplicación con arquitectura de microservicios.
- *DevOps (Development and IT operations):* conjunto de prácticas que combina el desarrollo de software y operaciones IT.
- *Bug:* error, defecto o fallo en el diseño de una aplicación informática.
- *Code smell:* relacionado con el software SonarQube. Se refiere a un problema relacionado con la mantenibilidad del código.
- *Issues*: referido al software SonarQube. Unidad de defecto.
- *Cluster*: unidad de computadoras unidas en un sistema distribuido.
- Máquina virtual: equipo virtual definido por software en un servidor físico.
- *Compute engine*: unidad de cómputo específica de *Google Cloud Platform*.
- GKE (Google Kubernetes Engine): producto de Google que ofrece un *cluster* con el software específico de kubernetes.
- *Hashing*: proceso de transformar un conjunto de caracteres en otro con el fin de ofuscar el contenido en plano.
- *POC (Proof of concept)*: prueba de concepto.
- *Gibibyte*: unidad de medida usada en computación, equivale a 1.073.741.824 bytes.
- *Secret key*: clave privada, usada en algoritmos de encriptación.
- *DTO (Data Transfer ObjecT)*: objeto simple y plano que representa una entidad y que se usa específicamente para el transporte de entidades entre capas.
- *Boilerplate*: partes de código que se repiten en varias ubicaciones.
- *Setter*: método de java que asigna el valor a una variable.
- *Getter*: método de java que recupera el valor de una variable.
- *Repositorio*: ubicación centralizada que almacena, organiza, mantiene y difunde información digital.
- *Commit*: comando específico del software de control de versiones git y que corresponde a un bloque de código actualizado en el repositorio.
- *Configmap*: objeto de la API de kubernetes que guarda datos no confidenciales en pares clave-valor.
- *Header*: referido al protocolo HTTP. Campo de una petición o respuesta que contiene información adicional o metadatos.
- *Merge*: mecanismo que fusiona un conjunto de código ubicado en una rama.

10. Bibliografía

1. https://www.baeldung.com/solid-principles
2. https://www.javatpoint.com/design-patterns-in-java
3. https://martinfowler.com/articles/microservices.html
4. https://medium.com/twodigits/java-microservice-layer-architecture-3874a9a6c611
5. https://medium.com/sahibinden-technology/package-by-layer-vs-package-by-feature-7e89cde2ae3a
6. https://www.javacodegeeks.com/2015/01/given-when-then-in-java.html
7. https://kubernetes.io/docs/concepts/configuration/configmap/
8. https://www.mirantis.com/cloud-native-concepts/getting-started-with-kubernetes/what-are-kubernetes-secrets/
9. https://www.digitalocean.com/community/tutorials/maven-commands-options-cheat-sheet
10. https://en.wikipedia.org/wiki/DevOps
11. https://en.wikipedia.org/wiki/Virtual_machine
12. https://en.wikipedia.org/wiki/Content_repository

11. Anexos

11.1 Anexo 1: Diagrama de Gantt


2022
2023
PEC1 - Plan de trabajo
Nombre
Fecha d...
Fecha de fin
PEC1 - Plan de trabajo 28/9/22 11/10/22
Entrevista a stakeholders 28/9/22 3/10/22
Síntesis de entrevistas 4/10/22 7/10/22
Planificación temporal 8/10/22 10/10/22
Entrega 11/10/22 11/10/22
PEC2 - Análisis y diseño 12/10/22 11/11/22
Extracción de requisitos 12/10/22 14/10/22
Diagrama casos de uso 15/10/22 17/10/22
Fichas de casos de uso 18/10/22 21/10/22
Prototipado interfaces 22/10/22 24/10/22
Diseño base de datos 25/10/22 26/10/22
Diagrama de clases principales 27/10/22 28/10/22
Diagrama de arquitectura 29/10/22 31/10/22
Prueba de concepto 1/11/22 10/11/22
Desarrollo software 1/11/22 6/11/22
Implementación base de datos 6/11/22 7/11/22
Despliegue en entorno cloud (GCP) 5/11/22 10/11/22
Entrega 11/11/22 11/11/22
PEC3 - Implementación 12/11/22 2/1/23
Implementación componentes 12/11/22 9/12/22
Testing componentes 10/12/22 17/12/22
Correcciones componentes 18/12/22 25/12/22
Despliegue en cloud componentes 26/12/22 1/1/23
Entrega 2/1/23 2/1/23
Memoria y presentación 3/1/23 19/1/23
Elaboración documento memoria 3/1/23 10/1/23
Elaboración presentación 11/1/23 18/1/23
Entrega 19/1/23 19/1/23
Tribunal 24/1/23 31/1/23