Universidad de Valladolid

Escuela de Ingeniería Informática

Trabajo Fin de Grado
Grado en Ingeniería Informática
Mención en Ingeniería de software

Creación de un Microservicio para la Integración de Planificadores Externos

Autor:
D. Jacobo Muñoz Martínez

Tutores:
D. Jesús Mª Vegas Hernández

Creación de un Microservicio para la Integración de Planificadores Externos

Jacobo Muñoz Martínez

Índice general

Índice de figuras

Índice de cuadros

Índice de listados

Dedicado a
mi familia

Agradecimientos

A mis padres por apoyarme incondicionalmente en estos cuatro años de carrera.

A mi hermana que a pesar de vivir muy lejos, siempre me ha estado apoyando como si estuviese aquí presente.

A mis amigos por hacer este paso por la universidad una etapa inolvidable.

A mi tutor Jesús por confiar en mí para realizar este proyecto y por la ayuda que me ha brindado

A mi tutor de GMV, Francisco por confiarme un proyecto importante y ayudarme en todo lo que ha podido.

Muchas gracias a todos

Resumen

El propósito de este TFG es desarrollar un microservicio para la empresa GMV (pionera en el sector del transporte inteligente) que logre resolver algunos problemas de estructura de su sistema llamado Suite basado en microservicios, brinde más independencia a sus microservicios más importantes (o microservicios *core*) respecto a los planificadores de recursos de sus clientes y mejore la retroalimentación de estos con el cliente por medio del envío de correos electrónicos. El proyecto consta de dos partes: el desarrollo de una librería para el envío de correos electrónicos y el desarrollo del microservicio llamado *Planner Integrator* que se encargará de gestionar las asignaciones de los vehículos y conductores de sus diferentes clientes.

Este microservicio se encarga de procesar y convertir los datos de los recursos proporcionados por los clientes y se comunica con los microservicios *core* correspondientes. Toda la gestión de recursos como las asignaciones de vehículos o conductores se almacena en las bases de datos de los diferentes microservicios y puede ser monitorizada en el sistema Suite.

Este proyecto fue desarrollado en el lenguaje C#, utilizando el framework .NET siguiendo una metodología ágil basada en Scrum. Para el desarrollo e integración continua y mantenimiento del código desarrollado se ha utilizado Jenkins y SonarQube respectivamente.

Abstract

The objective of this final degree project is to develop a microservice for the company GMV (a pioneer in the intelligent transport sector) to solve some of the structural problems of its microservices-based Suite system called "Suite", provide more independence to its most important microservices (or core microservices) regarding the resource planners of its customers and improve the feedback of these with the customer by sending emails.

This microservice is in charge of processing and converting the resource data provided by the clients and communicates with the corresponding core microservices. All resource management such as vehicle or driver assignments are stored in the different microservices'databases and can be monitored in the Suite system.

This project was developed in C#, using the .NET framework, following an agile method based on Scrum. For the development and continuous integration and maintenance of the developed code, it has been used Jenkins and SonarQube.

Parte I

Memoria del Proyecto

Capítulo 1

Descripción

1.1. Contexto

Este proyecto se ha realizado en el contexto de las prácticas de empresa que se han desarrollado en la empresa GMV.

La empresa GMV es muy conocida en muchos ámbitos como el espacio, defensa, ciberseguridad, etc. Sin embargo, en Valladolid se desarrollan numerosos proyectos en el ámbito del **Transporte inteligente (ITS)** ofreciendo servicios a empresas de transporte público o privado en varias ciudades de España y países como Chipre, Malta y recientemente a Estados Unidos. GMV ofrece a sus clientes un producto llamado *ITS Suite* para poder controlar y consultar información sobre sus servicios de transporte público ya sean los horarios, horas de llegada, retrasos que se pueden producir en una línea en concreto, averías de los autobuses, etc. El sistema Suite que ofrece GMV no se basa en una arquitectura monolítica y han basado su desarrollo y arquitectura en microservicios. Varios de esos microservicios son puntos clave en su sistema, ya que muchos clientes de GMV se comunican con ellos y un pequeño fallo puede resultar fatal. Uno de estos microservicios se llama **Dated Production Plan (DPP)** y tiene la función de gestionar las importaciones de recursos de sus diferentes clientes (horarios, vehículos, conductores, etc.). El problema reside en que un pequeño cambio en el planificador del cliente o en la forma de almacenar sus datos obliga a realizar cambios en el microservicio mencionado anteriormente, lo cual no es óptimo y requiere un trabajo adicional que puede ser muy costoso. El objetivo de este proyecto de fin de grado es desarrollar un microservicio llamado **Planner Integrator** que será un intermediario entre el cliente y el microservicio DPP y así se conseguirá que este nuevo microservicio solo se encargue de recibir las peticiones lanzadas por el cliente y que las interfaces de DPP no tenga que ser modificadas. Este microservicio será una herramienta muy útil para muchos proyectos de la empresa a largo plazo y notificará y aumentará la retroalimentación con los clientes mediante el envío de correos electrónicos cuya implementación también entra dentro de este proyecto.

Este proyecto se va a enfocar en la gestión de asignaciones (o cambios) de vehículos y conductores. Estas asignaciones pueden ser de 2 tipos: asignaciones planificadas o asignaciones online. La principal diferencia entre estos 2 tipos de cambios es que en los

planificados se suele operar con ficheros (generalmente con extensión `.csv`) y en los cambios online se envían los datos a través de una petición `HTTP`. Este proyecto solo se va a centrar en los cambios online y en un futuro se implementarán los cambios planificados.

1.2. Motivación

Mi interés y motivación por las arquitecturas basadas en microservicios surgió cursando la asignatura Desarrollo Basado en Componentes y Servicios (DBCS) y durante mis prácticas de empresa curriculares y extracurriculares en GMV.

Mi curiosidad por desarrollar y aprender sobre los sistemas basados en microservicios surgió desarrollando las prácticas de DBCS, ya que me pareció un tema muy útil y eficiente y sobre todo muy importante debido a que la mayoría de sistemas que se encuentran en el mercado y empresas que desarrollan sus propios sistemas están basados en microservicios.

Es por ello que decidí entrar en GMV y hacer el proyecto de fin de grado dentro de la empresa, además de poder desarrollar un microservicio que estaría integrado en su sistema y que a largo plazo podría ofrecer servicios a varios de sus proyectos y clientes.

1.3. Objetivos del trabajo

En esta sección se expondrán los objetivos de este proyecto, los cuales se pueden dividir en 2 tipos: objetivos prácticos y objetivos personales.

- En primer lugar, este proyecto tiene como **objetivo práctico** la creación de un microservicio para lidiar con un problema de arquitectura y comunicación entre microservicios en un sistema real. Este microservicio deberá permitir a los clientes de GMV gestionar sus conductores y vehículos de forma online asignándolos a un servicio, desasignándolos de un servicio o reemplazándolos por otros. Para conseguir este objetivo se hará un estudio de la tecnología utilizada, el framework de Microsoft .NET, así como de su sintaxis, funciones, etc. y de los patrones de diseño que se podrían aplicar para desarrollar un microservicio limpio y con buena calidad y mantenimiento para sus usos y actualizaciones futuras. Este microservicio, como se dijo anteriormente, será desarrollado con el framework .NET en C# y ofrecerá varios servicios a los planificadores de los distintos clientes, como la asignación, desasignación y reemplazo tanto de conductores como de vehículos.

- Para terminar con esta sección, los **objetivos personales** impuestos son manejar con fluidez las nuevas herramientas y tecnologías que se han utilizado para desarrollar el proyecto, realizar un software real y útil desde 0 pasando por todas las fases de desarrollo que conlleva (análisis, diseño, implementación, pruebas, etc.) y adaptarse a un ambiente de trabajo, a un flujo de trabajo y planificación profesional.

1.4. Estructura del documento

Esta memoria se estructura de la siguiente forma:

- **Parte 1: Memoria del proyecto**
  - Un 1er capítulo en el que se expondrá la motivación que ha llevado a desarrollar este proyecto y sus objetivos.
  - Un 2º capítulo en el que se presentará el proyecto y se dará una panorámica del proyecto junto con sus ventajas e inconvenientes frente a otras propuestas.
  - Un 3er capítulo se detallarán las metodologías y herramientas que se han utilizado para plantear el proyecto.
  - Un cuarto 4º en el que se desarrollará la planificación del proyecto, estimaciones de esfuerzo, presupuestos y los riesgos que pueden poner en peligro el transcurso del proyecto.
- **Parte 2: Documentación técnica**
  - Un 5º capítulo en el que se presenta la parte de análisis y se analizan los requisitos del cliente.
  - Un 6º capítulo en el que se presenta el diseño del microservicio y demás herramientas adicionales y se detallan los patrones de diseño que se han usado.
  - Un 7º capítulo en el que se presenta la implementación de la aplicación y se destacan las partes de la implementación más características de la arquitectura.
  - Un 8º capítulo en el que se presentan las pruebas de los end-points y las pruebas de integración que se han realizado.
- **Parte 3: Conclusiones y líneas futuras**
  - Un 9º capítulo que expondrá las conclusiones que se obtendrán gracias al desarrollo del proyecto: análisis de la solución propuesta, de la consecución de los objetivos y posibles mejoras y ampliaciones
- **Parte 4: Apéndices**
  - Un apéndice A que presenta el manual de instalación
  - Un apéndice B que presenta el manual de usuario

Capítulo 2

Estado del arte

2.1. Problemas de diseño en los sistemas

Pocos años atrás y actualmente, muchas empresas se encuentran en un proceso de migración y cambio de arquitectura de sus sistemas, ya que en el pasado adoptaron un diseño erróneo, poco apropiado o apropiado a corto plazo para ellos. Uno de estos diseños erróneos o poco apropiados son los sistemas monolíticos.

El término **"monolito"** suele referirse a algo grande, apelmazado o glacial, entre otros, lo cual no está muy alejado de la realidad de las arquitecturas software monolíticas. Un **sistema monolítico** es un modelo tradicional de sistema software que se compila como una unidad unificada y que es autónoma e independiente de otras aplicaciones. Es una red informática grande y única con una única base de código fuente que satisface todos los intereses, necesidades y requisitos de los clientes. Este tipo de sistemas pueden resultar prácticos al inicio del proyecto, ya que no hay mucha carga de código ni dependencias entre los diferentes componentes de este. En la figura 2.1 se puede observar un esquema simplificado de un sistema monolítico [7].

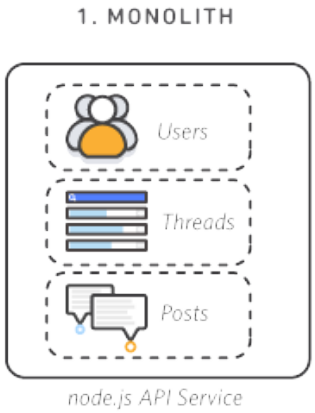

Figura 2.1: Esquema simplificado de un sistema monolítico [23]

2.1.1. Ventajas de los sistemas monolíticos

Las arquitecturas monolíticas, al igual que las arquitecturas basadas en microservicios, ofrecen ventajas y también desventajas. La ventaja principal de las arquitecturas monolíticas es la velocidad en el desarrollo debido a la simplicidad de tener una aplicación o sistema basado en una única base de código. Alguna de las ventajas de un sistema monolítico son las siguientes:

- **Proceso de desarrollo**: es más fácil empezar un proyecto con aplicaciones monolíticas, ya que no requiere mucha planificación inicial y un diseño más sencillo. Se pueden ir agregando módulos de código según sea necesario.
- **Implementación sencilla**: un único archivo o directorio ejecutable facilita la implementación a los desarrolladores
- **Pruebas simplificadas**: una aplicación monolítica es una unidad única y centralizada, por lo que las pruebas de integración y unitarias se pueden hacer más rápido que con una aplicación distribuida.
- **Depuración sencilla**: con todo el código ubicado en un solo lugar, es más fácil encontrar bugs y fallos.

Estas ventajas son muy atractivas para los desarrolladores, pero solo a corto plazo y en el inicio del proyecto [7].

2.1.2. Desventajas de los sistemas monolíticos

A medida que aumenta el tamaño y la funcionalidad se van encontrando inconvenientes como estos [7]:

- **Velocidad de desarrollo en aplicaciones grandes**: a medida que va aumentando el tamaño y la complejidad en un sistema monolítico, la velocidad de desarrollo es más lenta.
- **Escalabilidad**: Las aplicaciones monolíticas se enfrentan a varios desafíos a medida que escalan, ya que contienen toda la funcionalidad en un único código base por lo que toda la información debe escalarse a medida que cambian los requisitos.
- **Fiabilidad**: si se produce un error en algún módulo, puede afectar a toda la disponibilidad de la aplicación.
- **Cambios en el sistema**: en cualquier momento se puede necesitar la adopción de una nueva tecnología para mantener actualizado el sistema y ese cambio afectaría a toda la aplicación, por lo que los cambios serían muy costosos y lentos.
- **Barrera de implementación**: un pequeño cambio en la aplicación puede requerir una nueva implementación de todo el monolito

2.2. Una nueva arquitectura: los microservicios

En 2009, Netflix tuvo muchos problemas de crecimiento. Su infraestructura no podía seguir el ritmo de la demanda de sus servicios, que crecía a toda velocidad, ya que se volvió muy popular. En esta situación, la empresa decidió principalmente reemplazar su arquitectura monolítica por otra de microservicios y migrar su infraestructura de TI de sus centros de datos privados a una nube pública.

Netflix se convirtió en una de las primeras empresas destacadas en migrar de un monolito a una arquitectura de microservicios basada en la nube. En la actualidad, Netflix tiene más de un millar de microservicios que administran y respaldan partes independientes de su sistema, mientras que sus desarrolladores implementan código con frecuencia.

Netflix fue pionero en lo que, desde entonces, es algo cada vez más habitual: la transición de una arquitectura monolítica a otra de microservicios [7].

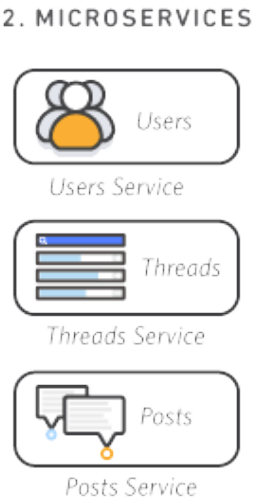

Figura 2.2: Esquema simplificado de un sistema basado en microservicios [23]

En estos últimos años los microservicios han ganado mucho impulso, atención y popularidad entre las empresas tecnológicas, ya que facilitan mecanismos y metodologías de entrega ágiles para arquitecturas orientadas a servicios.

Los microservicios son pequeñas aplicaciones con una sola responsabilidad que se despliegan, escalan y prueban de forma independiente dentro de cada microservicio. Estos microservicios tienen su propia lógica empresarial y centro de persistencia de datos (por lo general, base de datos) con un objetivo específico. Desacoplan los principales intereses específicos de dominios empresariales en bases de código independientes. Los microservicios no reducen la complejidad, pero hacen que cualquier complejidad sea visible y más gestionable, ya que separan las tareas en procesos más pequeños que funcionan de manera independiente entre sí y contribuyen al conjunto global. La adopción de microservicios suele ir de la mano de DevOps (Kubernetes, Docker, Git, Jenkins, SonarQube, etc. [8]),

ya que son la base de las prácticas de entrega continua [1] con las que los equipos pueden adaptarse rápidamente a los requisitos de los usuarios. En los sistemas monolíticos los módulos no se pueden ejecutar de manera independiente, es por eso que los sistemas monolíticos se pueden dividir microservicios dando como resultado un sistema granular cuyos elementos se comunican mediante mensajes (APIs remotas o RPC o APIs RESTful) como se muestra en la figura 2.3.

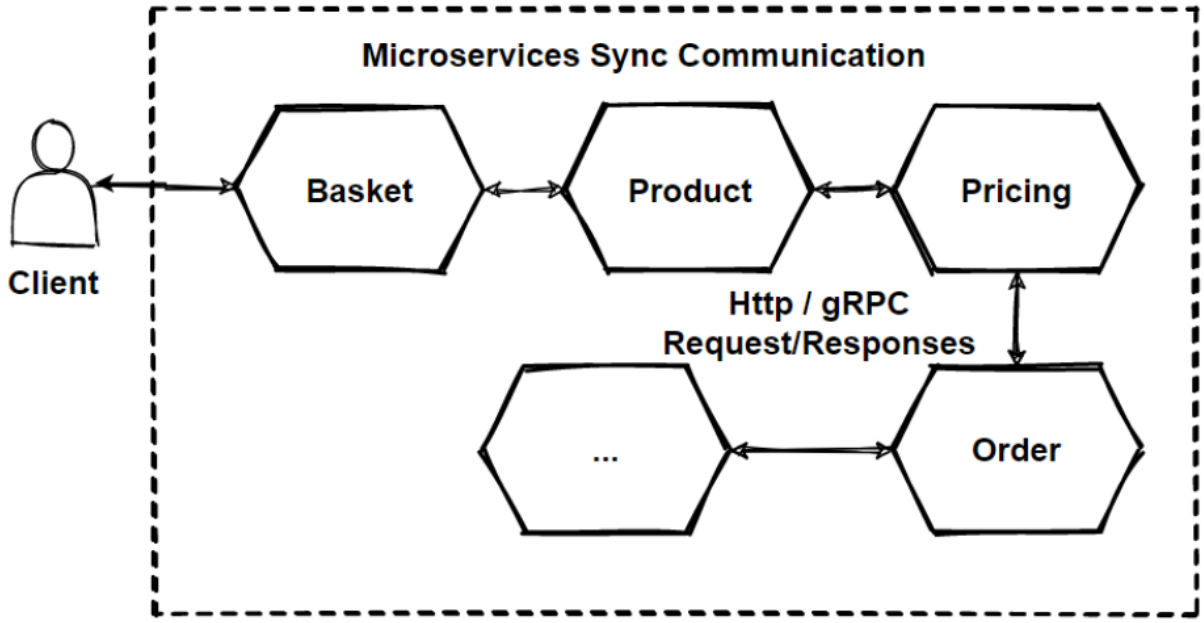

Figura 2.3: Esquema simplificado de la comunicación entre microservicios [27]

Además, permite a los equipos de desarrollo con metodologías de trabajo ágiles estructurar su trabajo y sus tareas alrededor de los microservicios que pueden ser desarrollados de forma independiente.

2.2.1. Ventajas de las arquitecturas basadas en microservicios

Los sistemas de grandes empresas como Amazon, LinkedIn, Uber, etc. se caracterizan por estar orientados a microservicios debido a las ventajas que ofrece este tipo de arquitectura [4]:

- **Agilidad**: promueve formas ágiles de trabajar con equipos pequeños que implementen con frecuencia.
- **Muy fácil de mantener y probar**: los equipos pueden hacer pruebas con el código y recular si algo no funciona como se espera. Esto facilita la actualización del código

[1] La entrega continua constituye un enfoque en el que los equipos publican productos de calidad con frecuencia y de forma predecible desde repositorios de código fuente hasta la producción de forma automatizada. [9]

y acelera el tiempo de salida al mercado de nuevas funciones. Además, es fácil aislar y corregir fallos en servicios individuales.

- **Implementación independiente**: los microservicios son unidades individuales, por lo que permiten una implementación independiente, rápida y sencilla de funciones individuales.
- **Flexibilidad tecnológica:** con las arquitecturas de microservicios, los equipos pueden elegir con libertad las herramientas y tecnologías que desean, ya que los diferentes microservicios solo se comunican entre sí, no dependen de las tecnologías de los demás.
- **Alta fiabilidad**: se puede implementar cambios para un microservicio en concreto sin el riesgo de que se caiga toda la aplicación.
- **Equipos más satisfechos**: los equipos de desarrolladores que trabajan con microservicios están mucho más satisfechos, ya que son más autónomos y pueden compilar e implementar de forma independiente, sin esperar semanas a que se apruebe una solicitud de incorporación de cambios.

2.2.2. Desventajas de las arquitecturas basadas en microservicios

Cuando se pasa de una pequeña cantidad de bases de código monolíticas a un gran número de sistemas y servicios distribuidos, surge una complejidad añadida. Este aumento de complejidad conduce a un desarrollo descontrolado o a un crecimiento rápido y no administrado. Al principio es normal añadir nuevas capacidades con menos velocidad y confianza que con una arquitectura monolítica. Al principio es normal ver difícil o no entender cómo se relacionan los diferentes componentes entre sí [7].

- **Desarrollo descontrolado**: los microservicios suman complejidad en comparación con las arquitecturas monolíticas, ya que hay más servicios en más lugares creados por varias personas. Si el desarrollo descontrolado no se gestiona adecuadamente, se reduce la velocidad de desarrollo y el rendimiento operativo.
- **Costes exponenciales de infraestructura**: cada nuevo microservicio puede tener su propio coste para el conjunto de pruebas, los manuales, la infraestructura de despliegue, las herramientas de supervisión, etc.
- **Más sobrecarga organizativa**: los equipos deben agregar otro nivel de comunicación y colaboración para coordinar las actualizaciones de los diferentes microservicios.
- **La propiedad no está definida**: a medida que se introducen más servicios, también lo hace el número de equipos que los ejecutan. Con el tiempo, se hace difícil conocer los servicios disponibles que un equipo puede aprovechar y con quién hay que hablar para obtener asistencia.

2.3. El proceso de migración a los microservicios

Muchos proyectos reales comenzaron al principio como un monolito y un tiempo más tarde evoluciona a un sistema basado en microservicios. Esta evolución del sistema a lo largo del tiempo se denomina **migración**. Antes de comenzar con la migración conviene seguir una serie de guías:

- **Estar preparado para cambios en la organización**: con los nuevos cambios de arquitectura hay que asegurarse que no se dejan de satisfacer los requisitos de usuario.
- **Estudiar el sistema**: es necesario identificar las independencias tanto de herramientas y tecnologías como de módulos de código.
- **Definir la estructura**: se debe definir la arquitectura del sistema, incluyendo las herramientas, frameworks, tecnologías, etc. Aunque no es el momento más adecuado para elegir las tecnologías, conviene no posponerlo.
- **Priorizar los componentes para la migración**: hay que realizar un estudio sobre los riesgos que pueden surgir al realizar la migración antes de ponerse en marcha con el proceso y ver que componentes son más críticos para el funcionamiento del sistema.
- **Realizar un buen diseño, implementación, pruebas e integración**: es recomendable adoptar un método ágil y usar herramientas DevOps como Git o GitHub, Jenkins, Kubernetes, etc.

Además, hay que tener en cuenta que el proceso de migración a una arquitectura basada en microservicios puede afectar a ciertos aspectos de la calidad:

- **Seguridad**: como la información se transporta entre los microservicios, se crea la necesidad de protegerla mediante técnicas de encriptación y métodos de autenticación.
- **Rendimiento**: generalmente los microservicios tienen un peor rendimiento que un monolito, aunque el rendimiento del sistema final puede depender de diferentes factores como la latencia de red o la virtualización, ya que el despliegue en máquinas virtuales puede añadir más carga.
- **Fiabilidad**: como los microservicios son distribuidos son, por naturaleza, menos fiables que los monolitos
- **Disponibilidad**: en arquitecturas basadas en microservicios la disponibilidad depende no solo de la disponibilidad de los propios microservicios, sino también de su integración. Por otro lado, los microservicios reducen el tiempo de despliegue, por lo que aumenta la disponibilidad.

- **Mantenibilidad**: los microservicios son independientes, por lo que su mantenibilidad es uno de los mejores aspectos de este tipo de arquitecturas.
- **Capacidad de prueba**: inicialmente un sistema basado en microservicios es más fácil de probar que un monolito, aunque las pruebas de integración pueden ser mucho más complejas depende de las dependencias y las conexiones entre los diferentes microservicios.

Es muy común que al realizar la migración de un sistema monolítico a un sistema basado en microservicios no sea suficiente solo con desacoplar módulos y que se comuniquen entre ellos, ya que hay que preservar todos los atributos de calidad, robustez, seguridad, etc. en el sistema. Como se explicó en la sección 1.1, existen sistemas (como en el caso de GMV) en los que ciertos de sus microservicios son clave para el funcionamiento del mismo (se denominan microservicios ***core***) y un pequeño fallo puede acarrear graves problemas para todo el conjunto. Es por ello que en los procesos de migración suele ser común la creación de microservicios adicionales los cuales se les denomina **pasarela** que se encargan de recibir y procesar los datos y enviar solo datos correctos al microservicio ***core***.

Capítulo 3

Metodología

3.1. Proceso de desarrollo

Para llevar a cabo este proyecto se ha adoptado la metodología que se usa en el equipo de proyectos en el que se me incluyó el entrar en la empresa: ***Scrum***. Scrum se incluye dentro de las metodologías ágiles para el desarrollo de proyectos software que, como se estableció en el ***Agile Manifesto*** o Manifesto ágil, se caracterizan por estos valores [20]:

- Individuales e interacciones sobre procesos y herramientas. Las reuniones con el equipo cobran mucha importancia.
- Software funcional sobre documentación. Aunque en GMV se valora mucho una buena documentación.
- Colaboración del cliente sobre negociaciones contractuales. La retroalimentación con el cliente es clave.
- Respuesta al cambio sobre seguir un plan establecido.

Scrum se basa en el empirismo que afirma que el conocimiento proviene de la experiencia y la toma de decisiones basadas en la observación. Es una metodología ágil en la que un equipo único y reducido (que involucra a personas que tienen las habilidades y experiencia para realizar el trabajo y compartir o adquirir dichas habilidades) desarrolla, entrega y mantiene software complejo de forma iterativa e incremental en bloques de tiempo cortos llamados (***sprints***), gestionando los riesgos e incertidumbres que puedan surgir durante ese tiempo [2]. El proceso se basa en 3 pilares:

- **Transparencia**: El proceso y el trabajo emergentes deben ser visibles tanto para aquellos que realizan el trabajo como para los que lo reciben. La transparencia permite la inspección. La inspección sin transparencia genera engaños y desperdicios.
- **Inspección**: Los artefactos de Scrum y el progreso hacia los objetivos acordados deben ser inspeccionados y revisados con frecuencia para detectar desviaciones o problemas en el trabajo potencialmente indeseables.

- **Adaptación**: Si algún aspecto del proceso se desvía fuera de los límites aceptables o el producto resultante es inaceptable, el proceso que se está aplicando debe ajustarse. El ajuste debe realizarse lo antes posible para minimizar la desviación adicional. Se espera que un equipo de Scrum se adapte en el momento en que aprenda algo nuevo por medio de la inspección.

Dentro de Scrum existen 3 componentes fundamentales que se desarrollarán en las siguientes subsecciones:

3.1.1. Roles

Los equipos Scrum son multifuncionales (los miembros tienen las habilidades necesarias para aportar valor en cada *sprint*) y autogestionados (se decide internamente quien hace qué, cuándo y como). El equipo Scrum es suficientemente pequeño para adoptar una metodología ágil y suficientemente grande para completar todo el trabajo en un *sprint* (por lo general 10 o menos personas). Todo el equipo de Scrum es responsable de crear un incremento valioso y útil en cada *sprint*. Scrum define tres responsabilidades específicas dentro del equipo de Scrum: el equipo de desarrollo, el *Product owner* (propietario del producto) y el *Scrum master*.

- **Equipo de desarrollo**: son las personas del equipo que se comprometen a desarrollar cualquier aspecto funcional y aportar valor en cada *sprint*
- **Propietario del producto o (*Product owner*)**: es una única persona y es el responsable de maximizar el valor del producto resultante del trabajo del equipo Scrum. También es responsable de tomar decisiones (las cuales deben ser respetadas por el equipo) y gestionar de forma eficaz la pila del producto (también llamado *Product backlog*, véase en la subsección 3.1.2)
- ***Scrum master***: es la persona que se encarga de establecer y asegurar que se adoptan las prácticas Scrum en el equipo. El *Scrum master* también es responsable de ser el punto de comunicación entre el equipo y el cliente.

En la Figura 3.1 se pueden observar los roles Scrum.

3.1.2. Artefactos

El resultado de las actividades en Scrum se materializa en artefactos, que representan trabajo o valor. Cada artefacto contiene un compromiso para garantizar que proporciona información que mejora la transparencia y el enfoque con el que se puede medir el progreso. Los artefactos considerados en Scrum son:

- ***Product backlog***: es una lista ordenada de características y mejoras que se implementarán en el producto. El responsable del *Product backlog* es el *Product owner* y puede influir en los desarrolladores ayudándoles a entender las tareas y a resolver los diferentes problemas que puedan surgirles. El objetivo que se persigue es el *Product goal* que describe el estado futuro o final del producto al que se quiere llegar.

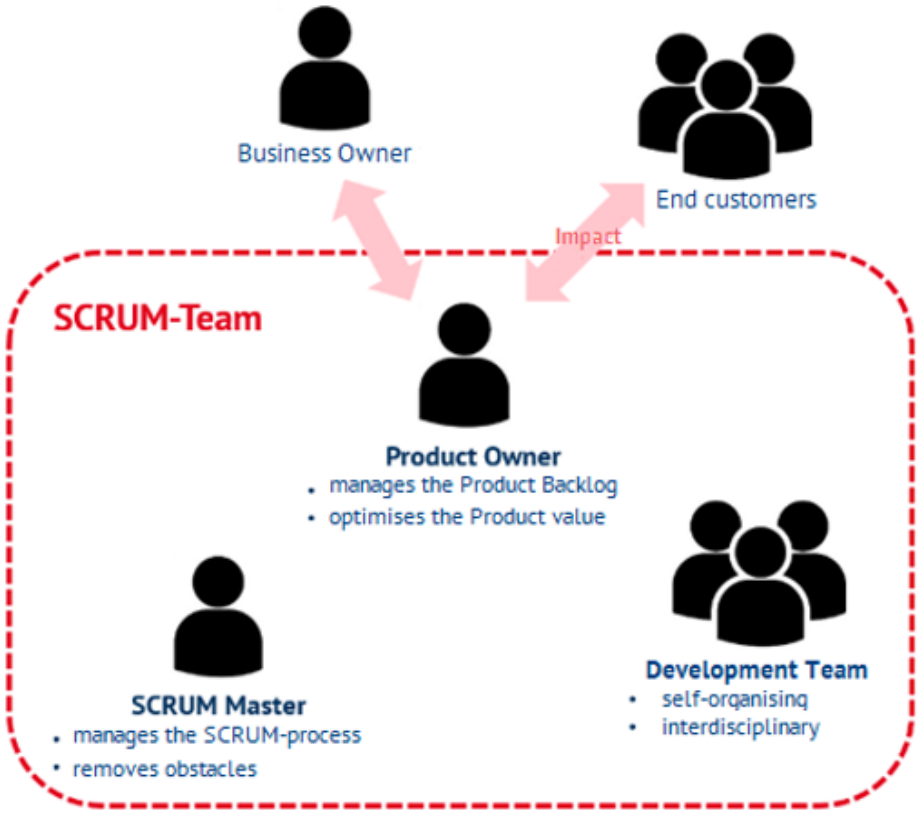

Figura 3.1: Roles en el equipo Scrum [24]

- ***Sprint backlog***: es una lista que contiene las tareas pendientes de realizar durante el *sprint* pertenecientes al *Product backlog*. Los principales responsables de realizar el trabajo son los desarrolladores. Es una imagen muy visible y en tiempo real del trabajo que los desarrolladores planean realizar durante el *sprint* para lograr el *Sprint goal* que es el objetivo que se establece para el *sprint*.

- **Incremento**: Es el resultado del trabajo realizado durante el *sprint*. Cada incremento es aditivo a todos los incrementos de sprints anteriores, fusionando todos ellos para acercarse al *Product goal*. Se pueden crear varios incrementos dentro de un *sprint*.

En la Figura 3.2 se pueden observar los artefactos Scrum y en la Figura 3.3 se pueden observar los artefactos junto con los eventos Scrum que se desarrollan en la subsección 3.1.3

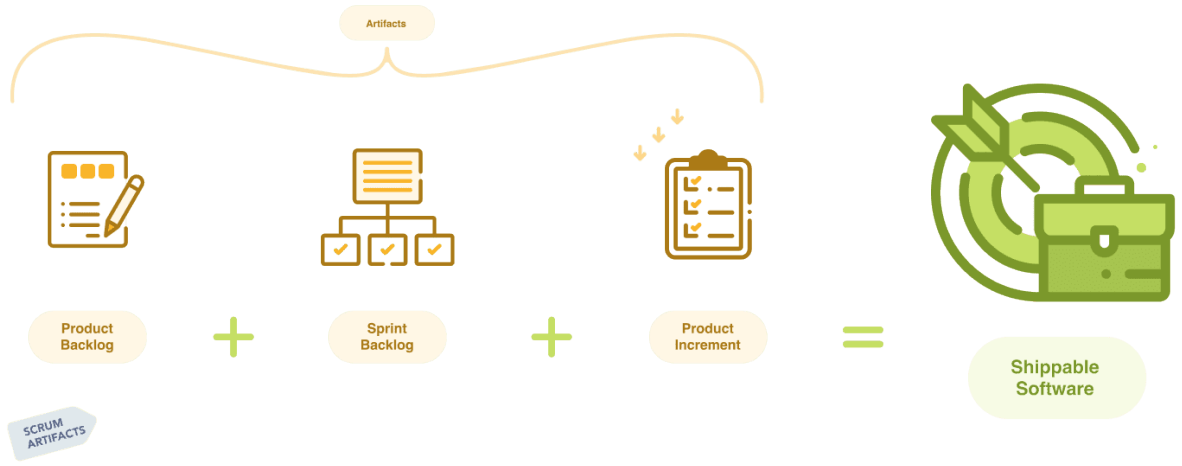

Figura 3.2: Artefactos Scrum [29]

3.1.3. Eventos

Cada evento en Scrum es una oportunidad para revisar y adaptar los artefactos de Scrum vistos en la subsección 3.1.2. Los eventos se utilizan en Scrum para crear regularidad y minimizar la necesidad de reuniones no planificadas o de urgencia y se llevan a cabo al mismo tiempo y lugar para reducir la complejidad. Los eventos Scrum son los siguientes:

- ***sprint***: es el contendor de todos los demás eventos. Son eventos de duración fija para crear consistencia y suelen durar menos de un mes. Un nuevo *sprint* comienza inmediatamente después de la finalización del *sprint* anterior. Durante un *sprint* los requisitos se congelan, es decir, no se pueden cambiar.

- **Planificación de *sprint* o *Sprint planning***: es el inicio del *sprint* y establece el trabajo que se va a realizar para el mismo. La planificación resultante es creada por el trabajo de todo el equipo Scrum. En esta reunión participan todos los miembros del equipo y se seleccionan los elementos del *Product backlog* (véase en la subsección 3.1.2) que ya han sido priorizados que se realizarán en el *sprint*. El artefacto resultante del *Sprint planning* es el *Sprint backlog* (véase en la subsección 3.1.2)

- **Scrum diario o *Daily scrum***: es una reunión diaria cuyo propósito es hacer un seguimiento del trabajo e inspeccionar el progreso que se ha realizado para alcanzar el *Sprint goal* (véase en la subsección 3.1.2). Suele tener una duración de 15 minutos como máximo y se realiza a la misma hora y el mismo lugar todos los días. Estas reuniones diarias mejoran la comunicación, identifican problemas, promueven la toma de decisiones, y en consecuencia, eliminan la necesidad de otras reuniones.

- **Revisión de *sprint* o *Sprint review***: es una reunión que se realiza al final del *sprint* y tiene como propósito inspeccionar el resultado del *sprint* y determinar futuras adaptaciones, funcionalidades y mejoras. El equipo presenta los resultados

de su trabajo a los clientes y se discute el progreso hacia el *Product goal* (véase en la subsección 3.1.2. Se pueden hacer cambios en el *Product backlog* (véase en la subsección 3.1.2

- **Retrospectiva de *sprint* o *Sprint retrospective***: es el evento que concluye el *sprint*. El equipo evalúa cómo fue el último *sprint* con respecto a individuos, interacciones, procesos y herramientas, qué fue bien, si se encontraron (o no) problemas y si fueron resueltos. Los elementos revisados a menudo varían según el dominio del trabajo. Si hubo algún desvío se identifica y se intenta encontrar el origen para poder corregirlo.

En la Figura 3.3 se pueden observar los eventos Scrum junto con los artefactos vistos en la subsección 3.1.2.

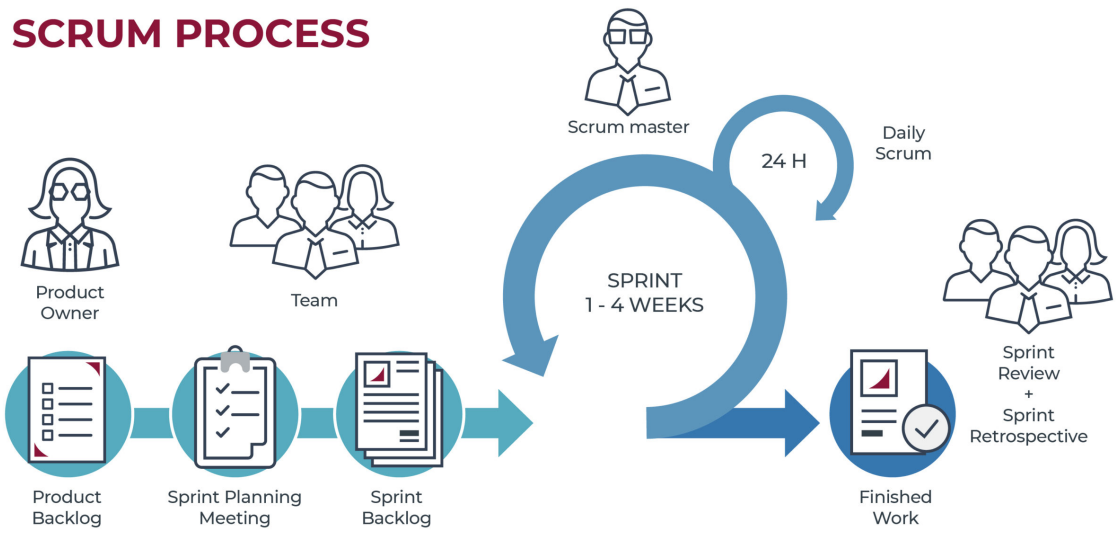

Figura 3.3: Eventos Scrum junto a los artefactos Scrum [36]

3.2. Adaptación de Scrum al proyecto

En este proyecto no se tomó ninguna decisión sobre qué metodología usar, ya que el equipo al que me asignaron en GMV ya usaba el método Scrum. Aun así, cuando se creó el equipo se tomó la decisión de adoptar el marco Scrum porque los proyectos que se iban a desarrollar iban a ser muy complejos y necesitaban un seguimiento muy frecuente y ser divididos en tareas para que fuese manejable para los desarrolladores. Además, que Scrum se volvió muy popular entre las empresas del sector de transporte.

Sobre los **roles**, en este proyecto ha habido principalmente dos personas (el estudiante que desarrolla el proyecto y su tutor de la empresa) y se han repartido de la siguiente forma:

- El estudiante se encargará del desarrollo, por lo que adoptará el papel del equipo de desarrollo visto en la subsección 3.1.1.

- El tutor de la empresa adoptará los papeles de *Scrum Master* y *Product Owner*. Se encarga de la repartición y priorización de tareas y contacto con las partes interesadas, así como de la revisión del trabajo y orientar al estudiante en el transcurso del proyecto.

Sobre los **eventos**, la duración de los sprints del equipo son de 2 semanas (salvo alguna excepción, la cual nunca sobrepasa las 3 semanas). El *Sprint Planning* y el *Sprint Review* se juntan y sea realizan en una sola reunión que se realiza los miércoles a las 10:30 cada 2 semanas. Las reuniones *Daily scrum* se realizan todos los días a las 9:25 de la mañana (salvo alguna excepción).

Sobre los **artefactos**, el tutor de la empresa se encarga de gestionar el *Product backlog* y el *Sprint backlog* mediante la herramienta Jira.

3.3. Tecnologías utilizadas

En esta sección se listan las tecnologías que han sido necesarias para llevar a cabo el proyecto:

- **.NET Framework [5]:** es un framework desarrollado por Microsoft que admite la creación y ejecución de servicios web y aplicaciones Windows. Esta tecnología tiene como enfoque cumplir estos objetivos:

  - Proporcionar un entorno de programación orientada a objetos coherente.
  - Ofrecer al desarrollador una experiencia coherente entre tipos de aplicaciones muy diferentes como las basadas en Windows o en web.
  - Garantizar la integración del código desarrollado en .NET con otras tecnologías.

  .NET consta de 2 componentes principales:

  - **Common Language Runtime (CLR):** administra la memoria, ejecución de subprocesos, ejecución de código. comprobación de la seguridad del código y compilación.
  - **Biblioteca de clases de .NET:** es una colección de tipos reutilizables que se integran con CLR. Está orientada a objetos, son fáciles de usar y reduce el tiempo y la curva de aprendizaje de .NET. Además, las bibliotecas, componentes y librerías de terceros se integran sin dificultad con las de .NET. La biblioteca de clases permite realizar diversas tareas de programación comunes. Además de estas, la biblioteca de clases incluye tipos adecuados para diversos escenarios de desarrollo especializados como aplicaciones de consola, servicios de Windows, aplicaciones de GUI con Windows Forms, etc.

Esta tecnología se ha usado para desarrollar todo el back-end y la funcionalidad del microservicio.

- **Git [3]:** es un software de control de versiones ideal para mantener aplicaciones y sistemas con numerosos archivos de código fuente. Git controla los cambios que han sido producidos en un fichero o conjunto de ficheros con el transcurso del tiempo para que el usuario pueda organizar dichos cambios en versiones y poder recurrir a ellas en caso de que lo necesite. La principal diferencia entre Git y otros software de control de versiones, como Subversion, es la forma en la que Git analiza sus datos. En otros sistemas, la información se almacena en una lista de cambios basada en archivos y consideran la información que almacenan como un conjunto de archivos y los cambios realizados en cada archivo a lo largo del tiempo (esto se describe comúnmente como control de versiones basado en delta). En Git se tratan los datos como instantáneas o snapshots, es decir, cada vez que el usuario confirma o guarda el estado de su proyecto o fichero, Git toma una fotografía de cómo se ven todos tus archivos en ese momento y almacena una referencia a esa instantánea. Para ser eficiente, si los archivos no han cambiado, Git no vuelve a almacenar el archivo, solo un enlace al archivo anterior idéntico que ya almacenó. Git piensa en sus datos más como un flujo de instantáneas. Para este proyecto se ha usado Bitbucket, una herramienta de alojamiento de código basada en Git y diseñada para equipos. Bitbucket se integra con Jira y ayuda a todo el equipo a mantener el proyecto organizado y permite crear un código de calidad con sus pruebas automatizadas.

  El flujo de trabajo que se ha seguido con Bitbucket ha sido el siguiente:

  1. Se crea una tarea en Jira con una descripción clara de que se debe desarrollar
  2. Desde la tarea de Jira creamos una rama de desarrollo en el repositorio correspondiente de Bitbucket
  3. Se desarrolla lo correspondiente a la tarea
  4. Se crea una Pull Request (o PR) que es una solicitud para fusionar código de una rama a otra (normalmente de la rama personal del desarrollador a la rama principal de desarrollo).
  5. Una vez creada la PR un supervisor revisa el código que se quiere subir a la rama principal y, si es necesario, propone cambios para que el desarrollador los corrija.
  6. El desarrollador corrige las tareas y cuando las sube a la PR, y esta se encarga de crear una *build* de Jenkins (véase en la sección 3.4) que puede completarse con éxito o no:
     - Si se completa con éxito: el desarrollador tendrá acceso a SonarQube para revisar code smells, vulnerabilidades, etc.
     - Si no se completa con éxito: Jenkins proporcionará retroalimentación del error para que el desarrollador pueda resolverlo (los errores pueden ser

debidos a falta de ficheros de configuración, baja cobertura de código, etc.)

7. Una vez se han corregido los cambios del supervisor, se revisa la inspección de código realizada por SonarQube y, si es necesario, se solucionan los code smells, bugs, vulnerabilidades, etc.
8. Cuando se corrige todo y la *build* de Jenkins de la última subida de código al repositorio se ha completado con éxito, se revisa el código otra vez y el supervisor aprueba la PR para que el desarrollador pueda fusionar su rama de desarrollo con la principal.

3.4. Herramientas utilizadas

En esta sección se listan las herramientas y programas que se han utilizado a lo largo del transcurso del proyecto:

- **Visual Studio 2022:** es una herramienta de desarrollo eficaz que permite realizar el ciclo de desarrollo software en un solo lugar. Es un entorno de desarrollo integrado (IDE) completo que se puede usar para desarrollar, editar, depurar y compilar el código. Visual Studio incluye compiladores, herramientas de control y análisis de código y una amplia selección de extensiones para adaptar el entorno al trabajo del usuario. Se ha usado para desarrollar código en .NET que, como se mencionó en la sección anterior, ha sido necesario para desarrollar el back-end del microservicio [6].

- **Postman:** es una herramienta utilizada para probar APIs, permitiendo al desarrollador enviar peticiones a servicios web y ver las respuestas de las peticiones. En este caso se ha utilizado principalmente para probar los endpoints del microservicio y las peticiones necesarias para hacerle funcionar.

- **Sourcetree:** es una herramienta que ofrece una interfaz de usuario para poder interactuar con repositorios de Git. Es una forma sencilla de empezar las los principiantes y más productiva para los experimentados. Desde el primer día en GMV recomendaban esta herramienta, ya que simplifica mucho las acciones que tiene que realizar el usuario en su repositorio [10].

- **SQL Server Management Studio:** es un entorno integrado para administrar cualquier infraestructura de SQL. Se usa para acceder a los componentes de SQL Server, Azure, etc. así como configurarlos, administrarlos y desarrollarlos. Para el proyecto ha sido necesario su uso, ya que al hacer las pruebas en local había que poblar las tablas de la base de datos y hacer consultas de prueba [21].

- **Astah UML:** es una herramienta que permite hacer todo tipo de diagramas UML. En este caso se ha usado para hacer diagramas de dominio, paquetes, etc.

- **Jira:** es un sistema de gestión de flujo de trabajo que permite al equipo del proyecto realizar un seguimiento del trabajo en cualquier escenario. El objetivo de Jira es que el trabajo se divida en tareas más pequeñas y manejables para asignarla al miembro del equipo adecuado y avanzar a través de un flujo de trabajo personalizable hasta que esté terminado [14].
- **Confluence:** es una herramienta utilizada principalmente para documentar software con multitud de macros para que la documentación sea limpia y ordenada. Es una herramienta que utilizan en la empresa GMV para documentar todo su software y procedimientos.
- **Microsoft Teams:** es una herramienta desarrollada por Microsoft cuyo objetivo es facilitar a las organizaciones y empresas la comunicación y la organización de sus equipos para que los miembros de estos estén informados, organizados y conectados. Microsoft Teams es la herramienta de comunicación con la que trabaja GMV.
- **Jenkins:** es un servidor open source para la integración continua (integraciones automáticas de un proyecto lo más a menudo posible para poder detectar fallos en la implementación y es utilizada para compilar y probar proyectos software de forma continua, por lo que facilita la integración de cambios en el proyecto y la publicación de nuevas versiones para entregar a los clientes. Además, para agilizar el proceso de revisión de calidad del código, GMV tiene SonarQube integrado con Jenkins, por lo que la integración continua y la inspección de código se hacen de una vez.
- **SonarQube:** es una herramienta de análisis de código abierto que permite inspeccionar y encontrar problemas de calidad, seguridad en el código, errores, vulnerabilidades y malas prácticas de programación. Además, te proporciona informes detallados y sugerencias sobre cómo mejorar la calidad de tu código.
- **NuGet:** es el gestor de paquetes software para .NET. NuGet ofrece la capacidad de subir tus propios paquetes y utilizar paquetes de otros. La Galería NuGet es el repositorio central de paquetes utilizado por los desarrolladores de aplicaciones y sistemas en .NET. GMV tiene su propio repositorio para paquetes internos de la empresa y se han utilizado principalmente para el desarrollo del microservicio [25].
- **Overleaf:** es una herramienta online para crear documentos PDF escritos en LaTeX. Se ha usado para escribir la memoria del proyecto.

3.5. Glosario

A continuación se muestra una tabla de abreviaturas con sus respectivos significados.

Abreviatura	Significado
MS	Microservicio
DPP	Dated Production Plan
PI	Planner Integrator
PR	Pull Request
CU	Caso de uso
ID	Identificador

Cuadro 3.1: Abreviaturas

Capítulo 4

Planificación

4.1. Planificación temporal

Como se explicó en la sección 3.1 del capítulo 3, la metodología de trabajo que se ha seguido para este proyecto ha sido Scrum con sprints típicamente de 2 semanas. [1]

A continuación se describe todo el trabajo realizado en cada uno de los sprints:

4.1.1. Sprint 0: 08/02/2024 - 21/02/2024

En la tabla 4.1 se muestra un resumen de las tareas que se han realizado durante el Sprint 0.

Tarea	Tiempo estimado	Tiempo invertido	Estado
Configuración del entorno de trabajo	12h	14h 30m	Completado
Lectura de documentación básica proporcionada por la empresa	5h	5h	Completada
Lectura de otros TFGs para tener en cuenta su estructura y contenido	6h	6h	Completada
Restructuración y adaptación de la plantilla de la memoria del TFG	2h	3h	Completada
TOTAL	**25h**	**28h 30m**	**Completado (100 %)**

Cuadro 4.1: Resumen del Sprint 0

[1] Los sprints 5 (subsección 4.1.5) y 6 (subsección 4.1.6) se han marcado con un **'*'** debido a que fueron alargados 1 semana más debido a cuestiones internas del equipo de desarrollo de la empresa

Este primer Sprint se ha dedicado principalmente a la preparación del material y la configuración de las herramientas necesarias como el Visual Studio, de SQL Server, etc. y a la adaptación e integración dentro del equipo y de la empresa, ya que este proyecto empezó cuando comenzaron mis prácticas de empresa.

4.1.2. Sprint 1: 21/02/2024 - 06/03/2024

En la tabla 4.2 se muestra un resumen de las tareas que se han realizado durante el Sprint 1.

Tarea	**Tiempo estimado**	**Tiempo invertido**	**Estado**
Diseño de la librería *MailService*	6h	5h 30m	Completada
Documentación inicial del microservicio *Planner Integrator*	5h	7h	Completada
Lectura de la documentación de las dependencias de .NET que se usarán en la librería	3h	3h 30m	Completada
Documentación de la memoria	10h	10h	Completada
TOTAL	**24h**	**26h**	**Completado (100 %)**

Cuadro 4.2: Resumen del Sprint 1

El Sprint 1 comenzó aprendiendo a hacer una buena documentación para ponerla a disposición de todo el personal de la empresa usando las diferentes macros y herramientas que proporciona Confluence. En este caso se documentó todo lo relativo al nuevo microservicio *Planner Integrator*, tanto su funcionamiento como sus entradas y salidas de datos.

Tras ello, se realizó un diseño detallado de la librería *MailService*, ya que es un recurso que podrán utilizar todos los desarrolladores de la empresa que lo necesiten para sus proyectos y más tarde su implementación, que será una buena oportunidad para ir aprendiendo y cogiendo soltura con la nueva tecnología de cara a la implementación del microservicio.

Por último, se consideró importante la lectura de la documentación de las dependencias y herramientas que se usarán en la implementación de la librería con el fin de hacer la implementación futura (que se incluye en el siguiente Sprint) más sencilla.

4.1.3. Sprint 2: 06/03/2024 - 20/03/2024

En la tabla 4.3 se muestra un resumen de las tareas que se han realizado durante el Sprint 2.

Tarea	Tiempo estimado	Tiempo invertido	Estado
Implementación de la librería MailService	12h	11h	Completado
Implementación de las pruebas de la librería MailService	12h	16h	Completada
Resolver problema: no pasa la build en Jenkins	3h	10h	Completada
Documentación de la memoria	10h	10h	Completada
TOTAL	**37h**	**47h**	**Completado (100 %)**

Cuadro 4.3: Resumen del Sprint 2

El Sprint 2 comenzó con la implementación [2] de la librería *MailService* como primera toma de contacto con el nuevo lenguaje de programación la cual no fue muy costosa, ya que su sintaxis y forma de trabajo no era muy dispar a otros lenguajes que se han cursado y aprendido durante la carrera.

Seguidamente, se prosiguió con los tests unitarios de la librería. Se implementaron los tests de error (lanzamiento de excepciones) y los tests *happy path* (consecución del éxito de la operación que se está probando). Se tardó más tiempo del estimado, ya que en el test del *happy path* era necesario usar *mocks* para simular el envío del correo por lo que se complicó más de lo esperado.

Una vez probada la funcionalidad de la librería y verificar todas sus pruebas, se procedió con la creación de la PR para poder sacar su primera versión y que los desarrolladores de la empresa pudiesen usar la librería. Como se explicó en el apartado de Git de la subsección 3.3 para poder fusionar la rama de desarrollo con la rama principal de un repositorio es necesario tener aprobación de los supervisores de la PR y que la *build* de Jenkins se haya completado con éxito. Las *builds* de Jenkins no se completaron con éxito debido a la falta de ciertos ficheros de configuración en el proyecto lo que llevó a la demora de la salida de la primera versión de la librería.

4.1.4. Sprint 3: 20/03/2024 - 03/04/2024

En la tabla 4.4 se muestra un resumen de las tareas que se han realizado durante el Sprint 3.

[2] Todas las tareas de implementación (incluidas las que se mencionan más adelante) incluyen el desarrollo del código y pruebas en la máquina local.

Tarea	Tiempo estimado	Tiempo invertido	Estado
Corregir code smells de la librería	2h	1h 30m	Completado
Resolver y arreglar las correcciones de la PR para sacar la primera version de la librería	5h	10h	Completada
Preparación de la plantilla y el repositorio para implementar el nuevo microservicio	3h	2h	Completada
Diseño de la parte de cambios online para el nuevo microservicio	15h	12h	Completada
Documentación de la memoria	10h	10h	Completada
TOTAL	**22h**	**28h 30m**	**Completado (100 %)**

Cuadro 4.4: Resumen del Sprint 3

El Sprint 3 comenzó con la corrección de los code smells de la librería, ya que el problema que surgió en el anterior Sprint ya fue resuelto y se pudo seguir con el trabajo.

Seguidamente, se prosiguió con la realización y corrección de las tareas de la PR. Esta tarea se demoró más de lo estimado, ya que los supervisores de la PR querían hacer el test del *happy path* más sencillo y legible, por lo que se decidió hacer un cambio de diseño en la librería y de paso hacerla más modular. Esto llevó a cambiar el test mencionado anteriormente y a un desacoplamiento en la clase de configuración de la librería. Una vez se hicieron todos los cambios propuestos en la librería, la PR fue aprobada y se hizo un *merge* de la rama de desarrollo personal a la rama principal del repositorio y con ello la publicación de la primera versión de la librería.

Una vez terminada la primera parte de este proyecto, la librería, se preparó todo para poder implementar el nuevo microservicio, en concreto se usó una de las plantillas que proporciona GMV para la implementación de nuevos microservicios y el repositorio en Bitbucket donde se va a depositar el código desarrollado.

El diseño de los cambios online del nuevo microservicio se hizo a partir de la plantilla que se generó en la anterior tarea para poder ahorrar tiempo y complejidad (ya que la plantilla nos proporciona la estructura básica del microservicio).

4.1.5. Sprint 4: 03/04/2024 - 24/04/2024 *

En la tabla 4.5 se muestra un resumen de las tareas que se han realizado durante el Sprint 4.

El Sprint 4 comenzó con la implementación inicial del microservicio que abarca los cambios online de vehículos y sus pruebas unitarias y de integración. Se estimaron 10 horas

Tarea	Tiempo estimado	Tiempo invertido	Estado
Implementación de la asignación online de vehículos	10h	15h	Completada
Implementación de la desasignación online de vehiculos	10h	9h	Completado
Implementación del reemplazo online de vehículos	10h	12h	Completada
Implementación de las pruebas de error de los cambios online de vehículos	10h	9h	Completada
Documentación de la memoria	10h	10h	Completada
TOTAL	**50h**	**55h**	**Completado (100 %)**

Cuadro 4.5: Resumen del Sprint 4

para cada tarea de implementación del microservicio que equivalen a 2 días laborables de prácticas.

La primera tarea fue la implementación de la asignación online de vehículos en la que tuve soporte del tutor de la empresa para avanzar en la fase inicial y saber como implementar las demás tareas.

Tras implementar la funcionalidad de los cambios online, se implementaron los tests de que lanzan excepciones de las 3 tareas anteriores. Los test *happy path* se decidió moverlos al siguiente sprint para poder corregir errores que se detectaron en las pruebas locales.

4.1.6. Sprint 5: 24/04/2024 - 15/05/2024 *

En la tabla 4.6 se muestra un resumen de las tareas que se han realizado durante el Sprint 5.

Este sprint comenzó con la implementación de los tests *happy path* de los cambios online de vehículos que, como se dijo en la subsección del anterior sprint, se aplazó para este. Esta tarea se alargó más de lo debido, ya que contienen mucha complejidad con la alta cantidad de *mocks* que hay que utilizar y se tuvieron que arreglar varios bugs para que las pruebas pasasen de forma exitosa.

Una vez implementados los tests se prosiguió con la implementación de los cambios online de conductores. Se tomaron de referencia los cambios online de los vehículos implementados en el anterior sprint por lo que no se han tenido demoras en estas tareas.

Tarea	Tiempo estimado	Tiempo invertido	Estado
Implementación de las pruebas *happy path* de los cambios online de vehiculos	10h	12h	Completada
Implementación de la asignación online de conductores	10h	7h	Completada
Implementación de la desasignación online de conductores	10h	8h	Completada
Implementación del reemplazo online de conductores	10h	10h	Completada
Documentación de la memoria	10h	10h	Completada
TOTAL	**50h**	**47h**	**Completado (100 %)**

Cuadro 4.6: Resumen del Sprint 5

4.1.7. Sprint 6: 15/05/2024 - 29/05/2024

En la tabla 4.7 se muestra un resumen de las tareas que se han realizado durante el Sprint 6.

Tarea	Tiempo estimado	Tiempo invertido	Estado
Implementación de las pruebas happy path de los cambios online de conductores	10h	12h	Completada
Corregir los code smells y tareas de la PR	5h	3h	Completada
Implementación de los tests de la API y sus controladores	10h	15h	Completada
Documentación de la memoria	10h	10h	Completada
TOTAL	**35h**	**38h**	**Completado (100 %)**

Cuadro 4.7: Resumen del Sprint 6

Este sprint comenzó con la implementación de los tests *happy path* de los cambios online de conductores y corrigiendo a la vez los code smells que iban surgiendo a medida que avanzaba el desarrollo.

Tras ello se procedió con la implementación de los tests de integración del microservicio para probar el despliegue, las conexiones y el lanzamiento de peticiones a los diferentes endpoints.

Una vez implementados los tests de integración se volvió a revisar el código para poner en producción el microservicio con los cambios online y así permitir a los clientes poderse integrar con él.

4.1.8. Sprint 7: 29/05/2024 - 12/06/2024

En la tabla 4.8 se muestra un resumen de las tareas que se han realizado durante el Sprint 7.

Tarea	Tiempo estimado	Tiempo invertido	Estado
Tomar decisiones respecto a las futuras funcionalidades	1h	2h	Completada
Diseñar la parte común de carga de vehículos y conductores	12h	10h	Completada
Leer la documentación y el código de un microservicio de referencia para la carga.	5h	3h	Completada
Documentación de la memoria	10h	10h	Completada
TOTAL	**28h**	**25h**	**Completado (100 %)**

Cuadro 4.8: Resumen del Sprint 7

El Sprint 7 comenzó con una pequeña reunión con el tutor de la empresa y más personal de la empresa para tomar decisiones sobre las futuras funcionalidades del microservicio. Finalmente, se decidió implementar la carga de vehículos y de conductores al sistema Suite para los clientes que lo necesitaran y la importación de horarios y topologías (estas dos últimas quedarán fuera del alcance de este proyecto).

Para realizar el diseño de la carga en el sistema tanto de vehículos como de conductores es necesario tener en cuenta que esta nueva funcionalidad se divide en 2 partes: una parte común (que es la que entrará dentro de este TFG) y una parte específica (esta parte no entrará por falta de tiempo). En el capítulo 6 se especificará la función de cada una de las partes.

Por último, antes de comenzar con la implementación se decidió consultar la documentación y el código de otro microservicio que se tomará de referencia para implementar la carga de vehículos y conductores.

4.1.9. Sprint 8: 12/06/2024 - 26/06/2024

En la tabla 4.9 se muestra un resumen de las tareas que se han realizado durante el Sprint 8.

Tarea	Tiempo estimado	Tiempo invertido	Estado
Implementacion de la parte común de carga de vehículos	12h	10h	Completada
Implementación de los tests de la parte común de carga de vehículos.	5h	3h	Completada
Implementacion de la parte común de carga de conductores	2h	1h	Aplazada
Implementación de los tests de la parte común de carga de conductores.	10h	10h	Aplazada
Documentación de la memoria	10h	10h	Completada
TOTAL	**39h**	**34h**	**Completado (60 %)**

Cuadro 4.9: Resumen del Sprint 8

Este sprint ha comenzado con la implementación de la carga de vehículos y sus tests unitarios. También se tomó la decisión de aplazar la carga de conductores, ya que se acerca la fecha final del proyecto y se consideró centrarse en la documentación de la memoria y corregir algunos fallos que se detectaron en la carga de vehículos.

4.1.10. Resumen de la ejecución del proyecto

Como se puede observar en la tabla 4.10 el tiempo invertido es ligeramente mayor que el estimado debido principalmente a los riesgos que se describen en las tablas 4.12 y 4.13, ya que han sido los riesgos del plan de control del proyecto que más se han materializado y que más han limitado el transcurso del proyecto. Esto es debido a que en las tareas de la asignatura cursada y tareas de la empresa surgieron algunos inconvenientes y se alargaron más de lo esperado y dichos retrasos se materializan en retrasos en las tareas del proyecto.

Como se puede observar los *sprints* que han tenido más carga de trabajo son los *sprints* 2, 4 y 5 los cuales se pueden ver las tablas 4.3, 4.5 y 4.6 respectivamente. Todos ellos corresponden a la implementación de las diferentes partes del proyecto y sus correspondientes pruebas.

4.2. Plan de control del proyecto

En esta sección se analizan los riesgos existentes para el proyecto. En la tabla 4.11 se muestra la exposición al riesgo respecto del impacto y la probabilidad.

Sprint	**Tiempo estimado**	**Tiempo invertido**
Sprint 0	25h	28h 30m
Sprint 1	24h	26h
Sprint 2	37h	47h
Sprint 3	22h	28h 30m
Sprint 4	50h	55h
Sprint 5	50h	47h
Sprint 6	35h	38h
Sprint 7	28h	25h
Sprint 8	39h	34h
Total	**310h**	**329h**

Cuadro 4.10: Resumen del tiempo estimado e invertido en cada *sprint*

Imp./Prob.	Baja	Media	Alta
Bajo	Bajo	Bajo	Medio
Medio	Bajo	Medio	Alto
Alto	Medio	Alto	Alto

Cuadro 4.11: Representación de la exposición al riesgo

En este proyecto se han identificado 8 riesgos los cuales pueden observarse en las tablas 4.12, 4.13, 4.14, 4.15, 4.16, 4.17, 4.18 y 4.19.

Identificador	R01
Nombre	Estudio y realización de la parte práctica de la asignatura optativa
Descripción	La planificación y fechas del proyecto se ven afectadas porque la asignatura que se cursa toma más tiempo del esperado.
Probabilidad	Alta
Impacto	Medio
Exposición	Alta
Plan de mitigación	Planificar estrictamente los trabajos de la asignatura y reestructurar el plan de proyecto cuanto antes si es necesario
Plan de contingencia	Establecer prioridades y un horario estricto para la realización de las actividades de la asignatura

Cuadro 4.12: Riesgo - R01: Estudio y realización de la parte práctica de la asignatura optativa

Identificador	R02
Nombre	Realización de tareas adicionales al TFG en la empresa
Descripción	Se tiene que dejar parado el transcurso del proyecto debido a la asignación de tareas ajenas al TFG en la empresa.
Probabilidad	Alta
Impacto	Medio
Exposición	Alta
Plan de mitigación	Planificar estrictamente las tareas adicionales de la empresa y reestructurar el plan de proyecto cuanto antes si es necesario
Plan de contingencia	Establecer prioridades en las tareas de la empresa como del proyecto

Cuadro 4.13: Riesgo - R02: Realización de tareas adicionales al TFG en la empresa

Identificador	R03
Nombre	Falta de familiarización con las tecnologías y herramientas
Descripción	Las tecnologías y herramientas a usar son poco familiares y hay que invertir más tiempo en formación y aprendizaje
Probabilidad	Media
Impacto	Medio
Exposición	Media
Plan de mitigación	Aprovechar los cursos que proporciona la empresa para poder coger soltura con las tecnologías lo antes posible
Plan de contingencia	Invertir más tiempo en formación o hacer horas extras para no comprometer las fechas del proyecto

Cuadro 4.14: Riesgo - R03: Falta de familiarización con las tecnologías y herramientas

Identificador	R04
Nombre	Caducidad de licencias
Descripción	Caducidad de las licencias de ciertas herramientas que se necesitan para seguir con el desarrollo del proyecto
Probabilidad	Alta
Impacto	Bajo
Exposición	Media
Plan de mitigación	Comprobar día a día las fechas de expiración de las licencias para avisar a la empresa de su caducidad
Plan de contingencia	Comunicar al soporte técnico de la empresa con la mayor brevedad posible

Cuadro 4.15: Riesgo - R04: Caducidad de licencias

Identificador	R05
Nombre	Alta complejidad de la arquitectura del sistema de la empresa.
Descripción	La arquitectura del sistema de GMV está compuesta por varios microservicios que tienen mucha funcionalidad y complejidad añadida y requiere de tiempo adicional comprender para qué sirve cada uno y si se necesita comunicarse con él o no.
Probabilidad	Alta
Impacto	Medio
Exposición	Alta
Plan de mitigación	Investigar y consultar documentación proporcionada por la empresa.
Plan de contingencia	Establecer prioridades en las tareas y/o ampliar los tiempos estimados del proyecto

Cuadro 4.16: Riesgo - R05: Alta complejidad de la arquitectura del sistema de la empresa.

Identificador	R06
Nombre	Contraer una enfermedad o cualquier problema de salud
Descripción	Los tiempos y fechas del proyecto se pueden ver afectados si el desarrollador contrae una enfermedad.
Probabilidad	Media
Impacto	Medio
Exposición	Media
Plan de mitigación	Hacer una planificación del proyecto añadiendo holgura a las actividades en caso de que el desarrollador necesite reposo.
Plan de contingencia	Replanificar el proyecto y sus fechas.

Cuadro 4.17: Riesgo - R06: Contraer una enfermedad o cualquier problema de salud

Identificador	R07
Nombre	Requisitos funcionales mal definidos
Descripción	Al analizar un requisito funcional a implementar, puede comprenderse de manera errónea y llevar a cabo un desarrollo no útil.
Probabilidad	Baja
Impacto	Alto
Exposición	Media
Plan de mitigación	Verificar y revisar con el equipo que se han entendido bien los requisitos y dejar claro el trabajo que se ha de realizar.
Plan de contingencia	Realizar un análisis del esfuerzo que supone corregir la implementación y priorizar tareas.

Cuadro 4.18: Riesgo - R07: Requisitos funcionales mal definidos

Identificador	R08
Nombre	Cambios en los requisitos
Descripción	A lo largo del desarrollo puede haber cambios en los requisitos, ya que este proyecto será útil (a largo plazo) para las integraciones con muchos de los clientes de GMV por lo que es más probable que haya cambios.
Probabilidad	Media
Impacto	Alto
Exposición	Alta
Plan de mitigación	Usar una metodología ágil para congelar los requisitos durante los sprints.
Plan de contingencia	Replanificar el proyecto y sus fechas.

Cuadro 4.19: Riesgo - R08: Cambios en los requisitos

4.3. Plan de presupuestos

En esta sección se describirá el plan presupuestario del proyecto. Se calcularán tanto los costes humanos como los materiales desde el principio hasta el final del proyecto. El presupuesto engloba el coste de contratación de un trabajador (en este caso, un desarrollador back-end junior), de todas las licencias de software necesarias, equipo de trabajo (ordenadores portátiles entre otros) y el consumo de electricidad.

4.3.1. Coste humano

El puesto que ha requerido este proyecto ha sido *desarrollador de back-end junior* y tiene un sueldo medio de 25.600€ brutos al año [31]. Con un salario anual de 25.600€ en Castilla y León, se aplican 5.311€ de impuestos (20.74 %). Es decir, el salario neto será de **20.289€ al año**, que se traducen en **1.691€ al mes**. En España la jornada laboral son 40 horas semanales o 160 horas mensuales [33] por lo que, para calcular el sueldo por hora:

$$\frac{1{,}691e}{mes} * \frac{1mes}{160horas} = 10,56e/hora$$

El proyecto ha durado 329 horas, por lo que el coste humano es de **3474,24 €.**

4.3.2. Hardware, software y otros

[3] Respecto al hardware usado para desarrollar el proyecto, GMV proporcionó el ordenador portátil *Lenovo ThinkPad P14s Gen 4* cuyo precio es de 1948.01€ [32]. La vida útil media de un portátil es aproximadamente 4 años que se traduce en 48 meses [35]. El precio aproximado por mes resulta en 40.58 €/mes por lo que el ordenador portátil durante el proyecto ha tenido un coste de **243.48€**.

Para la redacción de la memoria se ha utilizado con el ordenador portátil *HP 15S-eq2088ns* cuyo precio es de 649 € [18]. Como se mencionó en la subsección 4.3.2 la vida útil de un ordenador portátil es de 48 meses por lo que el coste aproximado del equipo sería de 13.52 €/mes. Entonces para cubrir los 6 meses del proyecto, el coste amortizado será de **81.12 €**.

Respecto al software usado, se ha usado la licencia de pago de Microsoft 365 Enterprise la cual incluye Microsoft Teams, que costó 11,70€ al mes [22] por lo que el total ha resultado en **70.20€**. También se ha utilizado la licencia de Visual Studio 2022 Professional la cual cuesta 45€ al mes [26], es decir, **270€** en total. Por último, la aplicación Astah Professional también necesita una licencia de pago para poder usarla, la cual ha costado 11.99€ [28] al mes y un total de **71.94€**.

Por último, hay que tener en cuenta el gasto de electricidad durante el tiempo de trabajo. El consumo medio de electricidad de un ordenador portátil es de 0.11 kWh

[3] El proyecto ha durado aproximadamente 5 meses, pero se tendrá en cuenta el caso de entregar el proyecto en la convocatoria extraordinaria, por lo que las licencias y material se usarán durante un mes más, es decir, durante 6 meses

[15]. El tiempo de duración del proyecto ha sido de 329 horas por lo que el consumo de electricidad durante la realización del proyecto será de 33 kWh. Dado que el precio de la electricidad durante el periodo del proyecto ha sido de 0.05427 €/kWh por lo que el coste total de consumo de electricidad ha sido de **1.79 €**.

4.3.3. Presupuesto total

En la tabla 4.20 se muestra un resumen de los costes que compone el presupuesto proyecto:

Concepto	Precio parcial	Horas	Precio total
Trabajo realizado por el desarrollador	10.56 €/hora	329h	3474.24 €
Ordenador portátil del desarrollador	40.58 €/mes	6 meses	243.48 €
Licencia de Microsoft 365 Enterprise	11.70 €/mes	6 meses	70.20 €
Licencia de Visual Studio 2022	45 €/mes	6 meses	270 €
Licencia de Astah Professional	11.99 €/mes	6 meses	71.94 €
Ordenador portátil para redactar la memoria	13.52 €/mes	6 meses	81.12 €
Consumo de electricidad	0.05427 €/kWh	33 kWh	1.79 €
		TOTAL	**4212,77 €**

Cuadro 4.20: Resumen del presupuesto del proyecto.

Parte II

Documentación técnica

Capítulo 5

Análisis

5.1. Requisitos

En este capítulo se describen los requisitos del proyecto.

5.2. Requisitos funcionales

Para este proyecto se van a dividir los requisitos funcionales en 2 partes: los cambios online y los cambios planificados:

- Requisitos funcionales relacionados con los cambios online:
  - *CU-01*. Asignación de vehículos. (Véase en la tabla 5.1)
  - *CU-02*. Desasignación de vehículos. (Véase en la tabla 5.2)
  - *CU-03*. Reemplazo de vehículos. (Véase en la tabla 5.3)
  - *CU-04*. Asignación de conductores. (Véase en la tabla 5.4)
  - *CU-05*. Desasignación de conductores. (Véase en la tabla 5.5)
  - *CU-06*. Reemplazo de conductores. (Véase en la tabla 5.6)
- Requisitos funcionales relacionados con los cambios planificados:
  - *CU-07*. Carga planificada de vehículos. (Véase en la tabla 5.7)
  - *CU-08*. Carga planificada de conductores. (Véase en la tabla 5.8)
- Requisito funcional adicional:
  - *CU-09*. Enviar correos electrónicos (Véase en la tabla 5.9)

5.2.1. Cambios online de vehículos

Título	CU-01. Asignar vehículos
Actor	Planificador externo
Descripción	El actor envía los datos correspondientes para que el vehículo deseado pueda ser asignado
Precondiciones	PRE-1. El actor debe estar identificado en el sistema Suite.
Postcondiciones	POST-1. El vehículo correspondiente quedará asignado.
Flujo normal	1. El actor solicita la asignación online de un vehículo. 2. El sistema comprueba que el actor está autenticado en el sistema Suite. 3. El sistema asigna el vehículo especificado por el actor.
Excepciones	2.1. El actor no está autenticado en el sistema Suite o las credenciales son incorrectas por lo que el sistema cancela la operación y el caso de uso se queda sin efecto. 2.2. Los datos proporcionados por el actor no son correctos por lo que el sistema cancela la operación y el caso de uso queda sin efecto.

Cuadro 5.1: CU-01. Asignar vehículos

Título	CU-02. Desasignar vehículos
Actor	Planificador externo
Descripción	El actor envía los datos correspondientes para que el vehículo deseado pueda ser desasignado.
Precondiciones	PRE-1. El actor debe estar identificado en el sistema Suite.
Postcondiciones	POST-1. El vehículo correspondiente quedará desasignado.

Flujo normal	1. El actor solicita la desasignación online de un vehículo. 2. El sistema comprueba que el actor está autenticado en el sistema Suite. 3. El sistema desasigna el vehículo especificado por el actor.
Excepciones	2.1. El actor no está autenticado en el sistema Suite o las credenciales son incorrectas por lo que el sistema cancela la operación y el caso de uso se queda sin efecto. 2.2. Los datos proporcionados por el actor no son correctos por lo que el sistema cancela la operación y el caso de uso queda sin efecto.

Cuadro 5.2: CU-02. Desasignar vehículos

Título	CU-03. Reemplazar vehículos
Actor	Planificador externo
Descripción	El actor envía los datos correspondientes para que el vehículo deseado pueda ser reemplazado.
Precondiciones	PRE-1. El actor debe estar identificado en el sistema Suite.
Postcondiciones	POST-1. El vehículo correspondiente reemplazará a otro tambien especificado.
Flujo normal	1. El actor solicita el reemplazo online de un vehículo por otro. 2. El sistema comprueba que el actor está autenticado en el sistema Suite. 3. El sistema reemplaza un vehículo por otro, ambos especificados por el actor.

Excepciones	2.1. El actor no está autenticado en el sistema Suite o las credenciales son incorrectas por lo que el sistema cancela la operación y el caso de uso se queda sin efecto. 2.2. Los datos proporcionados por el actor no son correctos por lo que el sistema cancela la operación y el caso de uso queda sin efecto.

Cuadro 5.3: CU-03. Reemplazar vehículos

En la figura 5.1 se puede observar el diagrama de casos de uso relacionados con los cambios online de vehículos.

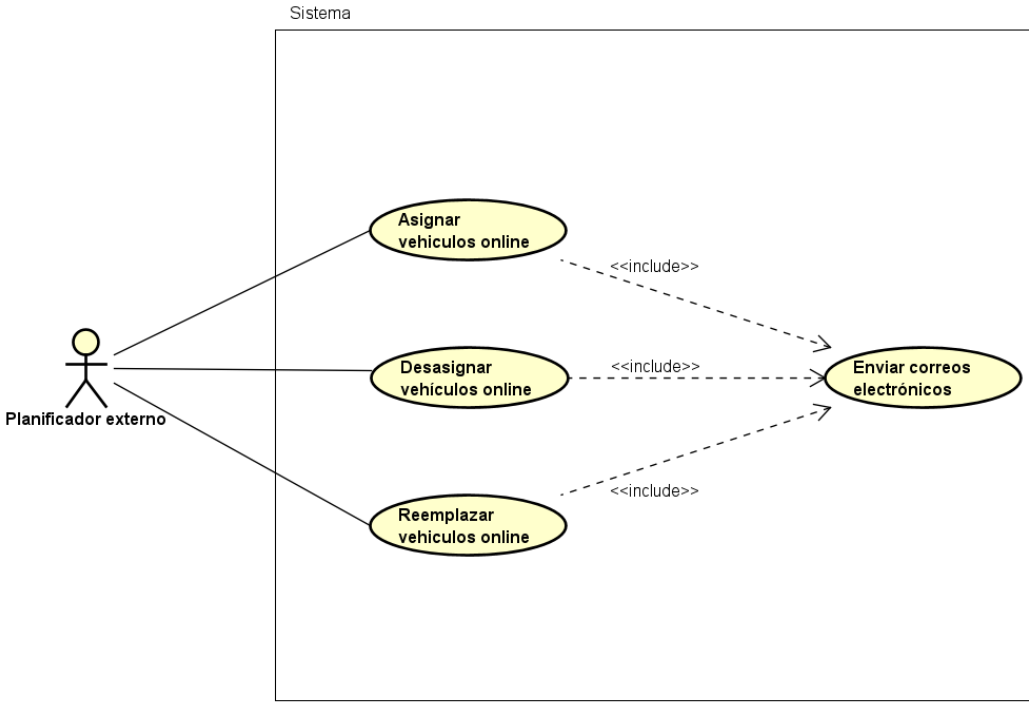

Figura 5.1: Diagrama de casos de uso relacionados con los cambios online de vehículos

5.2.2. Cambios online de conductores

Título	CU-04. Asignar conductores
Actor	Planificador externo
Descripción	El actor envía los datos correspondientes para que el conductor deseado pueda ser asignado
Precondiciones	PRE-1. El actor debe estar identificado en el sistema Suite.
Postcondiciones	POST-1. El conductor correspondiente quedará asignado.
Flujo normal	1. El actor solicita la asignación online de un conductor. 2. El sistema comprueba que el actor está autenticado en el sistema Suite. 3. El sistema asigna el conductor especificado por el actor.
Excepciones	2.1. El actor no está autenticado en el sistema Suite o las credenciales son incorrectas por lo que el sistema cancela la operación y el caso de uso se queda sin efecto. 2.2. Los datos proporcionados por el actor no son correctos por lo que el sistema cancela la operación y el caso de uso queda sin efecto.

Cuadro 5.4: CU-04. Asignar conductores

Título	CU-05. Desasignar conductores
Actor	Planificador externo
Descripción	El actor envía los datos correspondientes para que el conductor deseado pueda ser desasignado.
Precondiciones	PRE-1. El actor debe estar identificado en el sistema Suite.
Postcondiciones	POST-1. El conductor correspondiente quedará desasignado.

Flujo normal	1. El actor solicita la desasignación online de un conductor. 2. El sistema comprueba que el actor está autenticado en el sistema Suite. 3. El sistema desasigna el conductor especificado por el actor.
Excepciones	2.1. El actor no está autenticado en el sistema Suite o las credenciales son incorrectas por lo que el sistema cancela la operación y el caso de uso se queda sin efecto. 2.2. Los datos proporcionados por el actor no son correctos por lo que el sistema cancela la operación y el caso de uso queda sin efecto.

Cuadro 5.5: CU-05. Desasignar conductores

Título	CU-06. Reemplazar conductores
Actor	Planificador externo
Descripción	El actor envía los datos correspondientes para que el conductor deseado pueda ser reemplazado.
Precondiciones	PRE-1. El actor debe estar identificado en el sistema Suite.
Postcondiciones	POST-1. El conductor correspondiente reemplazará a otro tambien especificado.
Flujo normal	1. El actor solicita el reemplazo online de un conductor por otro. 2. El sistema comprueba que el actor está autenticado en el sistema Suite. 3. El sistema reemplaza un conductor por otro, ambos especificados por el actor.

Excepciones	2.1. El actor no está autenticado en el sistema Suite o las credenciales son incorrectas por lo que el sistema cancela la operación y el caso de uso se queda sin efecto. 2.2. Los datos proporcionados por el actor no son correctos por lo que el sistema cancela la operación y el caso de uso queda sin efecto.

Cuadro 5.6: CU-06. Reemplazar conductores

En la figura 5.2 se puede observar el diagrama de casos de uso relacionados con los cambios online de conductores.

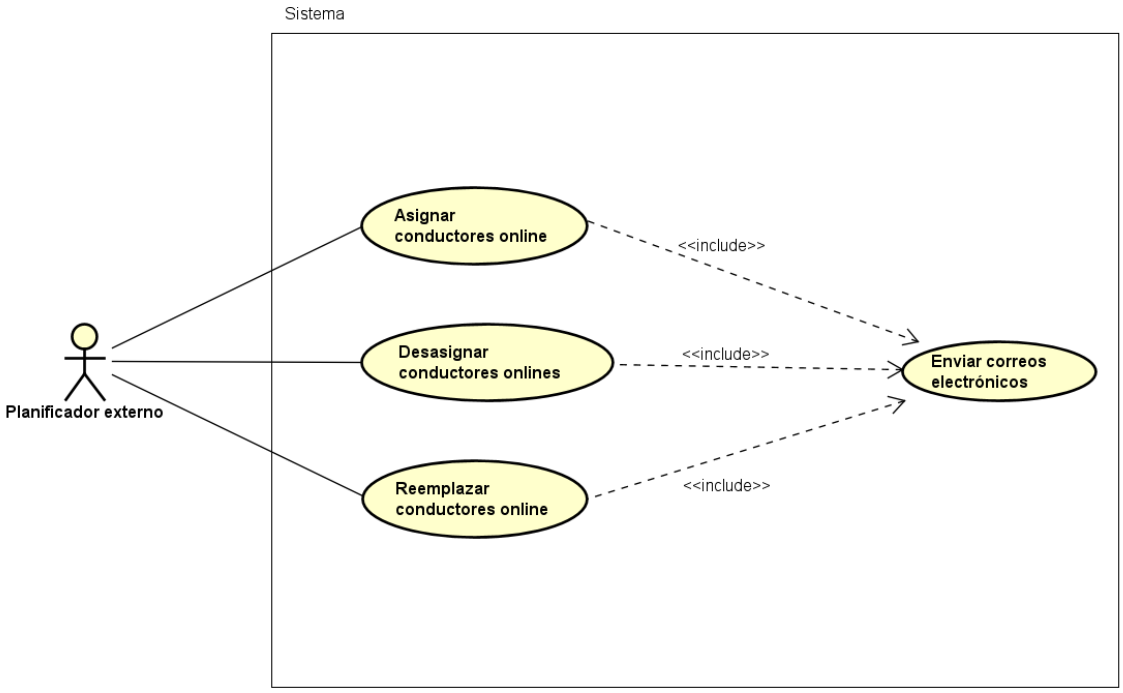

Figura 5.2: Diagrama de casos de uso relacionados con los cambios online de conductores

5.2.3. Cambios planificados

Título	CU-07. Carga planificada de vehículos
Actor	Planificador externo
Descripción	El actor prepara todos los dias los vehículos que estarán activos para el dia siguiente y el microservicio se encargará de cargarlos automáticamente.
Precondiciones	PRE-1. El actor debe estar identificado en el sistema Suite.
Postcondiciones	POST-1. Todos los vehículos proporcionados por el cliente en sus sistema quedarán cargados en el sistema Suite para ese día.
Flujo normal	1. El actor proporciona los datos de los vehículos en su sistema. 2. El sistema solicita los datos que ha proporcionado el actor en su sistema. 3. El sistema carga en el sistema Suite los vehículos correspondientes para ese día.
Excepciones	2.1. El sistema no está autenticado en el sistema del cliente o las credenciales son incorrectas por lo que el sistema cancela la operación y el caso de uso queda sin efecto.

Cuadro 5.7: CU-07. Carga planificada de vehículos

Título	CU-08. Carga planificada de conductores
Actor	Cliente
Descripción	El actor prepara todos los dias los conductores que estarán activos para el dia siguiente y el microservicio se encargará de cargarlos automáticamente.
Precondiciones	PRE-1. El actor debe estar identificado en el sistema Suite.
Postcondiciones	POST-1. Todos los conductores proporcionados por el actor en su sistema quedarán cargados en el sistema Suite para ese día.

Flujo normal	1. El actor proporciona los datos de los conductores en su sistema. 2. El sistema solicita los datos que ha proporcionado el actor en su sistema. 3. El sistema carga en el sistema Suite los conductores correspondientes para ese día.
Excepciones	2.1. El sistema no está autenticado en el sistema del cliente o las credenciales son incorrectas por lo que el sistema cancela la operación y el caso de uso queda sin efecto.

Cuadro 5.8: CU-08. Carga planificada de conductores

En la imagen 5.3 se puede observar el diagrama de casos de uso relacionados con los cambios planificados tanto de vehículos como de conductores.

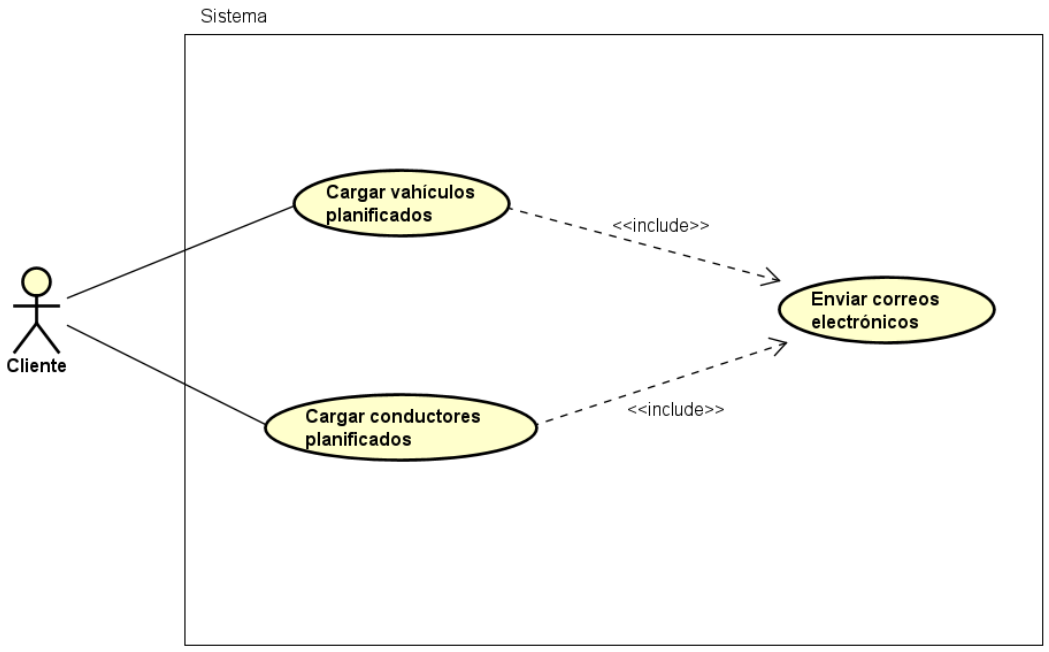

Figura 5.3: Diagrama de casos de uso relacionados con los cambios planificados

5.2.4. Requisitos funcionales adicionales

Título	CU-09. Enviar correo electrónico
Actor	Cliente
Descripción	El actor enviará un correo automáticamente cada vez que se realicen uno de los 8 casos de uso anteriores.
Precondiciones	PRE-1. El actor debe estar identificado en el sistema Suite.
Postcondiciones	POST-1. Todos los conductores proporcionados por el actor en su sistema quedarán cargados en el sistema Suite para ese día.
Flujo normal	1. El actor proporciona los datos necesarios para enviar el correo electrónico. 2. El sistema envía el correo electrónico al remitente.
Excepciones	2.1. Las credenciales proporcionadas por el cliente son incorrectas por lo que el sistema cancela la operación y el caso de uso queda sin efecto.

Cuadro 5.9: CU-09. Enviar correo electrónico

5.3. Atributos de calidad

En esta sección se analizan los posibles indicadores de calidad del proyecto desarrollado. El principal indicador de calidad es la **robustez** que aporta al sistema Suite. Como se explicó en el capítulo 1, el propósito de este proyecto es proporcionar al sistema Suite de GMV una capa más de procesado de datos de terceros para que el microservicio DPP (que es una parte muy crítica del sistema) no tenga que adaptarse y tener que cambiar sus interfaces cada vez que tiene que recibir datos de un planificador externo nuevo. También contribuye a una mejor **seguridad**, ya que esta capa adicional que se añade al sistema comprueba la integridad de los datos para ahorrar problemas muy graves cuando lleguen al sistema Suite.

Capítulo 6

Diseño

Tanto el diseño y la implementación se divide en 2 partes.

- **Librería *MailService***: es una librería que se encargará de enviar los correos de para informar al cliente del error o el éxito de la operación que ha solicitado. Se diseñó e implementó al principio del proyecto, ya que aparte de servir para este proyecto, también puede servir para cualquier proyecto de GMV que lo necesite. Además, me sirvió para poder familiarizarme con .NET y C# antes de empezar con el diseño e implementación del microservicio.
- **Microservicio *Planner Integrator***: es la 2ª parte del proyecto y la más compleja y extensa, por lo que la fase de diseño se ha tomado con detalle y de forma muy cuidadosa.

6.1. Patrones de diseño utilizados

A lo largo del desarrollo del proyecto se han utilizado distintos patrones de diseño. En esta sección se detallarán los principales patrones utilizados.

6.1.1. Patrón fachada

El patrón fachada es un patrón de diseño estructural que proporciona una interfaz simplificada de acceso a un framework, librería, o cualquier otro grupo complejo de clases. Su uso viene motivado por la necesidad de estructurar un proyecto software y reducir su complejidad con la división en subsistemas, minimizando las comunicaciones y dependencias entre estos. [16]. En la figura 6.1 se puede observar la estructura del patrón.

Uso del patrón en el proyecto

Se ha usado principalmente para la comunicación entre microservicios y proveer un punto de comunicación con el microservicio a las clases interesadas en ello. Principalmente en la comunicación de los controladores del microservicio con los servicios que ofrecen la

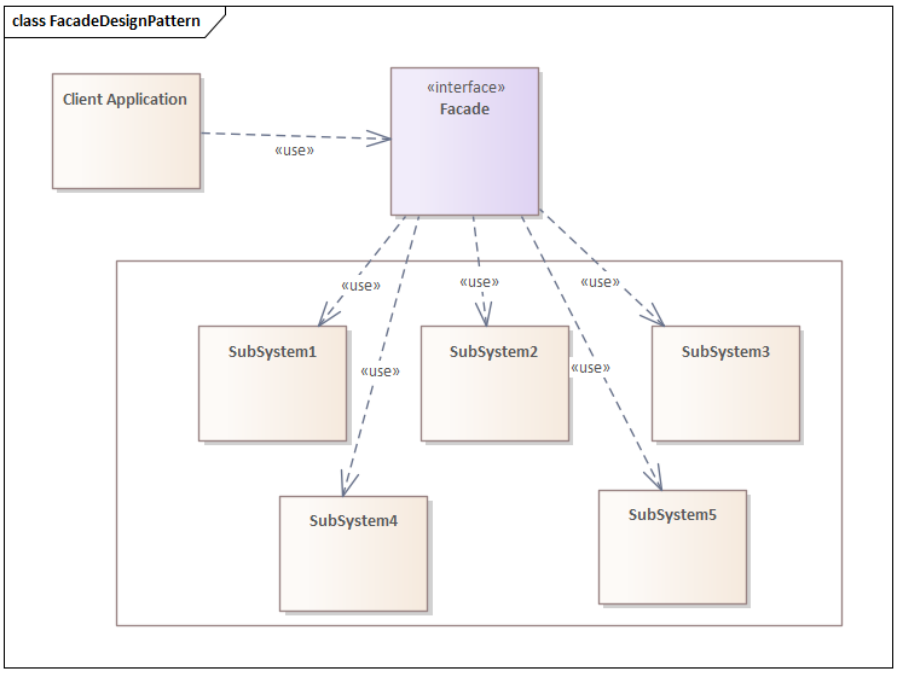

Figura 6.1: Patrón fachada [19]

funcionalidad al *endpoint* que ha sido llamado y a la hora de usar la librería *MailService* para enviar un correo electrónico.

6.1.2. Principio de inversión de dependencias

El Principio de inversión de dependencia es una forma de desacoplar módulos de software y dependencias entre módulos. Este patrón invierte las relaciones de dependencia desde los módulos de alto nivel a los módulos de bajo nivel, lo que hace que los módulos de alto nivel sean independientes de los detalles de implementación del módulo de bajo nivel [12]. El principio establece:

1. Los módulos de alto nivel no deben depender de los módulos de bajo nivel. Ambos deberían depender de abstracciones (interfaces).

2. Las abstracciones no deberían depender de los detalles. Los detalles (implementaciones concretas) deben depender de abstracciones.

En la figura 6.2 se puede observar un ejemplo de la estructura del principio de inversión de dependencias.

Uso del patrón en el proyecto

Se ha usado principalmente para la llamada a otros microservicios *core* como Operational Context o Dated Production Plan.

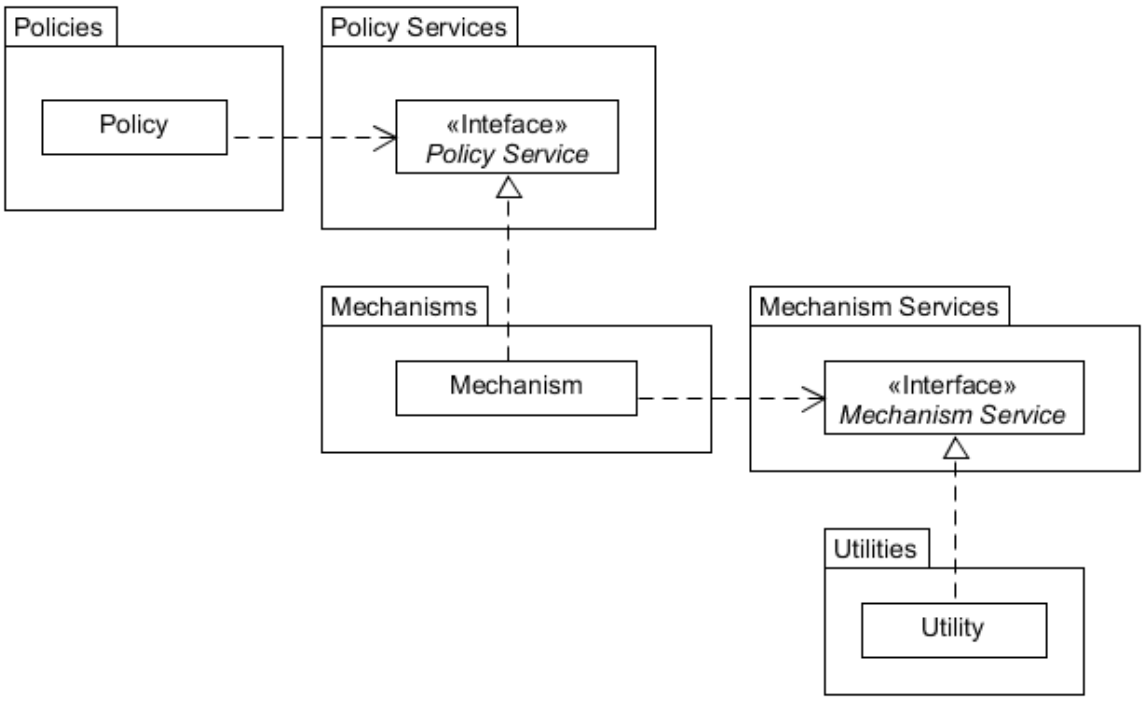

Figura 6.2: Principio de inversión de dependencias [12]

6.2. Diseño del proyecto

En la figura 6.3 se puede observar un esquema simplificado del funcionamiento global y las comunicaciones del microservicio:

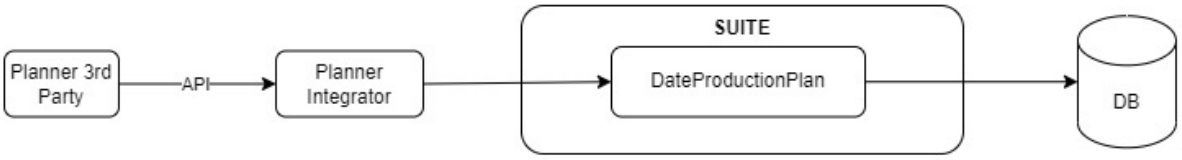

Figura 6.3: Arquitectura lógica del microservicio.

El flujo y funcionamiento es sencillo:

1. El planificador externo del cliente envía los datos a través de peticiones Http a la API del microservicio PI

2. El PI procesa los datos y se comunica con DPP

3. Según los datos que reciba del PI, DPP asignará, desasignará o reemplazará vehículos o conductores y hará cambios en la BBDD del microservicio.

4. Una vez se haya completado la operación solicitada, se enviará un correo electrónico al cliente informándoles del resultado.

Esto es una pequeña y sencilla presentación al funcionamiento del proyecto. En las subsecciones siguientes se detallará el diseño de las 2 partes del mismo.

6.3. Diseño de la librería *MailService*

El diseño de la librería `MailService` se realizó en fases tempranas del proyecto. Como se mencionó en la subsección 4.1.4 debido a un problema en la implementación de un test de la librería se decidió hacer un cambio de diseño y hacer la librería más modular. El primer diseño del proyecto .NET de la librería `MailService` se puede observar en la figura 6.4

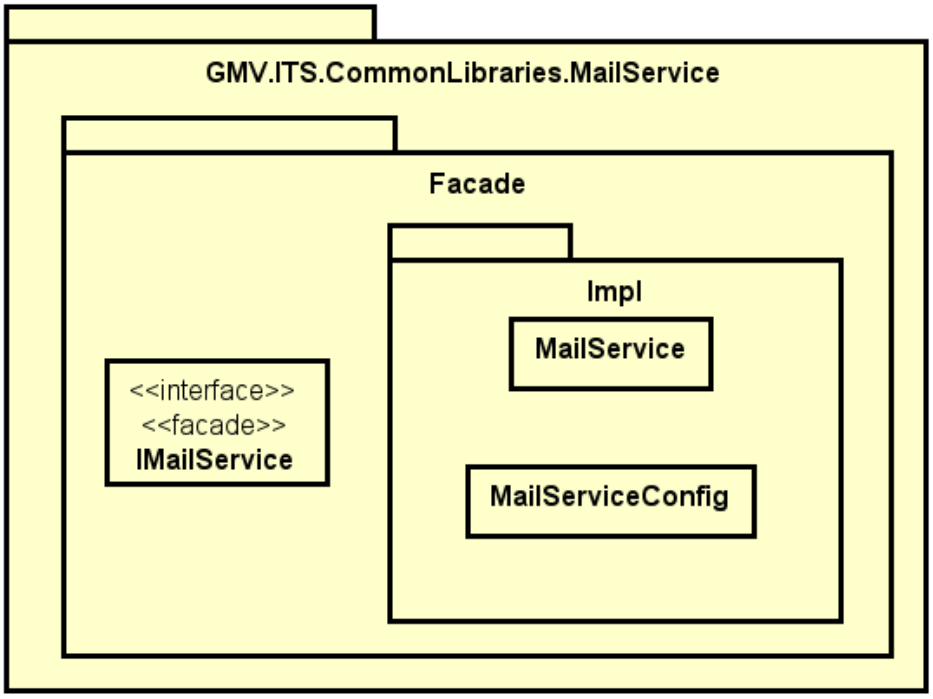

Figura 6.4: Diagrama de paquetes inicial de la librería

En la figura 6.5 se puede observar el diseño actual de la librería.
En ambos diseños se sigue la misma estructura de paquetes:

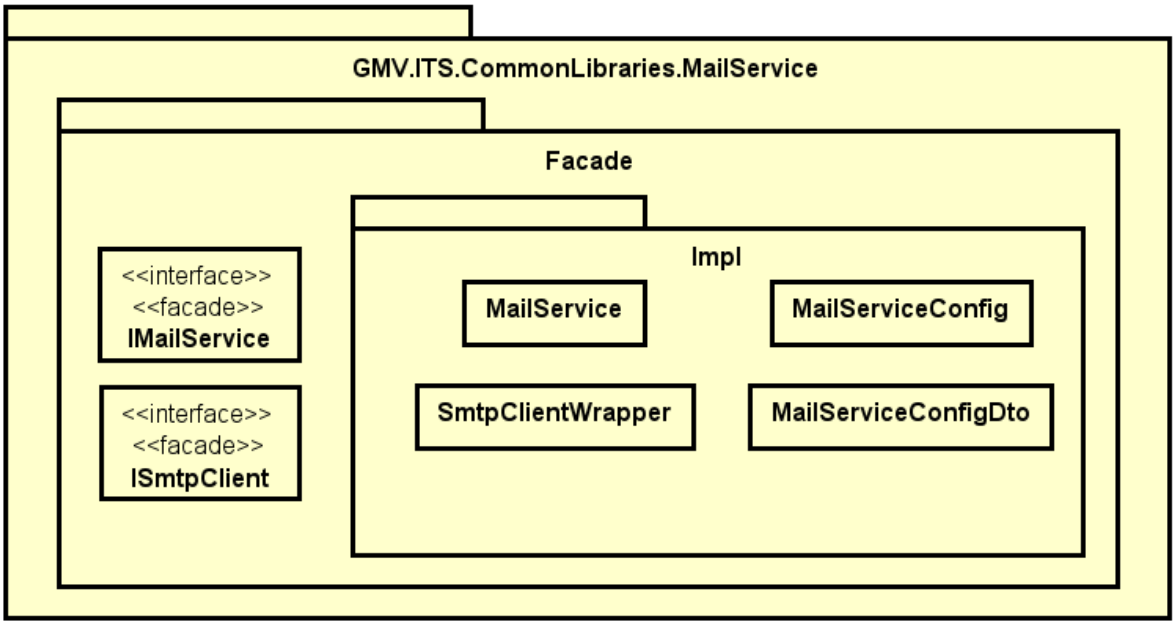

Figura 6.5: Diagrama de paquetes actual de la librería

- Paquete `Facade` (hace referencia al patrón de diseño *Facade* o *fachada*): que contiene las interfaces con las que se va a comunicar el cliente, servicio o cualquier clase que lo necesite.

- Paquete `Impl`: está contenido en el paquete `Facade` y contiene la clase principal `MailService` que es la que se encarga de enviar los correos electrónicos y sus clases auxiliares como `MailServiceConfigDto` que contienen los datos del correo electrónico.

Diagramas de clase y de secuencia

En esta subsección se explicarán los motivos del cambio de diseño de la librería así como cambios adicionales secundarios con ayuda de los diagramas de dominio de ambas versiones.

En primer lugar, una de las ventajas la versión inicial es que era más sencilla de entender y de usar, ya que solo era necesario crear una instancia de `MailServiceConfig`, rellenar sus datos, pasárselos a `MailService` e invocar el método `Send()`. La principal desventaja de esta aproximación era la imposibilidad de realizar los tests con *Mocks*, ya que lo único que se tendría que simular es el envío del correo del `SmtpClient` estaba

interno de manera privada en `MailService`. En las figuras 6.6 y 6.7 se pueden observar los diagramas de dominio simple y detallados respectivamente de la versión inicial.

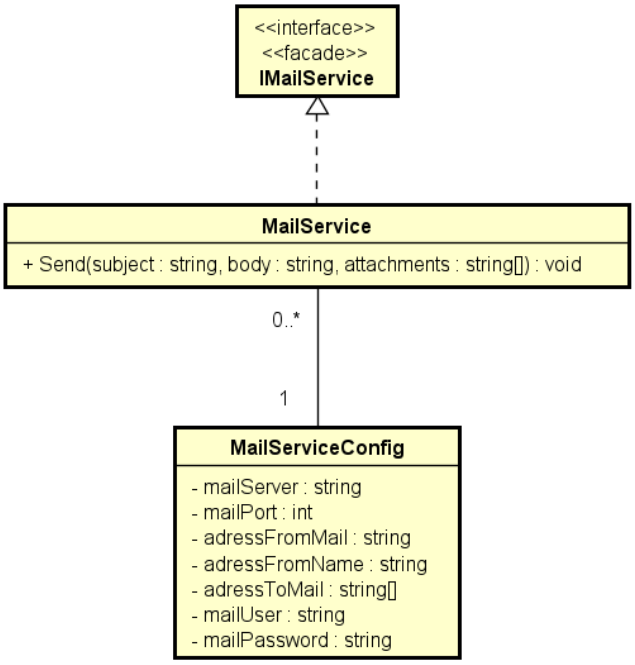

Figura 6.6: Diagrama de dominio simple de la versión inicial de la librería

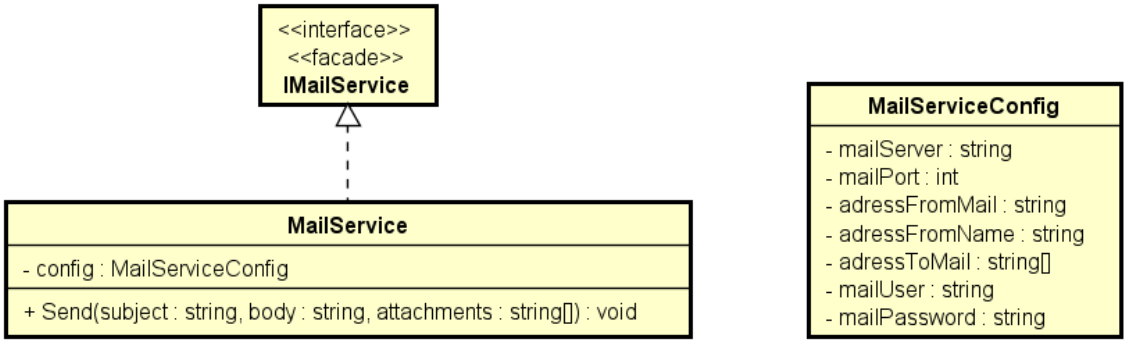

Figura 6.7: Diagrama de dominio detallado de la versión inicial de la librería

Con la nueva aproximación de la librería se consigue resolver el problema de la implementación del test añadiendo la interfaz `ISmtpClient` que es la que será simulada en test por los *mocks* y que actúa como fachada. `SmtpClientWrapper` que se encarga de crear el cliente SMTP. También se decidió separar de `MailServiceConfig` los atributos de la comprobación de la integridad de los mismos. Los atributos se encuentran en `MailServiceConfigDto` y la comprobación de la integridad de los mismos se encuentra

en `MailServiceConfig`. Por último, también se decidió quitar el usuario y la contraseña del remitente de `MailServiceConfig` por seguridad y que no quedase almacenado en el objeto `MailServiceConfig` en texto plano y los tenga que introducir el usuario directamente en el cliente SMTP como se indica en el diagrama de secuencia de la figura 6.10. En las figuras 6.8 y 6.9 se pueden observar los diagramas de dominio simple y detallados respectivamente de la versión actual.

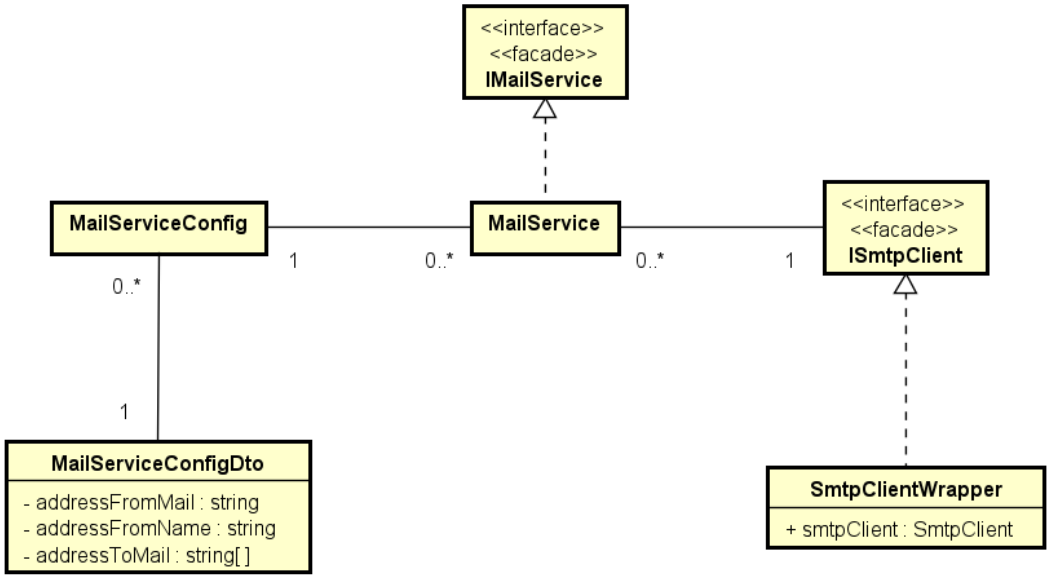

Figura 6.8: Diagrama de dominio simple de la versión actual de la librería

Por último, en la figura 6.10 se puede observar el diagrama de secuencia que realiza el servicio o la clase que necesite usar la librería para que funcione correctamente.

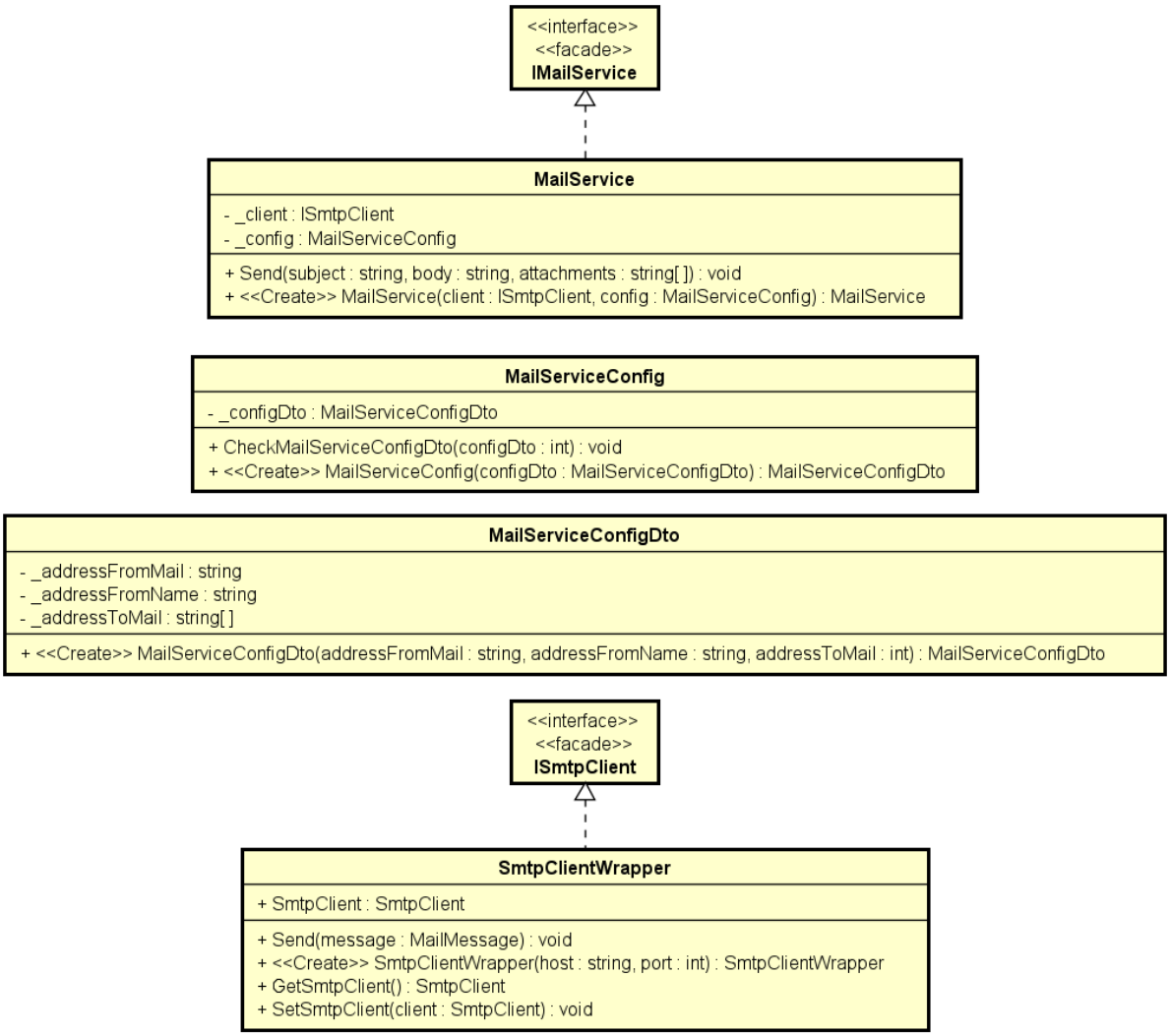

Figura 6.9: Diagrama de dominio detallado de la versión actual de la librería

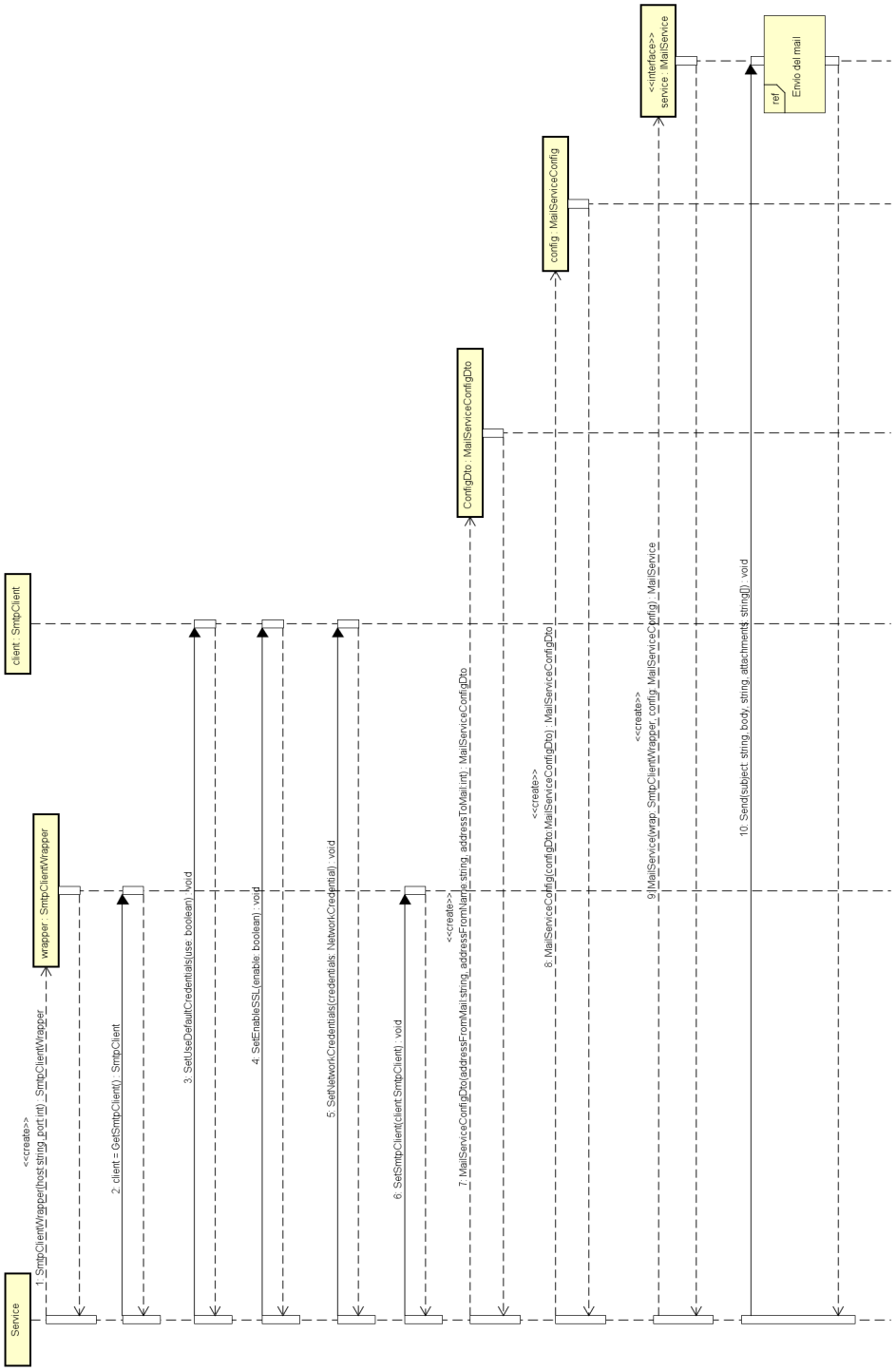

Figura 6.10: Diagrama de secuencia de la versión actual de la librería

6.4. Diseño del microservicio *Planner Integrator*

6.4.1. Arquitectura del microservicio

En GMV se sigue un estándar para la creación e implementación de sus microservicios por medio de una plantilla. Se dispone de 2 tipos de plantillas: la simple y la compleja. La plantilla simple crea un microservicio sin frontend y la compleja con frontend. En el caso de este proyecto se usó la plantilla simple cuya arquitectura general se puede observar en la figura 6.11.

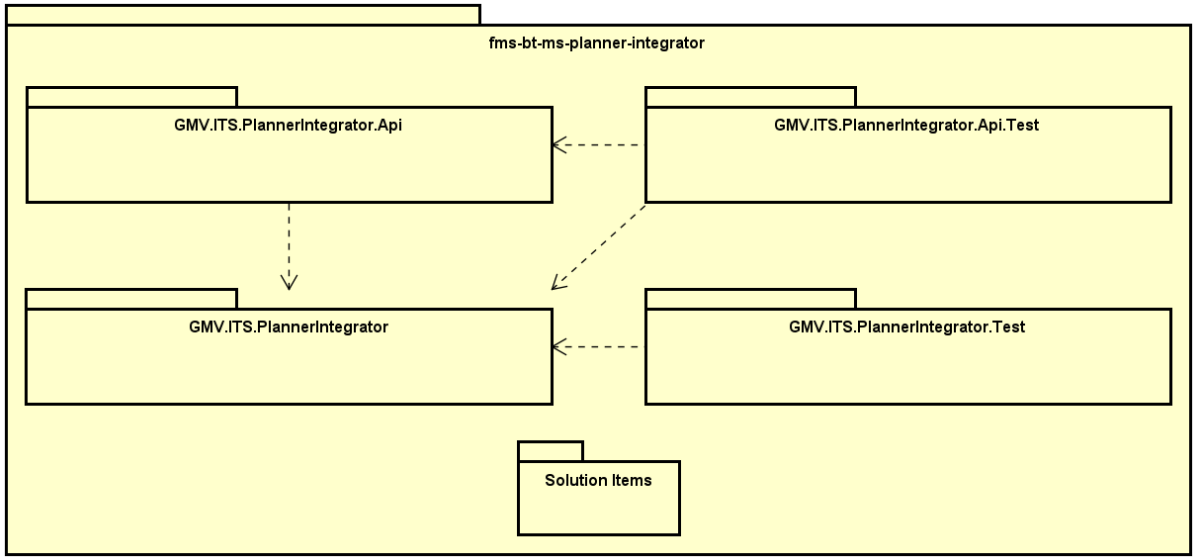

Figura 6.11: Arquitectura general del microservicio

Dentro del paquete general `fms-bt-ms-plannerintegrator` se encuentran 4 paquetes distintos que corresponden a 4 proyectos .NET: 2 proyectos de funcionalidad (las 2 principales capas del microservicio) y 2 proyectos de test (se corresponden con los que su nombre terminan con `.Test`).

Primeramente, se centrará la atención en los proyectos de funcionalidad:

- El paquete `GMV.ITS.PlannerIntegrator` (se denomina la capa **microservicio**): es el que contiene las clases de funcionalidad que se denominan **Servicios** así como su los modelos de datos que utiliza. En la figura 6.12 se puede observar su estructura interna. A continuación se explicará la función de los paquetes más relevantes para el desarrollo del microservicio:
  - El paquete `Attribute`: gestiona la autorización y los permisos de interacción con las bases de datos de los microservicios con los que se comunica el Planner Integrator.
  - El paquete `Definitions`: contiene constantes como el nombre del microservicio y el nombre de los nombre de los permisos que se utilizan el paquete `Attribute`.

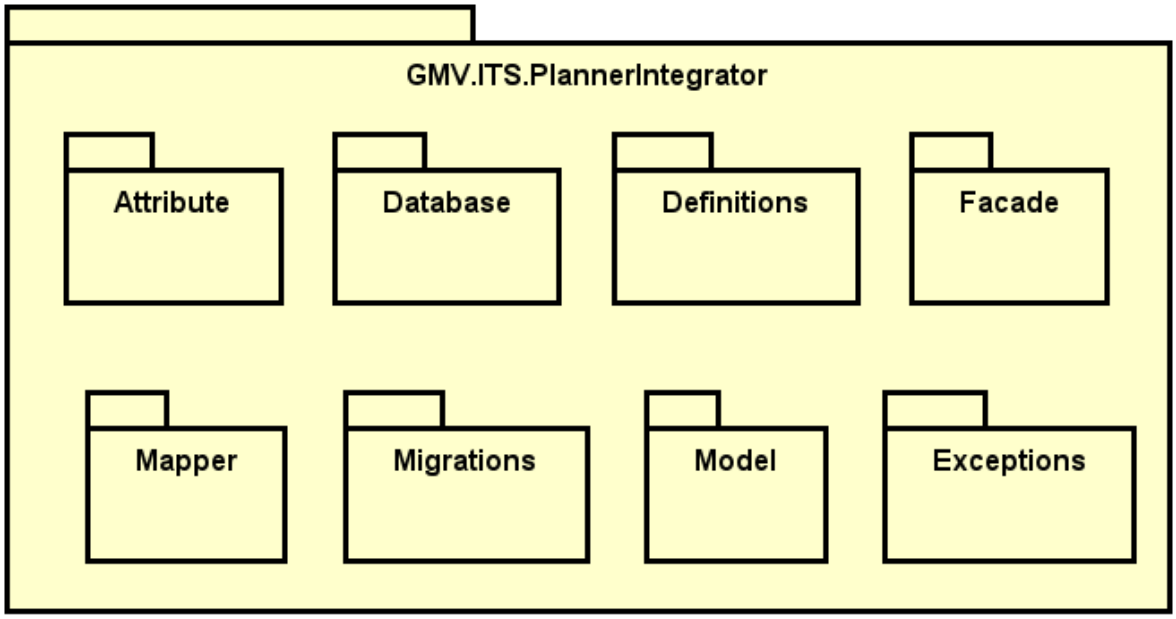

Figura 6.12: Arquitectura del paquete `GMV.ITS.PlannerIntegrator`

  - Los paquetes `Database` y `Migrations`: contienen todo lo relacionado con la creación, gestión y migración de la base de datos del microservicio. En este proyecto como el microservicio es una pasarela no requiere de una base de datos (por el momento).
  - El paquete `Facade`: en este paquete se encuentran los servicios que le dan funcionalidad a los endpoints del microservicio.
  - El paquete `Model`: contiene todas las clases del modelo que usarán los servicios del paquete `Facade` y los tests.
  - El paquete `Exceptions`: contiene excepciones personalizadas en caso de que se requiera.

- El paquete `GMV.ITS.PlannerIntegrator.Api` (se denomina la capa **API**): es el encargado de levantar el microservicio y su API para que los clientes se puedan comunicar con él. En la figura 6.13 se puede observar su estructura interna. A continuación se explicará la función de los paquetes más relevantes para el desarrollo del microservicio:

  - El paquete `Controllers`: contiene los controladores públicos y privados encargados de levantar los endpoints y llamar a los servicios.
  - El paquete `HostedServices`: se encarga de iniciar el microservicio añadiendo los servicios que utilizarán los endpoints.

Este microservicio dispone de una arquitectura de capas relajada o arquitectura de capas abierta, ya que los componentes solamente interactúan con los de su capa y los

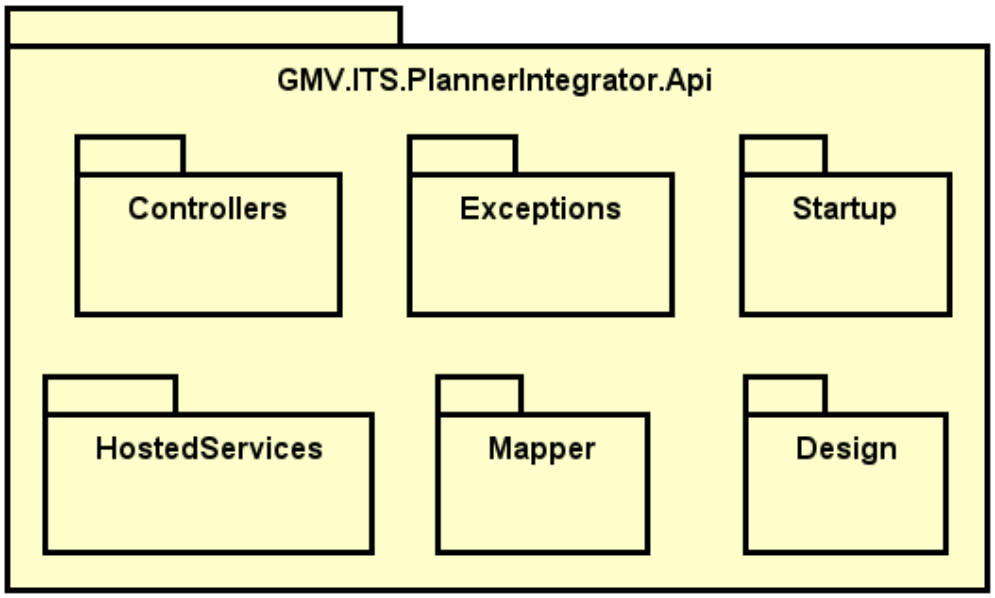

Figura 6.13: Arquitectura del paquete `GMV.ITS.PlannerIntegrator.Api`

de capas inferiores (considerando la capa API como la más inferior), proporcionando de esa manera cohesión entre los componentes de una misma capa y disminuyendo el acoplamiento entre capas lo máximo posible. [17]. En la figura 6.14 se pueden observar de una forma simplificada las dependencias que tiene la capa de API con la capa de microservicio. Y en la figura 6.15 se pueden observar las dependencias entre las principales

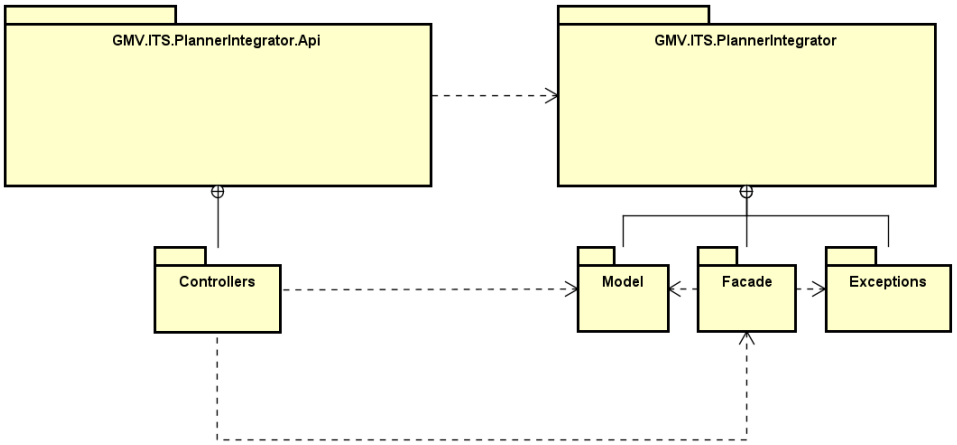

Figura 6.14: Dependencias entre las capas del microservicio

clases de las distintas capas.

La capa API es la que se encarga de recibir las peticiones del cliente por medio de sus controladores del paquete `Public` que son los que definen la *URI* común de los *endpoints*

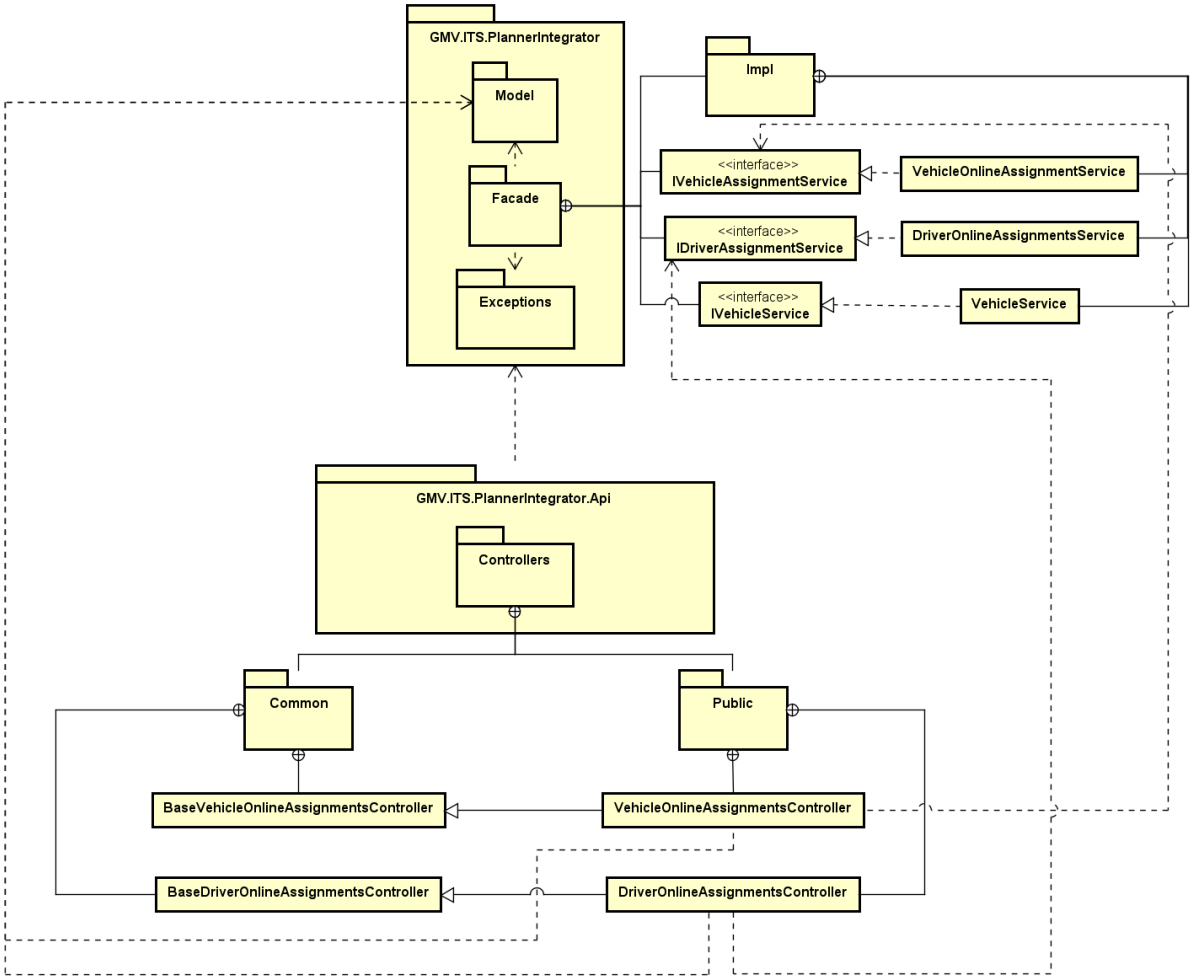

Figura 6.15: Diagrama Dependencias entre las clases de las distintas capas

(por ejemplo, en el caso de los cambios online de vehiculos, la `URI` común será la que compartan todos los endpoints de los cambios online de vehículos). Dependiendo que petición realice el cliente (asignar, desasignar o reemplazar) se llamará a un endpoint o a otro (con los datos que correspondan, los cuales se muestran en la figura 6.16) en los controladores del paquete `Common` que son los encargados de llamar a la capa Microservicio para realizar las operaciones requeridas por el cliente.

6.4.2. Modelo de datos

En esta sección se entrará más en detalle sobre el paquete `Model` y sus clases. Principalmente, este paquete contiene los datos con los que operan los planificadores externos y que envían a través de peticiones HTTP a los distintos *endpoints* del microservicio.

A estos datos se les decidió llamarles `Requests`, ya que son los datos que recibe el microservicio de las peticiones que envía el planificador.

En primer lugar, las `Requests` se dividen en 2 grupos: las referentes a los cambios online de vehículos y las referentes a los cambios online de conductores, como se puede observar en la figura 6.16. La estructura es igual en ambos casos: una clase base que contiene los datos comunes a todas las clases hijas y las clases hijas que contienen los datos específicos para cada operación (asignar, desasignar o reemplazar). A continuación se describen los datos de las `Requests`:

- **Cambios online de vehículos:**
  - `BaseVehicleRequest`
    - `startTime`: fecha de inicio del cambio
    - `endTime`: fecha de fin del cambio
    - `blockCode`: código del servicio de vehículo (o bloque) al que se va a asignar el vehículo especificado
    - `blockOrganizationCode`: código de la organización a la que pertenece el bloque especificado
  - `VehicleAssignmentRequest`
    - `vehicleCode`: código del vehículo que se quiere asignar (también se llama sideCode)
    - `vehicleOrganizationCode`: código de la organización a la que pertenece el vehículo que se quiere asignar [1]
  - `VehicleUnassignmentRequest`
    - `vehicleCode`: código del vehículo que se quiere desasignar
    - `vehicleOrganizationCode`: código de la organización a la que pertenece el vehículo que se quiere desasignar
  - `VehicleReplacementRequest`
    - `oldVehicleCode`: código del vehículo que se quiere reemplazar
    - `oldVehicleOrganizationCode`: código de la organización a la que pertenece el vehículo que se quiere reemplazar
    - `newVehicleCode`: código del nuevo vehículo que se quiere asignar al servicio
    - `newVehicleOrganizationCode`: código de la organización a la que pertenece el nuevo vehículo que se quiere asignar
- **Cambios online de conductores:**
  - `BaseDriverRequest`
    - `startTime`: fecha de inicio del cambio

[1] Normalmente la organización del vehículo suele ser la misma que la organización del servicio pero no siempre, ya que existe un concepto llamado **flotas amigas** el cual consiste en la compartición de vehículos entre distintas organizaciones.

  - `endTime`: fecha de fin del cambio
  - `dutyCode`: código del servicio de conductor (o *duty*) a la que se va a asignar el conductor especificado
  - `dutyOrganizationCode`: código de la organización a la que pertenece la *duty* especificada
- `DriverAssignmentRequest`
  - `driverCode`: código del conductor que se quiere asignar
  - `driverOrganizationCode`: código de la organización a la que pertenece el conductor que se quiere asignar
- `DriverUnassignmentRequest`
  - `driverCode`: código del conductor que se quiere desasignar
  - `driverOrganizationCode`: código de la organización a la que pertenece el conductor que se quiere desasignar
- `DriverReplacementRequest`
  - `oldDriverCode`: código del conductor que se quiere reemplazar
  - `oldDriverOrganizationCode`: código de la organización a la que pertenece el conductor que se quiere reemplazar
  - `newDriverCode`: código del nuevo conductor que se quiere asignar al servicio
  - `newDriverOrganizationCode`: código de la organización a la que pertenece el nuevo conductor que se quiere asignar

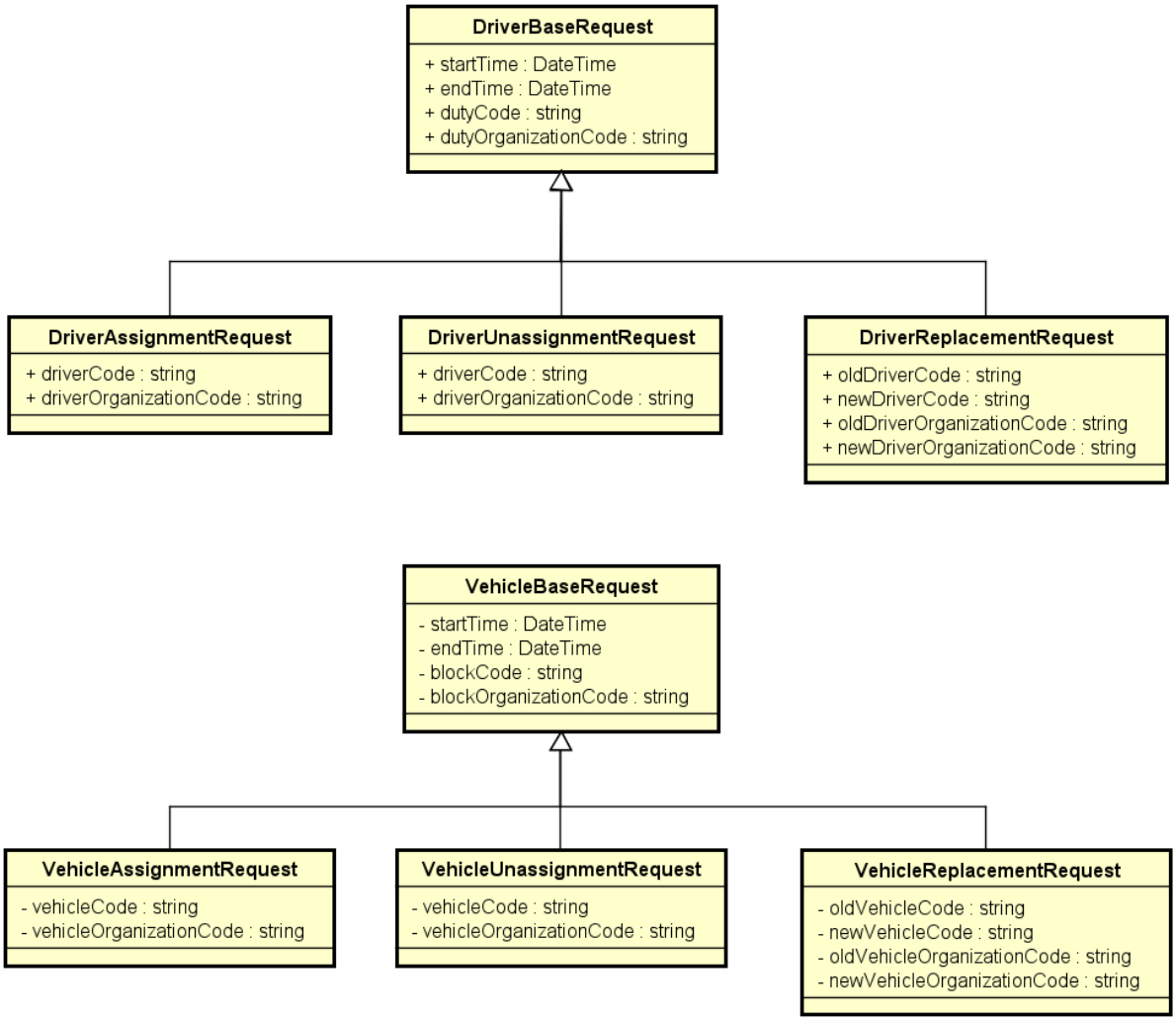

Figura 6.16: Clases del paquete `Model` que utilizan los controladores y servicios

6.4.3. Diseño de los cambios online

En esta sección se detallará la secuencia principal de las principales operaciones que realiza el microservicio a petición del cliente. La asignación, desasignación y reemplazo son operaciones con un flujo de ejecución muy similar por lo que solamente se pondrán ejemplos para la asignación de vehículos y seguidamente para la asignación de conductores.

Diseño de los cambios online de vehículos

En la figura 6.17 se describe el flujo principal de la asignación online de vehículos. La operación comienza cuando el actor lanza una petición al microservicio la cual recibe el controlador encargado de los cambios online de los vehículos, el cual invoca al servicio correspondiente (en este caso el servicio `VehicleAssignmentService`). Llegados a este

punto, la operación no es tan sencilla como recoger los datos recibidos en la petición y enviárselos al microservicio DPP del sistema Suite, ya que los datos que recibimos del cliente son datos "públicos"que maneja el cliente y el microservicio DPP opera con datos "privados", ya que es un microservicio *core* para el sistema Suite. Es por eso que hay que utilizar los datos que recibimos para poder obtener los datos necesarios para poder enviárselos al microservicio DPP.

En primer lugar, es necesario obtener el código privado `appId` del vehículo a partir de su código público `vehicleCode` y del código de su organización. Es necesario especificar también el código de la organización, ya que el código del vehículo solo es único dentro de su organización, es decir, puede haber 2 vehículos con el mismo código que pertenecen a organizaciones diferentes. La obtención del código privado del vehículo hace referencia al diagrama de secuencia de la figura 6.18. Primero se hace una llamada al microservicio OC para obtener todos los vehículos con el mismo código y tras eso se comprueban todos los vehículos devueltos y se filtra el que pertenece a la organización deseada. Tras encontrar el vehículo requerido se extrae su código `appId`.

Lo mismo ocurre con el código del bloque que proporciona el cliente, es necesario encontrar su código privado (también denominado `appId`). La obtención del código privado del bloque hace referencia al diagrama de secuencia de la figura 6.19. Para obtener el código privado del bloque es necesario especificar su código público, el código de su organizacion y la fecha en el que está localizado el bloque. En este caso se llama al microservicio DPP que es el microservicio encargado de gestionar los servicios de vehículo (bloques) y de conductor (*duties*) para obtener todos los bloques para la fecha y organización especificadas. Tras obtener todos los bloques se busca el que corresponda con el código proporcionado por el cliente y se extrae su `appId`.

Una vez que se tengan todos los datos disponibles para realizar la operación, se crea el objeto que almacena todos los datos, en este caso se trata de la clase `DatedVehicleWorkAssignmentRequest` y se llama a la operación de asignación del vehículo del microservicio DPP.

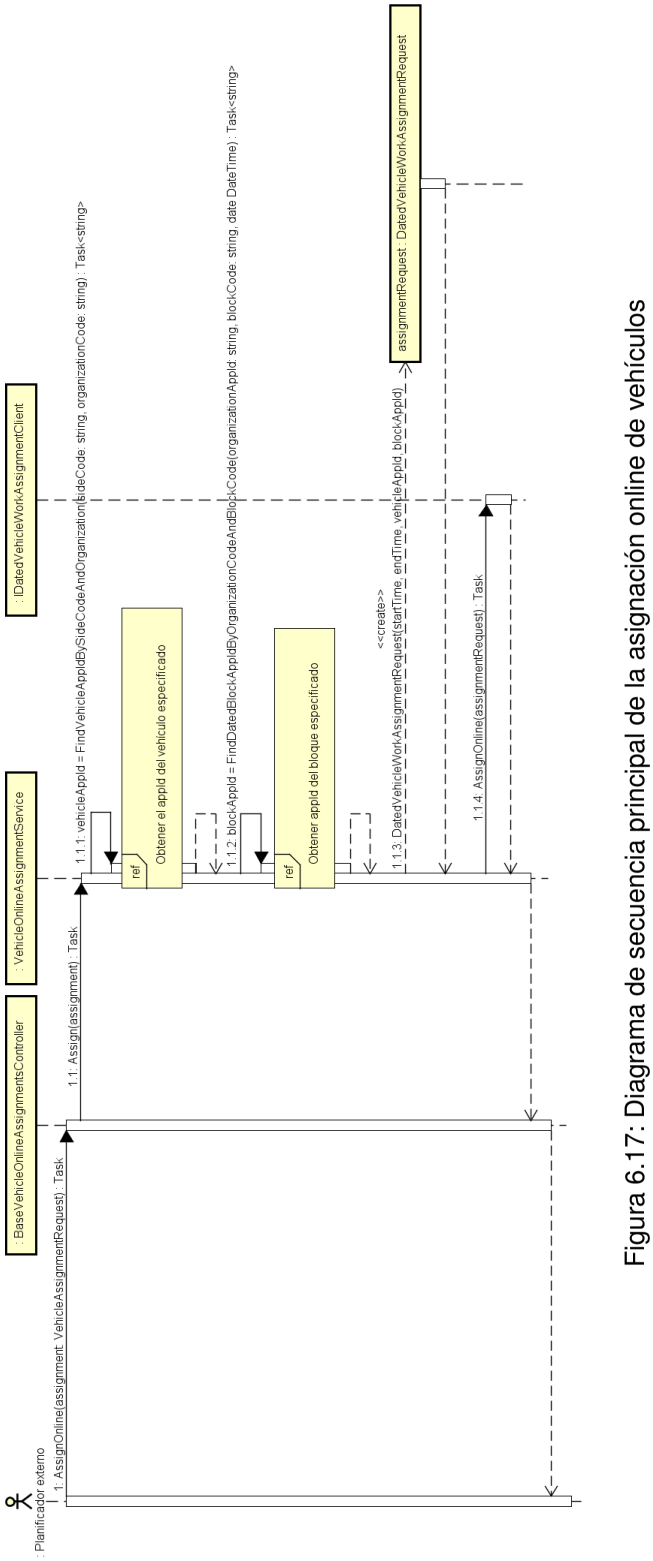

Figura 6.17: Diagrama de secuencia principal de la asignación online de vehículos

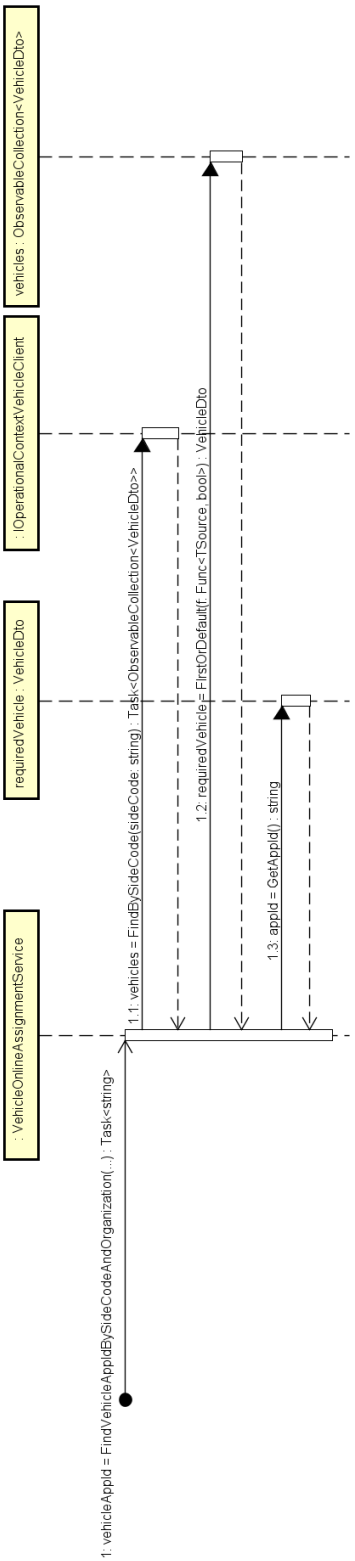

Figura 6.18: Diagrama de secuencia intermedio para la obtención del `appId` del vehículo

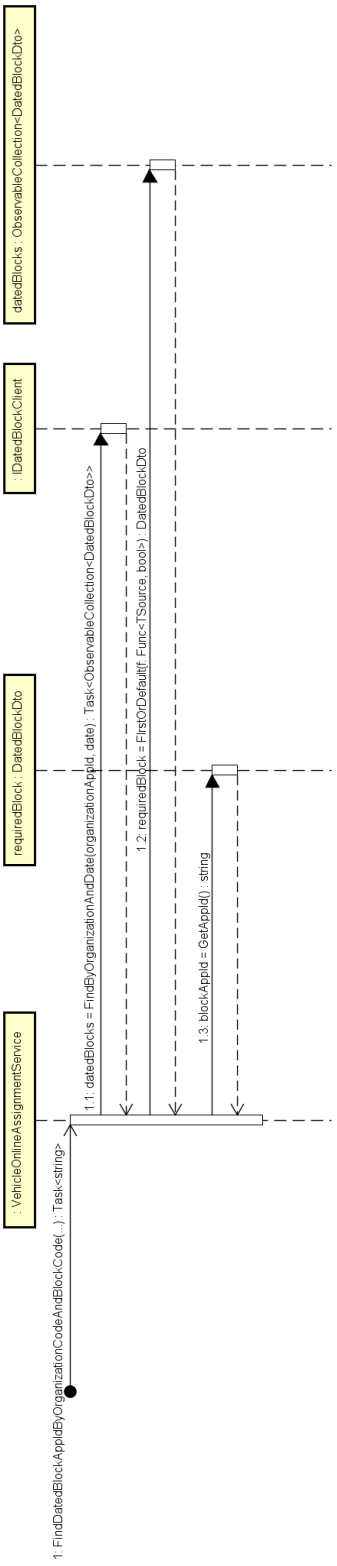

Figura 6.19: Diagrama de secuencia intermedio para la obtención del `appId` del bloque

Diseño de los cambios online de conductores

En la figura 6.20 se describe el flujo principal de la asignación online de conductores. La secuencia de esta operación es muy similar a la descrita en la subsección anterior 6.4.3 por lo que en esta subsección solo se expondrán las diferencias notables que existen entre estas 2 operaciones.

La principal diferencia reside en que como en OC no existe una operación la cual permita obtener a todos los conductores por su código, se decidió obtener todos los conductores disponibles y tras ello, filtrarles por su código y el código de su organización. Una vez se ha obtenido el conductor requerido, se extrae su `appId` (véase en la figura 6.21).

Las *duties* a los conductores son lo que los bloques a los vehículos, por lo que la obtención del `appId` de la *duty* (véase en la figura 6.22) es muy similar a la obtención del bloque en el caso de los vehículos como se observa en la figura 6.19.

6.4.4. Diseño de la API de cambios online

La API que ofrece el microservicio se divide en 2: los endpoints para realizar los cambios online de vehículos y los endpoints para realizar los cambos online de los conductores. Se pueden observar en el diagrama de la figura 6.23.

6.4.5. Diseño de la carga planificada de vehículos

La carga planificada de vehículos en el sistema Suite se compone de 2 partes: una parte específica para cada cliente y una parte común. La función de la parte específica es obtener de forma diaria los vehículos de la API del cliente y enviárselos a la parte común (esta parte privada queda fuera del alcance del proyecto por falta de tiempo). La parte común es la que se encarga de recibir los vehículos del cliente y cargarlos. En la figura 6.24 se puede observar el flujo principal de esta operación.

Primeramente, se reciben los vehículos de la parte específica del cliente y se obtienen los vehículos que existen en el sistema Suite de ese cliente. La carga de vehículos consiste en crear los vehículos que se reciben del cliente que no están cargados en el sistema Suite, actualizar los vehículos que existen en la lista que se reciben del cliente y en el sistema Suite y eliminar los vehículos que existen en el sistema Suite, pero no en la lista de vehículos que se recibe del cliente. Como se observa en la figura 6.24, primero se obtienen los vehículos que se tienen que crear, seguidamente los que se tienen que actualizar y por último los que han de borrarse del sistema.

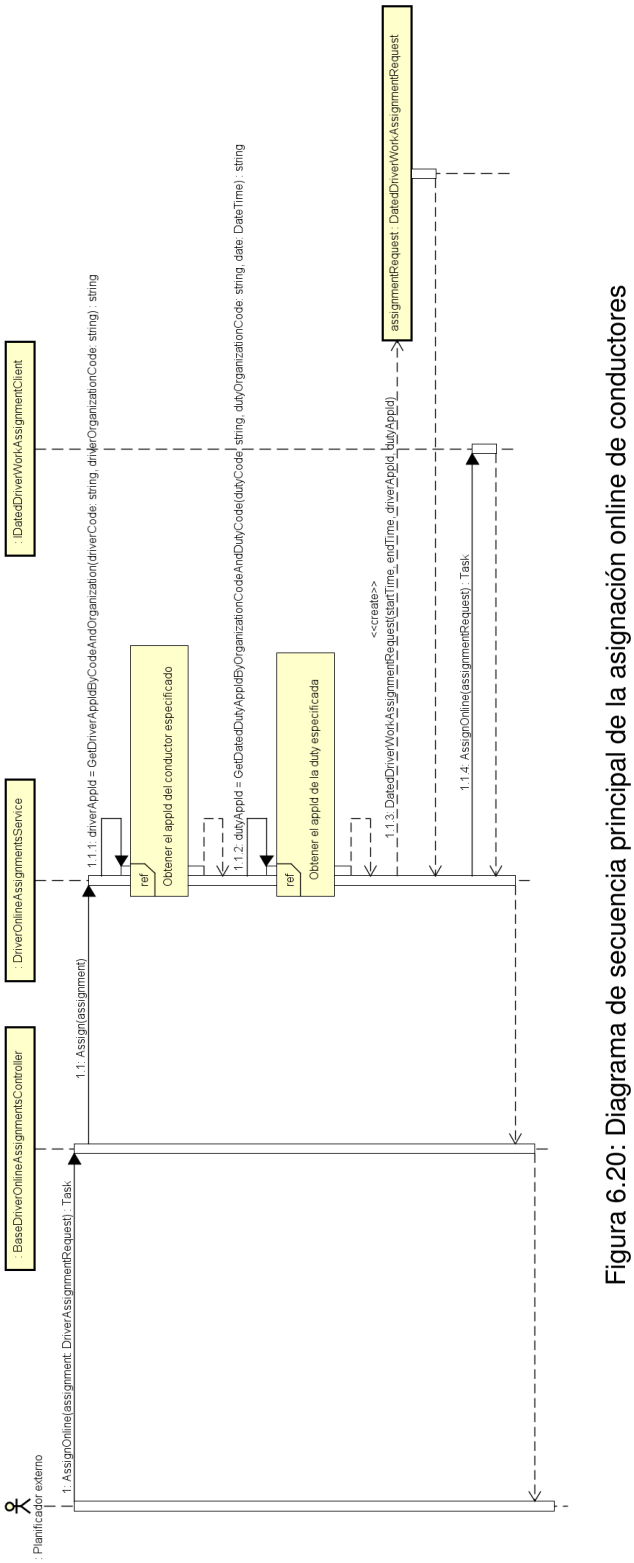

Figura 6.20: Diagrama de secuencia principal de la asignación online de conductores

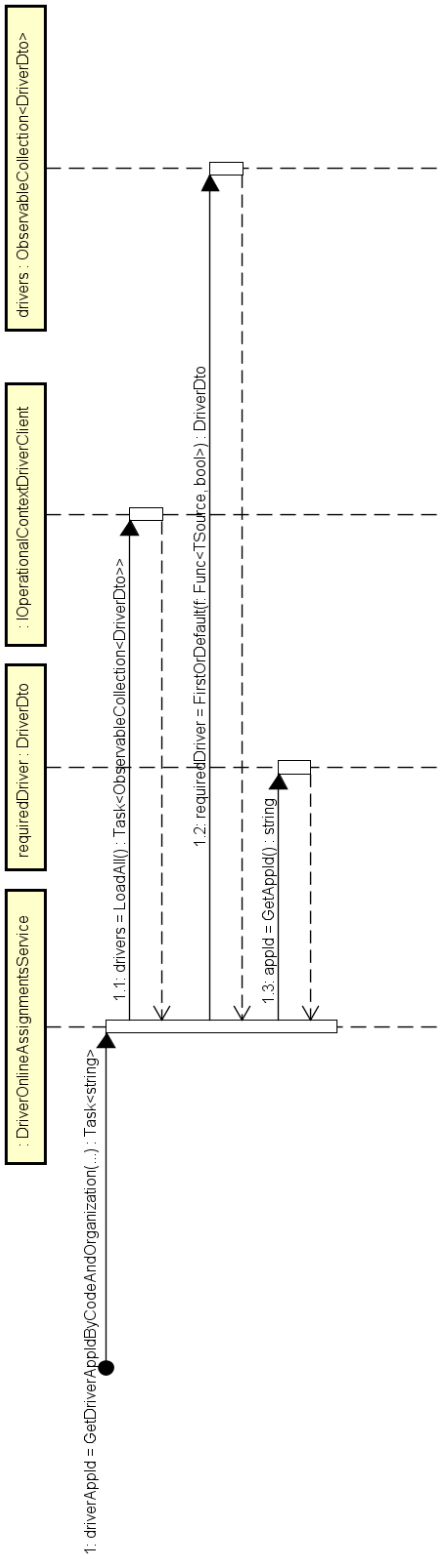

Figura 6.21: Diagrama de secuencia intermedio para la obtención del `appId` del conductor

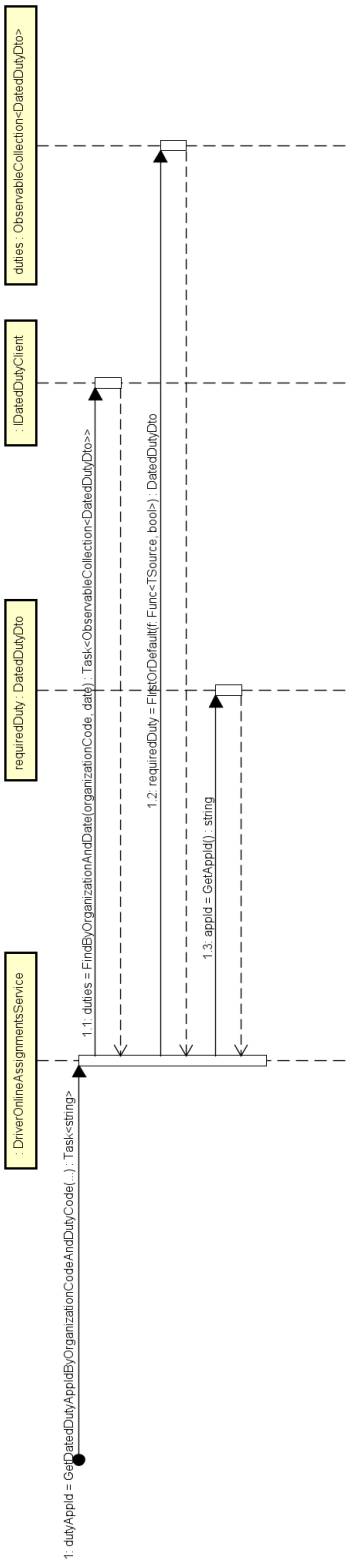

Figura 6.22: Diagrama de secuencia intermedio para la obtención del `appId` de la *duty*

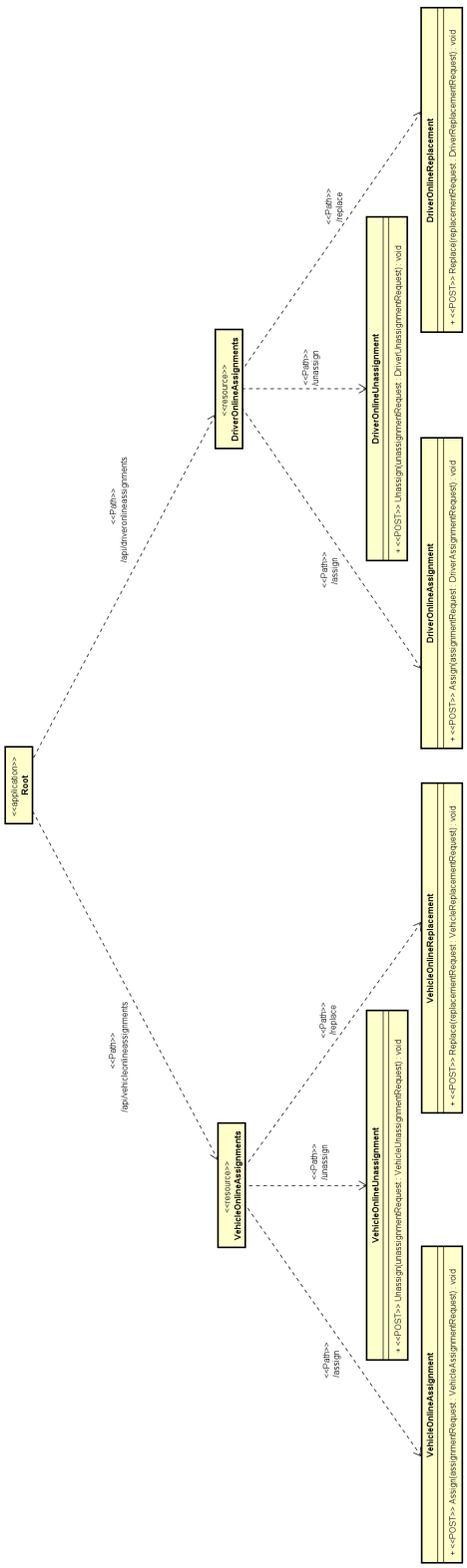

Figura 6.23: Diagrama de recursos de la API

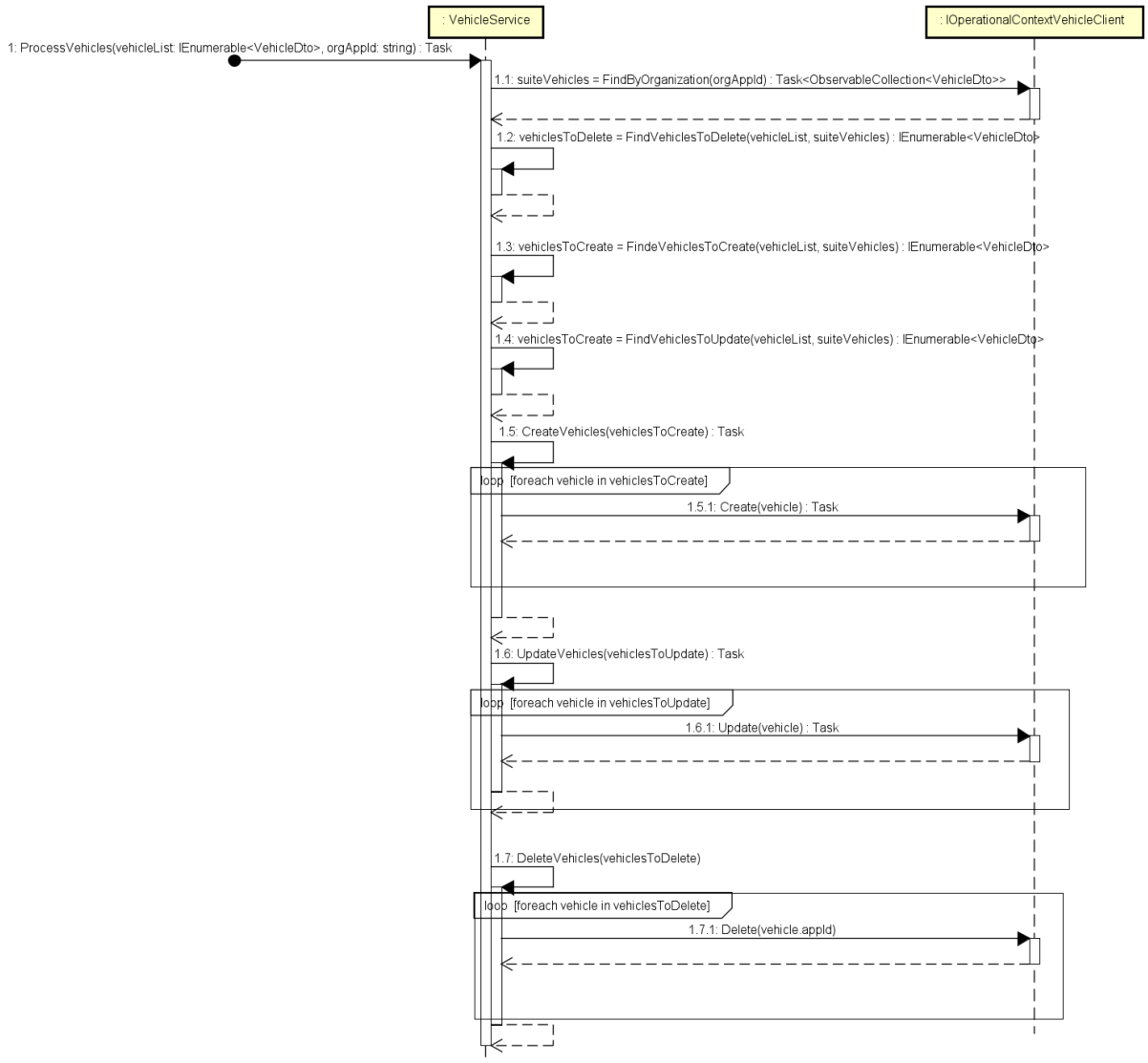

Figura 6.24: Diagrama de secuencia de la carga planificada de vehiculos

Capítulo 7

Implementación

En este capítulo se describe más detalladamente las tecnologías implicadas en la implementación del sistema. Por temas de privacidad, en este capítulo no se expondrá código concreto del software desarrollado en el proyecto.

7.1. Guía de estilo

Dado que la librería y el microservicio es muy probables que se utilicen en varios proyectos tanto nacionales como internacionales se decidió seguir una guía de estilo tanto de cara al desarrollador como al cliente:

- Para el **desarrollador**: en GMV ya se usaban unos estándares para el desarrollo de código por lo que se trasladaron también a este proyecto. Las reglas principales son las siguientes:
  - El nombre de los atributos privados debe ser precedido por '_' y el de las interfaces siempre tiene que estar precedido por una **I** mayúscula.
  - El nombre de los métodos tanto públicos como privados deben empezar por mayúscula.
  - El nombre de los atributos públicos de una clase deben empezar por mayúscula.
  - Todos los nombres de clase, método, variables, etc. deben estar en inglés.
  - Los métodos de los controladores solo pueden llamar al servicio que les corresponda, es decir, no pueden encapsular funcionalidad adicional. En el código 7.1 se puede observar un ejemplo de ello.

```
1        public async Task ExampleEndpoint(...)
2        {
3            // Solo se llama al servicio correspondiente
4            await _exampleService.Example(...);
5        }
```

Listing 7.1: Ejemplo de llamada a un servicio

- Para el **cliente**: para los mensajes de error y notificación por correo electrónico se decidió usar el inglés tanto para proyectos nacionales como internacionales.

7.2. Integración continua

Como se mencionó en la sección de 3.4 se usó Jenkins para la integración continua del proyecto. Para configurar los proyectos .NET para la integración con Jenkins son necesarios algunos ficheros de configuración los cuales se encuentran en el paquete `Solution Items` (véase en la figura 6.11). Entre estos ficheros de configuración se encuentra el `.gitignore` para evitar sobrecargar el repositorio y la herramienta Jenkins de ficheros innecesarios. Los demás se listan a continuación:

- El fichero `Directory.Build.Props`: en él se almacena la versión del sistema Suite y sus paquetes que va a usar el proyecto. Este fichero lo utiliza por ejemplo el fichero `before.PlannerIntegrator.sln.targets`.

- El fichero `before.PlannerIntegrator.sln.targets`: se encarga de importar los paquetes NuGet básicos para la integración continua del proyecto como SonarQube, Git, Api, etc. En el código 7.2 se puede observar el formato del fichero. Ciertas líneas del fichero se han ocupado por un '**(...)**'por cuestiones de privacidad.

```
1         <PropertyGroup Condition="$(NuGetPackageRoot) == ''">
2           <!-- Ruta base a los paquetes NuGet -->
3           <NuGetPackageRoot>(...)</NuGetPackageRoot>
4         </PropertyGroup>
5
6         <!-- Importacion de otros ficheros -->
7         <Import Project="Directory.Build.props" />
8
9         <ImportGroup Condition="(...)">
10          <Import Project="(...)\SonarQube.targets"/>
11          <Import Project="(...)\Versioning.targets"/>
12          <Import Project="(...)\DotnetTools.targets"/>
13          <Import Project="(...)\Git.targets"/>
14          <Import Project="(...)\ApiCompat.targets"/>
15        </ImportGroup>
```

Listing 7.2: Contenido del fichero `before.PlannerIntegrator.sln.targets`

- El fichero `Directory.Build.targets`: se encarga de importar los paquetes NuGet necesarios para el funcionamiento del proyecto como el framework para la implementación de los tests `xunit` y las dependencias de GMV como los paquetes del microservicio OC o DPP. En el código 7.3 se puede observar el formato del fichero.

```
1         <ItemGroup Label="GMV Dependencies">
2           (...)
3         </ItemGroup>
4
```

```
<ItemGroup Label="3rd party dependencies">
  <PackageReference Update="Microsoft.AspNetCore*" Version=
     "3.1.1" />
  (...)
</ItemGroup>

<ItemGroup Label="Test project dependencies">
  <PackageReference Update="Microsoft.NET.Test.Sdk" Version
     ="16.11.0" />
  <PackageReference Update="Moq" Version="4.13.0" />
  <PackageReference Update="xunit*" Version="2.4.1" />
</ItemGroup>
```

Listing 7.3: Contenido del fichero `Directory.Build.Props`

- El fichero `dotnet-tools.json`: se encarga de definir las tareas que realizará Jenkins cuando se suba el código al repositorio. En el código 7.4 se puede observar el formato del fichero.

```
{
  (...)
  "tools": {
    "dotnet-ef": {
      (...)
    },
    "dotnet-sonarscanner": {
      (...)
    },
    "dotnet-reportgenerator-globaltool": {
      (...)
    }
  }
}
```

Listing 7.4: Contenido del fichero `dotnet-tools.json`

- El fichero `global.json`: se encarga de definir la versión del SDK de .NET que se ha utilizado en le proyecto. En el código 7.5 se puede observar el formato del fichero.

```
{
  "sdk": { "version": "3.1.101" }
}
```

Listing 7.5: Contenido del fichero `global.json`

7.3. Implementación de la librería *MailService*

Para implementar la librería *MailService* se utilizó la dependencia de .NET predeterminada `System.Net.Mail`. Esta dependencia proporciona clases importantes como

`SmtpClient` que es la clase encargada de realizar el envío del correo o `MailMessage` que almacena toda la información del correo electrónico que se va a enviar.

El método principal de la librería `Send(...)` se encuentra en la clase principal `MailService`. En una primera aproximación se decidió realizar 2 métodos distintos como se observa en el código 7.6, uno para el envío de correos sin archivos adjuntos y otro para el envío de correos con archivos adjuntos. Pero al final se decidió acoplar todo en un método debido a la duplicación de código que se producía en ambos métodos.

```
1    public void Send(string subject, string body)
2    {
3        (...)
4    }
5
6    public void SendWithAttachments(string subject, string body, string
         [] attachments)
7    {
8        (...)
9    }
```

Listing 7.6: Aproximación inicial de la librería *MailService*

Para realizar el envío del correo correctamente hay que configurar todo lo necesario del correo y de ello se encargan las clases `MailServiceConfig`, `MailServiceConfigDto` y `SmtpClientWrapper`. Como se explicó en la sección 6.3 del capítulo 6, la clase `MailServiceConfigDto` almacena la dirección del remitente, su nombre y la/s dirección/es del/los destinatario/s y la clase `MailServiceConfig` se encarga de comprobar la integridad de los datos mencionados anteriormente. Por último, la clase `SmtpClientWrapper` se encarga de configurar el cliente `SMTP` con otros datos como el puerto o el servidor de correo.

Una vez completada la implementación se subió el código desarrollado a Jenkins para poder sacar la primera versión de la librería. Pero surgió algún inconveniente, ya que algunos de los ficheros del paquete `Solution Items` no estaban bien configurados por lo que fue necesario configurarlos de forma correcta. Cuando se consiguió solucionar el problema se corrigieron los *code smells* reportados por SonarQube y se puso a disposición de la empresa la primera versión de la librería.

7.4. Implementación del microservicio *Planner Integrator*

Antes de la implementación de la funcionalidad del microservicio, fue necesario poner a punto la plantilla mencionada en la sección 6.4.1. Esta puesta a punto de la plantilla supuso:

- **Configuración de las dependencias de GMV**: lo normal es que los microservicios se comuniquen entre ellos por medio de peticiones HTTP usando las APIs que despliega cada uno. En GMV se realiza de una forma diferente, en el código no

se crean ni configuran las peticiones `HTTP` de forma explícita en el código, sino que se usan clientes privados de cada microservicio en forma de paquetes NuGet que se encuentran en un repositorio propio de GMV de paquetes NuGet. Este paso previo a la implementación se centra en la configuración de ciertos archivos proporcionados en la plantilla para que el microservicio en tiempo de compilación pueda acceder al repositorio central NuGet de GMV y bajar todos los paquetes correspondientes.

- **Creación y configuración de los controladores del microservicio**: el siguiente paso es la creación de los controladores que pondrán los endpoints a disposición del cliente. Como se puede observar en la figura 6.15 del capítulo 6, en el paquete `Controllers` existen otros 2 paquetes más: `Public` y `Common`. El primero de ellos contiene los controladores que definen la URI común para varios endpoints (véase en el código 7.7). Estos controladores heredan de los definidos en el paquete `Common` que son los que definen los endpoints concretos y los que llaman a los servicios correspondientes (véase en el código 7.8). Una vez configurados los controladores se ejecutó el microservicio para probar que los endpoints estaban bien definidos y que recibía las peticiones de forma correcta.

```
1     [Produces(MediaTypeNames.Application.Json)]
2     [Route(ApiRoute)] // Define la ruta del controlador
3     public class ExampleController : BaseExampleController
4     {
5         internal const string ApiRoute = "api/exampleuri";
6         public ExampleController(IExampleService exampleService)
7             : base(exampleService){}
8     }
```

Listing 7.7: Configuración de un controlador del paquete `Public` de ejemplo

```
1     public class ExampleBaseController : ControllerBase
2     {
3         // Definicion del endpoint
4         internal const string ExampleEndpoint = "example";
5         // Servicio que usara el controlador
6         protected readonly IExampleService _exampleService;
7
8         public BaseExampleController(IExampleService exampleService
              )
9         {
10            _exampleService = exampleService;
11        }
12
13        [HttpPost(ExampleEndpoint)] // Tipo de operacion CRUD
14        public async Task ExampleEndpoint(...)
15        {
16            await _exampleService.ExampleMethod(...);
17        }
18    }
```

Listing 7.8: Configuración de un controlador base del paquete `Common` de ejemplo

- **Creación y configuración de los servicios**: el siguiente paso consiste en crear los ficheros que contienen la funcionalidad de los endpoints y sus interfaces y añadirlos a ciertos ficheros de configuración de dependencias para que puedan ser utilizados por otros softwares de GMV que lo requieran. Por último, se añadió la llamada de los controladores a los servicios y se probó que se ejecutaban correctamente cuando se llamaba a algún endpoint.

- **Creación de las `Requests` que usarán los endpoints**: el último paso previo a la implementación de la funcionalidad del microservicio consistió en definir los datos que usaría cada endpoint también llamados `Requests`. Estas clases del modelo de datos están definidas en el diagrama de la figura 6.16. Primero se crearon las clases padre cuyo nombre contiene la palabra `Base` y tras ellas las clases que heredan de las clases padre cuyo nombre contienen las palabras `Assignment`, `Unassignment` o `Replacement` según sea el caso.

Una vez se realizaron estos 3 pasos se procedió con la implementación de la funcionalidad de cada endpoint.

7.4.1. Implementación de los cambios online

Esta parte de la implementación del microservicio comenzó con la implementación de los métodos de asignación tanto de vehículos como de conductores, ya que es el método más sencillo de implementar para poder detectar errores y no cometerlos en la implementación de los siguientes métodos.

Una vez finalizados estos 2 primeros métodos se realizó una prueba en local teniendo levantado el sistema Suite se detectaron algunos problemas con los IDs y los códigos que se pasaban al microservicio, ya que el servicio procesaba los códigos que no tenía que procesar por lo que se arregló ese bug y se hicieron las pruebas en local necesarias.

Una vez hechas y pasadas las pruebas de estos 2 métodos iniciales se procedió a implementar la desasignación y reemplazo de conductores y vehículos los cuales también se hicieron pruebas en local superadas con éxito.

Tras implementar toda la funcionalidad de los servicios se creó una PR en el repositorio del proyecto y se corrigieron tanto los *code smells* reportados por SonarQube como las tareas de mejora de la PR creadas por los revisores de esta.

7.4.2. Implementación de la carga planificada de vehículos

Como se mencionó en la sección 6.4.5 la carga planificada de vehículos consiste en 2 partes: una parte común y una parte específica para cada cliente que se quiera integrar con este microservicio. Esta última parte no entrará dentro de este TFG, ya que no se disponía de tiempo para realizarlo. En la implementación no surgió ningún problema por lo que todo salió como se planificó. El único inconveniente surgió al realizar las pruebas en local, ya que este caso de uso se compone de las 2 partes mencionadas anteriormente de las cuales solo está implementada una de ellas así que para probar que todo funcionaba

correctamente se decidió crear un endpoint de prueba para simular que la parte específica del cliente enviase a la parte común todos los vehículos que hay que cargar.

Una vez probado todo, como se describió en la anterior sección, se creó una PR en el repositorio del proyecto, se corrigieron los *code smells* reportados por SonarQube y las tareas de mejora de la PR creadas por los revisores de esta.

Capítulo 8

Pruebas

8.1. Guías de estilo

En la implementación de las pruebas también se siguen ciertas guías y estándares de buenas prácticas. Estos son alguno de ellos:

- Los nombres de los métodos que prueban el flujo de error de un método debe seguir la siguiente regla: `NombreDelMetodoQueSePrueba_CasoDeError_LoQueRetorna()`. Por ejemplo, si se quiere probar el método `Send(...)` cuando se le envían argumentos nulos, el nombre del método de prueba debería ser algo así: `Send_NullArguments_ThrowsArgumentNullException()`.
- Los nombres de los métodos que prueban el *happy path* o el flujo esperado de un método deben ser iguales que el método que prueban. Por ejemplo, si se quiere probar el flujo esperado del método `Send(...)`, el nombre del método de prueba será `Send()`.
- Todos los métodos de prueba se dividen en 3 partes:
  - `Arrange`: es la parte del método de prueba donde se preparan los datos que se van a utilizar.
  - `Act`: es la parte del método de prueba donde se invoca la función o la parte de código que se quiere probar.
  - `Assert`: es la parte del método de prueba donde se verifican los resultados de la invocación anterior.

  En el código estas partes son sencillas de identificar, ya que se marcan con comentarios como se muestra en el código 8.1:

```
1         [Fact]
2         public void Send()
3         {
4             // Arrange
```

```
5               var service = (...);
6
7               // Act
8               service.Send(...);
9
10              // Assert
11              mock.Verify(...);
12              Assert.(...);
13          }
```

Listing 8.1: División en partes de un método de prueba

En las 2 secciones siguientes se hace un análisis de cobertura del código el cual se divide en 2 partes:

- **Cobertura de línea**: describe cuantas líneas de código sobre las totales han sido cubiertas por los tests
- **Cobertura de rama o condición**: describe cuantas condiciones sobre las totales han sido cubiertas por los tests.

8.2. Pruebas de la librería *MailService*

Las pruebas de la librería *MailService* han sido en su totalidad pruebas unitarias, es decir, son las que prueban una unidad funcional de código más pequeña (comúnmente funciones) [37]. Primero se implementaron todas las pruebas que pueden causar un fallo en el flujo esperado de la librería así como la comprobación de excepción cuando se pasan argumentos nulos, cadenas de caracteres vacíos, etc. Tras eso se implementó el test *happy path* para la consecución del envío del correo. Para evitar el envío de un email cada vez que se ejecutasen los tests, se usaron `Mocks` para simular ese envío. Esto fue el principal causante del cambio de diseño de la librería que se describió en la sección de diseño de la librería. (subsección 6.3).

En la tabla 8.1 se puede observar la cobertura de línea por paquetes de la librería *MailService*.

Paquete	**Nº de lineas cubiertas**	**Nº total de lineas**	**% de cobertura de linea**
GMV.ITS.CommonLibraries.Mail.Facade	134	145	92.40 %
Total	**134**	**145**	**92,40 %**

Cuadro 8.1: Tabla de cobertura de línea de la librería MailService

En la tabla 8.2 se puede observar la cobertura de rama o condición por paquetes de la librería *MailService*.

Paquete	**Nº de ramas cubiertas**	**Nº total de ramas**	**% de cobertura de rama**
GMV.ITS.CommonLibraries.Mail.Facade	30	32	93.70 %
Total	**30**	**32**	**93,70 %**

Cuadro 8.2: Tabla de cobertura de rama de la librería MailService

8.3. Pruebas del microservicio *Planner Integrator*

A diferencia que las pruebas de la librería, las pruebas del microservicio incluyen pruebas unitarias y pruebas de integración. Las pruebas de integración son aquellas que se realizan una vez que se han pasado las pruebas unitarias y prueban que todos los elementos unitarios que componen el software (en este caso el microservicio), funcionan juntos correctamente probándolos en grupo. Se centra principalmente en probar la comunicación entre los componentes y sus comunicaciones. [13].

Las pruebas de la capa de microservicio son en su totalidad tests unitarios, ya que prueban la funcionalidad de los servicios que son llamados desde la capa API. Al igual que las pruebas unitarias de la librería *MailService* se implementaron primero los tests de error y lanzamiento de excepciones y tras ellos los tests *happy path* haciendo uso de `Mocks` para simular las llamadas a otros microservicios del sistema Suite.

Las pruebas de la capa API son en su totalidad de integración, ya que prueban que el levantamiento de la API se realiza de forma correcta así como el funcionamiento de todos sus endpoints y llamadas a los servicios correspondientes.

En la tabla 8.3 se puede observar el resumen de cobertura de línea de la capa microservicio.

Paquete	**Nº de lineas cubiertas**	**Nº total de lineas**	**% de cobertura de linea**
GMV.ITS.PlannerIntegrator.Attribute	1	3	33,33 %
GMV.ITS.PlannerIntegrator.Definitions	7	7	100 %
GMV.ITS.PlannerIntegrator.Exceptions	2	8	25 %
GMV.ITS.PlannerIntegrator.Database	30	38	78,90 %
GMV.ITS.PlannerIntegrator.Model	24	24	100 %
GMV.ITS.PlannerIntegrator.Facade	234	234	100 %
Total	**298**	**314**	**94,90 %**

Cuadro 8.3: Tabla de cobertura de línea de la capa microservicio, `GMV.ITS.PlannerIntegrator`

En la tabla 8.4 se puede observar el resumen de cobertura de línea de la capa API.

Paquete	**Nº de lineas cubiertas**	**Nº total de lineas**	**% de cobertura de linea**
GMV.ITS.PlannerIntegrator.Api.Exceptions	3	3	100 %
GMV.ITS.PlannerIntegrator.Api.HostedServices	18	26	69,23 %
GMV.ITS.PlannerIntegrator.Api.Startup	42	42	100 %
GMV.ITS.PlannerIntegrator.Api.Controllers	32	32	100 %
Total	**95**	**103**	**92,23 %**

Cuadro 8.4: Tabla de cobertura de línea de la capa API, `GMV.ITS.PlannerIntegrator.Api`

En la tabla 8.4 se puede observar el resumen de cobertura de rama de la capa microservicio. Los paquetes cuya fila tienen 3 guiones en todas sus celdas no disponen de código con ramas o estructuras condicionales.

Paquete	**Nº de ramas cubiertas**	**Nº total de ramas**	**% de cobertura de rama**
GMV.ITS.PlannerIntegrator.Attribute	- - -	- - -	- - -
GMV.ITS.PlannerIntegrator.Definitions	- - -	- - -	- - -
GMV.ITS.PlannerIntegrator.Exceptions	- - -	- - -	- - -
GMV.ITS.PlannerIntegrator.Database	1	2	50 %
GMV.ITS.PlannerIntegrator.Model	- - -	- - -	- - -
GMV.ITS.PlannerIntegrator.Facade	19	20	95 %
Total	**20**	**22**	**90,90 %**

Cuadro 8.5: Tabla de cobertura de rama de la capa `GMV.ITS.PlannerIntegrator`

Parte III

Conclusiones y líneas futuras

Capítulo 9

Conclusiones

9.1. Consecución de los objetivos

Una vez concluido el proyecto, se puede decir que se han cumplido satisfactoriamente los objetivos del proyecto definidos en la sección 1.3, tanto los objetivos prácticos como los personales.

- **Objetivos prácticos**: se ha conseguido desarrollar el microservicio el cual ha conseguido lidiar con un problema de arquitectura en un sistema real y actualmente está funcionando y siendo utilizado por clientes de GMV. Este objetivo no hubiese sido posible sin un estudio y una adaptación previa a la nueva tecnología con la que se iba a desarrollar el proyecto.
- **Objetivos personales**: gracias a una formación previa al desarrollo del proyecto se ha conseguido el objetivo de manejar con fluidez las nuevas herramientas y tecnologías. También se ha conseguido trabajar en un entorno profesional siguiendo un flujo de trabajo y planificación real.

9.2. Líneas futuras

Las líneas de futuro contempladas para el proyecto son las siguientes:

- **Nuevas funcionalidades**: este proyecto tiene mucho potencial de escalabilidad, ya que los clientes y el sistema Suite de GMV gestionan muchos recursos que pueden ser gestionados por este microservicio como los horarios, asignaciones planificadas, topologías, etc. y con la gestión de estos recursos abre la posibilidad a más clientes de poder integrarse.
- **Uso de la base de datos local**: con la posibilidad de implementar nuevas funcionalidades y de la integración de más clientes, puede ser muy atractivo almacenar ciertos datos en la base de datos local que crea el microservicio con el objetivo de mejorar el rendimiento y el almacenamiento de datos

Bibliografía

[1] M. Cotterell y B. Hughes, *Software project management*. International Thomson Computer Press, 1995. (visitado 10-05-2024).

[2] K. Schwaber y J. Sutherland, "La guía de Scrum", *Scrumguides. Org*, vol. 1, pág. 21, 2013. (visitado 11-05-2024).

[3] S. Chacon y B. Straub, *Pro git*. Springer Nature, 2014.

[4] X. Larrucea, I. Santamaria, R. Colomo-Palacios y C. Ebert, "Microservices", *IEEE Software*, vol. 35, n.º 3, págs. 96-100, 2018. DOI: 10.1109/MS.2018.2141030.

[5] gewarren, *Información general acerca de .NET Framework - .NET Framework*, 2024. dirección: https://learn.microsoft.com/es-es/dotnet/framework/get-started/overview (visitado 11-04-2024).

[6] anandmeg, *¿Qué es el IDE de Visual Studio?* Dirección: https://learn.microsoft.com/es-es/visualstudio/get-started/visual-studio-ide?view=vs-2022 (visitado 05-04-2024).

[7] Atlassian, *Comparacón entre la arquitectura monolítica y la arquitectura de microservicios*. dirección: https://www.atlassian.com/es/microservices/microservices-architecture/microservices-vs-monolith (visitado 16-04-2024).

[8] Atlassian, *Elige herramientas para cada etapa del ciclo de vida de DevOps*. dirección: https://www.atlassian.com/es/devops/devops-tools (visitado 01-07-2024).

[9] Atlassian, *Entrega continua: primeros pasos con CI/CD | Atlassian*. dirección: https://www.atlassian.com/es/continuous-delivery (visitado 21-04-2024).

[10] Atlassian, *Sourcetree | Free Git GUI for Mac and Windows*. dirección: https://www.sourcetreeapp.com (visitado 05-04-2024).

[11] *Calculadora de IRPF - Castilla y León - 2024 Salario neto después de impuestos*. dirección: https://es.talent.com/tax-calculator?salary=25600&from=year&region=Castilla+y+Le%C3%B3n (visitado 21-05-2024).

[12] Colaboradores de los proyectos Wikimedia, *Principio de inversión de la dependencia - Wikipedia, la enciclopedia libre*. dirección: https://es.wikipedia.org/w/index.php?title=Principio_de_inversi%C3%B3n_de_la_dependencia&oldid=154101213 (visitado 13-06-2024).

[13] Colaboradores de los proyectos Wikimedia, *Prueba de integración - Wikipedia, la enciclopedia libre*. dirección: https://es.wikipedia.org/w/index.php?title=Prueba_de_integraci%C3%B3n&oldid=146725338 (visitado 22-06-2024).

[14] *Documentación de Jira*. dirección: https://confluence.atlassian.com/jira (visitado 05-04-2024).

[15] *Ecorresponsabilidad*. dirección: http://www.ecorresponsabilidad.es/fichas/portatil.htm (visitado 25-05-2024).

[16] *Facade*. dirección: https://refactoring.guru/es/design-patterns/facade (visitado 13-06-2024).

[17] Y. C. González-Carvajal, *Apuntes del Tema 3 de la asignatura Diseño de Software*. (visitado 14-06-2024).

[18] *HP 15S-eq2088ns AMD Ryzen 7 5700U/16GB/512GB SSD/15.6* . dirección: https://www.pccomponentes.com/hp-15s-eq2088ns-amd-ryzen-7-5700u-16gb-512gb-ssd-156 (visitado 25-05-2024).

[19] P. Kumar, "Facade Design Pattern in Java - Prabhu Kumar - Medium", *Medium*, dirección: https://prabhuch.medium.com/facade-design-pattern-in-java-29fe1436b1cd (visitado 13-06-2024).

[20] *Manifesto for Agile Software Development*. dirección: https://agilemanifesto.org (visitado 09-05-2024).

[21] markingmyname, *SQL Server Management Studio (SSMS) - SQL Server Management Studio (SSMS)*. dirección: https://learn.microsoft.com/es-es/sql/ssms/sql-server-management-studio-ssms?view=sql-server-ver16 (visitado 05-04-2024).

[22] *Microsoft 365 para empresas | Pequeñas empresas | Microsoft 365*. dirección: https://www.microsoft.com/es-es/microsoft-365/business (visitado 21-05-2024).

[23] *Monolítico frente a microservicios: diferencia entre arquitecturas de desarrollo de software - AWS*. dirección: https://aws.amazon.com/es/compare/the-difference-between-monolithic-and-microservices-architecture (visitado 30-05-2024).

[24] S. Nitzsche, "Toolbox of agile methods: SCRUM, Design Thinking and Co. - Energy BrainBlog", *Energy BrainBlog*, dirección: https://blog.energybrainpool.com/en/toolbox-of-agile-methods-scrum-design-thinking-and-co (visitado 30-05-2024).

[25] *NuGet Gallery | Home*. dirección: https://www.nuget.org (visitado 21-04-2024).

[26] *Opciones de precios y compra | Visual Studio*. dirección: https://visualstudio.microsoft.com/es/vs/pricing/?tab=business (visitado 21-05-2024).

[27] M. Ozkaya, "Microservices Communications - Design Microservices Architecture with Patterns & Principles - Medium", *Medium*, ISSN: 3198-7671. dirección: https://medium.com/design-microservices-architecture-with-patterns/microservices-communications-f319f8d76b71 (visitado 30-05-2024).

[28] *Pricing for Individual Licenses of Astah Software - Astah.* dirección: `https://astah.net/pricing/individual` (visitado 21-05-2024).

[29] M. Sarda, "Scrum Roles & Scrum events | Scrum Artifacts : Scrum Framework Overview", *Cloud Training Program*, dirección: `https://k21academy.com/scrum-master/scrum-roles-artifacts-events-scrum-framework` (visitado 30-05-2024).

[30] *Solicitud de extracción (Pull Request).* dirección: `https://www.techopedia.com/es/definicion/pull-request` (visitado 13-04-2024).

[31] *Sueldo: Junior Backend Developer en España en 2024.* dirección: `https://www.glassdoor.es/Sueldos/junior-backend-developer-sueldo-SRCH_KO0,24.htm` (visitado 21-05-2024).

[32] *ThinkPad P14s Gen 4 | Lenovo España.* dirección: `https://www.lenovo.com/es/es/p/laptops/thinkpad/thinkpadp/thinkpad-p14s-gen-4-(14-inch-intel)/21hfcto1wwes1` (visitado 24-05-2024).

[33] *Todo sobre la jornada completa: características, salario y modificaciones.* dirección: `https://payfit.com/es/contenido-practico/jornada-completa` (visitado 21-05-2024).

[34] *¿Cuánto cuesta el kilovatio hora de luz (kWh) en España?* Dirección: `https://tarifaluzhora.es/info/precio-kwh` (visitado 25-05-2024).

[35] *¿Cuánto tiempo duran los portátiles? Visión general para organizaciones | NinjaOne.* dirección: `https://www.ninjaone.com/es/blog/cuanto-tiempo-duran-los-ordenadores-portatiles-para-las-organizaciones` (visitado 24-05-2024).

[36] *¿Qué es Scrum? Conoce el Framework que agiliza el Trabajo en Equipo.* dirección: `https://www.escueladenegociosydireccion.com/revista/business/scrum-framework-agiliza-trabajo-equipo` (visitado 30-05-2024).

[37] *¿Qué son las pruebas unitarias?: explicación de las pruebas unitarias en AWS.* dirección: `https://aws.amazon.com/es/what-is/unit-testing` (visitado 22-06-2024).

Parte IV

Apéndices

Apéndice A

Manual de despliegue

En esta sección se describe todo lo necesario para desplegar el microservicio desarrollado en este proyecto.

Antes de comenzar con el manual es importante recalcar que este manual solo será útil para el personal de la empresa GMV, ya que es necesario el acceso a ciertas áreas de diferentes plataformas exclusivas de la empresa y se omitirán ciertos datos por cuestiones de privacidad.

A.1. Requisitos

A continuación se listan los requisitos previos que se deben tener instalados en la máquina para configurar y realizar el despliegue de forma correcta:

- .NET Core SDK (preferiblemente de la versión 3 para adelante)
- Maven
- SQL Server
- Rabbit MQ
- Git
- NuGet
- Node

A.2. Compilación y ejecución

El primer paso es bajar todos los microservicios esenciales para el correcto funcionamiento del sistema Suite. Una vez se han descargado todos hay que ir uno por uno ejecutando los comandos siguientes en el directorio raíz de cada microservicio: `dotnet clean && dotnet restore` y `dotnet build`.

Tras compilar todos los microservicios esenciales del sistema Suite, para cada uno de ellos se ejecutará el comando `dotnet run` para ejecutarlos.

Tras ello hacer estos 2 pasos anteriores con el microservicio *Planner Integrator* y ya estaría completamente desplegado.

Apéndice B

Manual de usuario

Este capítulo final se dedica a desarrollar el manual del usuario final.

B.1. Manual de Usuario

Para usar el microservicio *Planner Integrator* es imprescindible tener un usuario registrado en el sistema Suite de GMV.

B.1.1. Cambios online

Para realizar los cambios online ya sea de vehículos o de conductores, el usuario o cliente deberá llamar al endpoint que desee de los descritos en la figura 6.23 con los datos correspondientes descritos en la subsección 6.4.2. En las figuras B.2 y B.1 se pueden observar algunas peticiones de ejemplo desde Postman.

B.1.2. Carga planificada de vehículos

Para poder realizar la carga planificada de vehículos, el usuario o cliente deberá dejar preparados todos los vehículos que se usarán en el día correspondiente en su sistema para que el *Planner Integrator* pueda cargarlos en el sistema Suite.

Figura B.1: Ejemplo de petición desde Postman del reemplazo de conductores

Figura B.2: Ejemplo de petición desde Postman de la asignación de vehículos