Universidad de Valladolid

ESCUELA DE INGENIERÍA INFORMÁTICA

GRADO EN INGENIERÍA INFORMÁTICA
Mención en Ingeniería de Software

Desarrollo de una herramienta de exportación de datos de transporte estáticos de acuerdo al modelo GTFS de Google

---

Alumno: Luis Diez Laso

Tutor académico: Diego Rafael Llanos Ferraris

Tutora en empresa: Patricia Cadenas Quijano

...

Agradecimientos

A toda mi familia, en especial a mis padres, por todas las cosas que han hecho por mi.

A mis amigos y compañeros de universidad, por todos los momentos tanto buenos como malos que hemos pasado juntos a lo largo de la carrera.

A mis profesores de universidad por todos los conocimientos transmitidos en estos 4 años, en especial a mi tutor en este proyecto, Diego Rafael Llanos.

A mis compañeros de GMV, por su paciencia, su actitud y por todas las explicaciones en las llamadas de Teams. Muchas gracias a todos, en especial a mi tutora de proyecto en la empresa, Patricia Cadenas Quijano, por haberme dado la posibilidad de realizar este proyecto.

Resumen

El propósito de este documento es describir el proceso de análisis, diseño e implementación de una nueva funcionalidad dentro de uno de los microservicios de los que se compone el proyecto ITSSuite de la empresa GMV Sistemas S.A.U.

Explicándolo de manera resumida, esta nueva funcionalidad consiste en la exportación de datos de transporte del sistema de acuerdo al formato Gtfs de Google. Esta exportación debe de cumplir determinados requisitos impuestos tanto por los clientes como por la propia empresa que se describirán en secciones posteriores.

En este documento se dejará reflejada toda la vida del proyecto, desde la realización del diseño y el comienzo de la planificación hasta finalizar el trabajo requerido, haciendo de esta manera un seguimiento detallado del proceso de desarrollo. También se comentarán las tecnologías utilizadas y los resultados obtenidos tras haber finalizado la implementación de esta nueva funcionalidad.

Abstract

The purpose of this document is to describe the process of analysis, design and implementation of a new functionality in one of the microservices of which the ITSSuite proyect of the company GMV Sistemas S.A.U is composed of.

In a summed up way, this new functionality consists in the exportation of transportation data from the enterprise system following the criteria of Google´s Gtfs format. This exportation must follow certain requirements imposed both by the customers aswell as by the own company.

In this document, it will be reflected all the life of the proyect, from the making of the design and the beginning of the planification until the work is finished; thus, making a detail tracing of the proyect. It will also comment the technologies used and the results obtained after the implementation of this new functionality.

Índice general

Agradecimientos **III**

Resumen **V**

Abstract **VII**

Lista de figuras **XIII**

Lista de tablas **XV**

1. Introducción **1**

1.1. Contexto 1

1.2. Motivación 2

1.3. Objetivos 2

1.4. Acerca del estándar Gtfs 3

1.5. Estructura de la memoria 4

2. Requisitos y Planificación **5**

2.1. Metodología elegida 5

2.2. Requisitos y product backlog 6

2.3. Análisis de riesgos 8

2.4. Planificación 10

2.5. Presupuesto 10

3. Tecnologías utilizadas **11**

3.1. Herramientas de planificación y gestión 11

3.2. Herramientas de desarrollo 12

3.3. Tecnologías utilizadas en el desarrollo backend 12

3.4. Herramientas de validación 13

3.5. Otras herramientas 13

3.6. Despliegue 14

4. Diseño **15**

4.1. Introducción al diseño 15

4.2. Microservicios de la Suite a usar o modificar 17

4.3. Principios a seguir en diseño de microservicios 17

4.4. Diseño detallado 18

4.4.1. Arquitectura de la API 18

4.4.2. Estructura de paquetes 19

4.4.3. Controladores 20

4.4.4. Detalle lógica de negocio 20

4.4.5. Arquitectura de microservicios 22

5. Pruebas **25**

5.1. Testing con Xunit 25

5.2. Testing con validadores de Gtfs 26

5.3. Pruebas sobre endpoints 27

6. Seguimiento del proyecto **29**

6.1. Sprint 1 30

6.1.1. Retrospectiva de sprint 30

6.2. Sprint 2 34

6.2.1. Retrospectiva de sprint 34
6.3. Sprint 3 35
6.3.1. Retrospectiva de sprint 36
6.4. Sprint 4 36
6.4.1. Retrospectiva de sprint 37
6.5. Sprint 5 38
6.5.1. Retrospectiva de sprint 38
6.6. Sprint 6 38
6.6.1. Retrospectiva de sprint 39
6.7. Sprint 7 40
6.7.1. Retrospectiva de sprint 40
6.8. Sprint 8 41
6.8.1. Retrospectiva de sprint 41
6.9. Sprint 9 43
6.9.1. Retrospectiva de sprint 43
6.10. Semana añadida de manera extraordinaria 44
6.11. Conclusiones del seguimiento 45
6.11.1. Gestión de los riesgos 45
6.11.2. Costes reales 46

7. Conceptos sobre el despliegue **47**
7.1. Introducción 47
7.2. Service Discovery Pattern 47
7.2.1. Descubrimiento del lado del cliente 49
7.2.2. Descubrimiento del lado del servidor 50

8. Conclusiones **51**
8.1. Objetivos no cumplidos 52

8.2. Líneas de trabajo futuras 52

Bibliografía **53**

A. Resumen de enlaces adicionales **57**

B. Imágenes auxiliares **59**

B.1. Imágenes auxiliares 59

Lista de Figuras

4.1. Interfaz que Google proporciona a las compañías de transporte para subir sus feeds Gtfs 16

4.2. Diagrama subida de *feeds* a Nexus 16

4.3. Arquitectura de la API 18

4.4. Estructura de paquetes 19

4.5. Controladores de la API 20

4.6. Parte de lógica de negocio de la exportación 21

4.7. Dependencias internas del microservicio 22

4.8. Arquitectura del microservicio explicada de manera superficial 23

4.9. Diseño detallado con los métodos de las fachadas 24

6.1. Diagrama de dominio en análisis de microservicios topología y horarios 32

6.2. Diagrama de dominio en análisis de microservicio de importación y exportación de datos 33

7.1. Localización de servicios en soluciones Cloud. Imagen extraída de [35] 48

7.2. Patrón Service Discovery. Variante descubrimiento del lado del cliente. Imagen extraída de [46] 49

7.3. Patrón Service Discovery. Variante descubrimiento del lado del servidor. Imagen extraída de [46] 50

B.1. Tabla de análisis de validadores Gtfs 60

Lista de Tablas

2.1. Fechas de comienzo y fin de los sprints 6
2.2. *Product backlog* con las historias de usuario 7
2.3. Riesgo 1 8
2.4. Riesgo 2 8
2.5. Riesgo 3 9
2.6. Riesgo 4 9
2.7. Riesgo 5 9
2.8. Presupuesto 10

5.1. Tabla resultados obtenidos validación feed Salamanca 26
5.2. Tabla resultados obtenidos validación feed Reus 26
5.3. Roles y códigos asociados para pruebas con endpoints 27
5.4. Tabla resultados obtenidos de llamadas a endpoints 28

6.1. Tabla planificación sprint 1 30
6.2. Tabla planificación sprint 2 34
6.3. Tabla planificación sprint 3 35
6.4. Tabla planificación sprint 4 37
6.5. Tabla planificación sprint 5 38
6.6. Tabla planificación sprint 5 39

6.7. Tabla planificación sprint 7 40

6.8. Tabla planificación sprint 8 41

6.9. Tabla planificación sprint 9 43

6.10. Tiempo estimado y dedicado para cada sprint 45

6.11. Costes reales del proyecto 46

Capítulo 1

Introducción

1.1. Contexto

El transporte público es algo esencial para un buen porcentaje de la población en la sociedad actual. Con la llegada de la globalización y las nuevas tecnologías, ha aparecido tanto la necesidad como la posibilidad de poner en común los horarios de transporte y la información geográfica asociada a ellos de los diferentes lugares del mundo, de manera que desde cualquier punto del mundo, seamos capaces de poder ver esa información y planear nuestros viajes.

Este es el propósito del estándar Gtfs de Google, establecer una especificación de datos que permita a las agencias de transporte público publicar sus datos de tránsito en un formato que pueda ser consumido por una gran variedad de aplicaciones software (como por ejemplo, Google Maps[20] o BikeShareMap[6]).

Por ejemplo: en el caso de Google Maps, cuando lo usamos para ver las horas por las que pasan diferentes autobuses por una parada, o ver los puntos por los que pasa un viaje de autobús, lo que estamos viendo es la información publicada en forma de ***feeds*** **Gtfs** que han sido enviados a Google Transit por las personas que pertenecen a la organización propietaria de esos vehículos.

1.2. Motivación

El desarrollo de este proyecto se va a realizar durante el transcurso de las prácticas (tanto curriculares como extracurriculares) en la empresa GMV Sistemas S.A., trabajando en el equipo encargado del desarrollo de la ITS Suite (un conjunto de aplicaciones relacionadas con sistemas inteligentes de transporte). Antes de hablar con las personas que trabajan en este empresa, desconocía la existencia de este estándar, pero durante la entrevista me ofrecieron la posibilidad de realizar el trabajo de fin de grado con ellos sobre una herramienta que permitiera exportar los datos de transporte estático de su sistema por medio de *feeds* Gtfs.

Tras esta primera introducción al estándar, decidí informarme más, y al final acepté la oferta por varias razones:

- Creo que trabajar con un estándar desconocido puede ser muy interesante.
- Esta herramienta se va a implementar sobre un conjunto de aplicaciones de software ya desarrollado, teniendo alguna de ellas una complejidad a la que no me he enfrentado anteriormente; y creo que tener que trabajar con tanto "código" ya desarrollado e implementar una nueva funcionalidad sobre este es una oportunidad de aprender mucho.
- El software desarrollado va a ser una funcionalidad muy útil para muchas personas, desde las empresas de transporte que usan las aplicaciones de la Suite, hasta todas las personas que usan los vehículos de transporte de esas empresas. El hecho de que el software acabe siendo de utilidad a un número considerable de personas de manera directa creo que es la parte más motivadora de todo el proyecto.

1.3. Objetivos

El objetivo principal de este proyecto es la creación de una herramienta que permita a los miembros de las organizaciones que usan la ITS Suite generar de manera "manual" *feeds* Gtfs generados con su información de tránsito disponible en el sistema. Cuando hablamos de generar de manera manual los *feeds* obviamente no nos referimos a que el usuario escriba los *feeds*, si no a que el usuario solicita al sistema generar los *feeds* correspondientes a su organización, y el sistema los genera con la información de la Suite y se los devuelve.

En el caso de la exportación manual, si el usuario quiere enviar los *feeds* que ha generado a distintas aplicaciones, tendría que hacerlo cada vez que genera un nuevo feed. Una vez hecha la exportación manual (y si el tiempo lo permite), pretende automatizarse el proceso de exportación de manera que los *feeds* se generen cada cierto tiempo configurable y se envíen de manera automática a las aplicaciones necesarias sin necesidad del usuario de intervenir en el proceso.

En el intento de conseguir estos objetivos también se pretende conseguir otros "objetivos de aprendizaje", de nuevas herramientas de programación como Visual Studio 2019[40], de un nuevo lenguaje de programación como es C#[43] y un framework nuevo como .NET CORE[25].

1.4. Acerca del estándar Gtfs

La creación del estándar Gtfs como lo conocemos hoy en dia, comenzó como un proyecto secundario para el empleado de Google "Chris Harrelson" en 2005. En aquella época, la información de tránsito en internet de muchas ciudades era inexistente. Chris conoció a Tim y Bibiana MCHugh, una pareja que trabajaba en una compañía de transporte de Portland, Oregon, y les habló de como estaba frustrado de no encontrar direcciones a seguir en ciudades desconocidas, mientras que en otras ciudades, había servicios que ya indicaban direcciones a seguir mientras que iba conduciendo.

Con el tiempo, la pareja se puso en contacto con Google proporcionándo a la compañía ficheros CSV con la información de tranporte de la compañía en la que trabajaban. De esta manera, Portland se convirtió en diciembre de 2005 en la primera ciudad en formar parte del "Google's Transit Trip Planner"[16]. En septiembre de 2006, otras 5 ciudades de los Estados Unidos fueron añadidas al "Google's Transit Trip Planner", y el formato de datos con el que se proporcionaba la información de transporte paso a ser conocido como *"Google Transit Feed Specification"*[19].

Con el tiempo, y debido al amplio uso de este formato, la parte de "Google"del nombre original fue visto como algo que podía asustar a algunos usuario potenciales de adoptar la especificación. Como consecuencia, en 2009, se cambió su nombre a *"General Transit Feed Specification"*.

A continuación, se va a hacer una descripción general de los ficheros que el software desarrollado para este proyecto será capaz de generar y se indicará si siempre son necesarios en un feed Gtfs para dar una visión general al lector del estándar. Se recomienda encarecidamente al lector que intente leer la especificación completa[17], ya que más adelante se hará referencia a campos específicos de algunos de los ficheros.

- **agency.txt**: define las empresas de transporte público que tienen un servicio representado en el conjunto de datos. Fichero obligatorio.
- **stops.txt**: define las paradas en las que los vehículos recogen o dejan pasajeros. También indica las estaciones y las entradas de las estaciones. Fichero obligatorio.
- **routes.txt**: rutas de transporte público. Una ruta es un grupo de viajes que se muestra a los pasajeros como un solo servicio. Fichero obligatorio.
- **trips.txt**: viajes para cada ruta. Un viaje es una secuencia de dos o más paradas que ocurre durante un período específico. Fichero obligatorio.
- **stop_times.txt**: proporciona las horas en las que un vehículo llega a una parada y sale de ella en cada viaje.
- **calendar_dates.txt**: contiene todas las fechas de servicio, y si el servicio va a estar activo o no ese día. Si se incluyera el archivo calendar.txt[15], este archivo no sería obligatorio, al no incluirlo pasa a ser obligatorio y ha de contener todas las fechas de servicio.

- **shapes.txt**: define las reglas para asignar las rutas de viaje de los vehículos, también conocido como alineamientos de rutas. Fichero opcional.

1.5. Estructura de la memoria

Este documento se estructura de la siguiente forma:

Capítulo 2 Requisitos y planificación: se describen los requisitos de usuario, los riesgos identificados en el proyecto y se elegirá una metodología de planificación para el proyecto.

Capítulo 3 Tecnologías utilizadas: se presentan las diferentes tecnologías y herramientas utilizadas durante el proyecto y una pequeña descripción de ellas.

Capítulo 4 Diseño: describe el diseño de la nueva funcionalidad a implementar y que patrones de diseño se usan en el propio.

Capítulo 5 Pruebas: describe todos los tipos de pruebas que se han hecho a lo largo del desarrollo, haciendo referencia en cada tipo de prueba a la herramienta utilizada para hacerlas.

Capítulo 6 Seguimiento del proyecto: describe, para cada sprint, varios aspectos:

- Al comienzo del sprint: planificación de las tareas que se van a realizar, junto con esfuerzo estimado.
- Al final del sprint: retrospectiva de las tareas realizadas. Cuanto tiempo ha llevado y ver como se ajusta a lo planeado.

La información se mostrará en tablas, en las que se verán las tareas asignadas para el sprint, junto con el esfuerzo estimado y el estado que tienen las tareas al final del sprint.

Capítulo 7 Conclusiones: describirá cuáles han sido los objetivos cumplidos, los que no lo han sido y las nuevas funcionalidades y cambios que podrían implementarse en el proyecto en el futuro.

Anexo A Resumen de enlaces adicionales: incluye enlaces de interés sobre el proyecto.

Apéndice B Imágenes auxiliares: incluye imágenes que no se han mostrado en los capítulos de la memoria pero pueden ser de utilidad.

Capítulo 2

Requisitos y Planificación

2.1. Metodología elegida

Para llevar a cabo la planificación de este proyecto, se ha decidido usar la metodología SCRUM[13], ya que es la que metodología utilizada en el equipo de desarrollo de producto dentro del que se va a llevar a cabo.

Las únicas diferencias a destacar de este proyecto planificado siguiendo esta metodología respecto de otros es el hecho de que el equipo de desarrollo estará formado por una única persona (el autor de este documento).

Siguiendo el desarrollo iterativo e incremental que establece la metodología SCRUM, los *requisitos de usuario* se recogen en forma de historias de usuario junto con una cantidad de *"story points"* que indican el esfuerzo requerido estimado para llevar a cabo esa historia de usuario. En este proyecto, cada *"story point"* es equivalente a unas 8 horas de trabajo. Al principio de cada *sprint* tendrá lugar una reunión conocida como *"Sprint planning"* en la que se asignarán al mismo una serie de tareas obtenidas a partir de las historias de usuario recogidas en el *product backlog*.

Cada *sprint* tendrá una duración de dos semanas, y en principio habrá 8 *sprints*. El primer *sprint* comenzará el 8 de febrero de 2022; por lo que en principio, el desarrollo se finalizará el lunes 31 de mayo, fecha de fin del octavo *sprint* (esta es una fecha estimada, puede que el desarrollo lleve más tiempo de lo esperado).

Sprint	Fecha de inicio	Fecha de fin
1	8/02/2022	21/02/2022
2	22/02/2022	07/03/2022
3	08/03/2022	21/03/2022
4	22/03/2022	04/04/2022
5	05/04/2022	18/04/2022
6	19/04/2022	02/05/2022
7	03/05/2022	16/05/2022
8	17/05/2022	30/05/2022

Tabla 2.1: Fechas de comienzo y fin de los sprints

2.2. Requisitos y product backlog

En el *product backlog* se recogen los requisitos de usuario en forma de historias de usuario que se pretenden implementar, y se les asigna una estimación del esfuerzo necesario para llevarlas a cabo (*story points*), y una prioridad establecida en función de su retorno sobre la inversión. Esta prioridad nos servirá para saber que historias de usuario son las que debemos de ir asignando primero a lo largo de los *sprints* durante el desarrollo del proyecto.

En este proyecto, se podría decir que solo tenemos una historia de usuario, ya que realmente lo único que el usuario desea es generar *feeds* Gtfs estáticos correspondientes a los datos de su organización.

Como esta historia de usuario realmente es muy general, se va a dividir en historias más pequeñas en las que el esfuerzo a dedicar a la funcionalidad a implementar sea más fácil de estimar.

Historia de usuario	Story points	Prioridad
Como usuario quiero poder exportar las agencias del sistema para poder enviar esa información a Google Transit (**HU-001**)	1	1
Como usuario quiero poder exportar los feed Gtfs en conjunto o por departamento dependiendo de un parámetro de configuración (**HU-002**)	2	2
Como usuario quiero poder exportar las paradas del sistema para poder enviar esa información a Google Transit (**HU-003**)	2	3
Como usuario quiero poder exportar las rutas del sistema para poder enviar esa información a Google Transit (**HU-004**)	2	4
Como usuario quiero poder exportar los recorridos del sistema para poder enviar esa información a Google Transit (**HU-005**)	1	5
Como usuario quiero poder exportar los viajes del sistema para poder enviar esa información a Google Transit (**HU-006**)	5	6
Como usuario quiero poder exportar las fechas en las que el servicio está activo o no del sistema para poder enviar esa información a Google Transit (**HU-008**)	5	6
Como usuario quiero poder exportar los tiempos de parada del sistema para poder enviar esa información a Google Transit (**HU-007**)	2	7
Como usuario quiero tener la seguridad de que los *feeds* generados están bien formados respecto al estándar y cumplen todas sus reglas antes de enviarlos a Google Transit. (**HU-010**)	3	8
Como usuario quiero poder exportar las tarifas del sistema para poder enviar esa información a Google Transit (**HU-009**)	5	9
Como usuario quiero que la exportación y obtención de los *feeds* por parte de Google Transit se haga de manera automática sin tener que intervenir en el proceso (**HU-011**)	5	10

Tabla 2.2: *Product backlog* con las historias de usuario

2.3. Análisis de riesgos

En esta sección se ha decidido seguir las recomendaciones dadas en el libro "Software Project Management. 5th Edition"(por ser el usado en la asignatura "Planificación y gestión de proyectos") para identificar los riesgos, categorizarlos y asignarles tanto una probabilidad de su ocurrencia y un impacto, como una manera de lidiar con estos (ya sea aceptando, evitando, estableciendo un plan de mitigación que reduzca su impacto o transfiriéndolo a otra persona u organización).

Para identificar los riesgos se ha recurrido a dos técnicas:

1. Brainstorming: en este caso es un poco distinto, ya que no se puede recurrir a los stakeholders del proyecto, y el equipo de desarrollo esta formado por una sola persona. Por tanto, en este caso, el desarrollador será quien desempeñe la labor de stakeholder. Antes de comenzar el primer *sprint*, el desarrollador ha dedicado 1 hora a pensar sobre los posibles riesgos que pueden surgir en este proyecto.

2. Uso de una checklist con riesgos dados en proyectos software ya completados.

A la hora de categorizarlos se dividirán en riesgos de proyecto (si pueden afectar a la completitud de los objetivos dados al equipo de desarrollo del proyecto) y riesgos de negocio (si amenazan que el software sea rentable económicamente).

Riesgo	Falta de formación del equipo de desarrollo
Categoria	Riesgo de proyecto
Probabilidad	Alta
Impacto	Medio
Plan	En este caso se ha establecido un plan de mitigación. Ha de dedicarse parte de las horas a familiarizarse con el lenguaje y herramientas a utilizar. Pedir ayuda a los otros miembros del equipo siempre que sea necesario.

Tabla 2.3: Riesgo 1

Riesgo	Retrasos respecto de lo planificado
Categoria	Riesgo de proyecto
Probabilidad	Media
Impacto	Alto
Plan	En este caso se ha establecido un plan de mitigación. A la hora de establecer el comienzo del proyecto, se ha decidido comenzar antes de lo necesario, de manera que en caso de que ocurra un imprevisto, podría añadirse un *sprint* más, y se podría seguir finalizando el proyecto antes de la fecha límite de entrega.

Tabla 2.4: Riesgo 2

Riesgo	Falta de disponibilidad del desarrollador por causas externas como enfermedades
Categoria	Riesgo de proyecto
Probabilidad	Media
Impacto	Alto
Plan	En este caso se ha establecido un plan de mitigación. A la hora de establecer el comienzo del proyecto, se ha decidido comenzar antes de lo necesario, de manera que en caso de que ocurra un imprevisto, podría añadirse un *sprint* más, y se podría seguir finalizando el proyecto antes de la fecha límite de entrega.

Tabla 2.5: Riesgo 3

Riesgo	Cambios en el estándar Gtfs
Categoria	Riesgo de proyecto
Probabilidad	Bajo
Impacto	Medio
Plan	En este caso se ha decido aceptar el riesgo. El desarrollador del proyecto se ha informado de como suelen ser los cambios que se plantean en los estándar de Google, en la mayoría de los casos suelen ser pequeños cambios,fáciles de implementar y muchas veces opcionales, para que la mayor parte de los sistemas que trabajan con estos estándares puedan seguir funcionando.

Tabla 2.6: Riesgo 4

Riesgo	Cambios en los requisitos
Categoria	Riesgo de proyecto
Probabilidad	Bajo
Impacto	Medio
Plan	En este caso se ha establecido un plan de mitigación. A la hora de establecer el comienzo del proyecto, se ha decidido comenzar antes de lo necesario, de manera que en caso de que ocurra un imprevisto, podría añadirse un *sprint* más, y se podría seguir finalizando el proyecto antes de la fecha límite de entrega.

Tabla 2.7: Riesgo 5

2.4. Planificación

Al tratarse este de un proyecto en el que se ha decidido seguir la metodología SCRUM, la planificación se realizará al principio de cada *sprint*, y se verá reflejada en el apartado 6 de seguimiento del proyecto.

2.5. Presupuesto

A la hora de calcular el presupuesto para este proyecto hay que tener en cuenta que solo se han realizado tareas de desarrollo de back-end, por lo tanto, se ha utilizado como salario el de un desarrollador back-end.

Para calcular el salario de un desarrollador back-end se ha utilizado el sitio web Glassdoor[39], donde actuales y antiguos empleados revisan sus salarios de manera anónima. De acuerdo con este sitio web el sueldo medio nacional de desarrollador back-end trabajando en jornada completa en España es de 36337 €/año, esto equivale a 18,92 €/hora. En este proyecto, se estima que se trabaje unas 20 horas/semana, durante 16 semanas, por lo tanto el presupuesto asignado a esta parte sería 6054,4 €.

Durante esas 16 semanas se pretende realizar como mínimo la parte correspondiente a la exportación manual. Una vez completada se planteará si incluir la automatización o no. El presupuesto calculado en este apartado se corresponde solamente a la parte de exportación manual.

Habiendo calculado el presupuesto asignado a la parte de pago a personal, solo queda el coste de los equipos de desarrollo, que en este caso, al hacerse el desarrollo con el equipo proporcionado por la empresa donde se están realizando las prácticas no se tiene en cuenta.

Se ha decidido; siguiendo la linea del plan de mitigación de alguno de los riesgos (por los que en caso de ser necesario, se puede alargar el proyecto un *sprint* más de lo necesario); asignar un porcentaje de presupuesto extra al ya establecido para tener una especie de "colchón" en caso de que alguno de los riesgos estudiados provocara problemas. En este caso el colchón será el gasto correspondiente a un *sprint* extra de trabajo del desarrollador.

Descripción de gasto	Total asignado
Tiempo de trabajo de personal	6054,4 €
Tiempo de trabajo de personal en sprint extra	756,8 €
Total	6811,2 €

Tabla 2.8: Presupuesto

Capítulo 3

Tecnologías utilizadas

A continuación se presentan las tecnologías y herramientas utilizadas para la realización del proyecto, junto con una pequeña descripción de cada una.

3.1. Herramientas de planificación y gestión

- **Jira**
  En el caso de los equipos que usan metodologías ágiles, Jira[5] proporciona tableros de scrum[13] y kanban[12] listos para usar. Los tableros permiten gestionar las tareas y ofrecen transparencia sobre el trabajo en equipo y visibilidad del estado de cada elemento de trabajo. Jira también proporciona funciones de seguimiento del tiempo e informes de rendimiento en tiempo real (diagrama de trabajo pendiente o de trabajo completado, informes de sprints, gráficos de velocidad) que permiten a los equipos evaluar su productividad.

- **Bitbucket**
  Bitbucket[3] es una herramienta de alojamiento de código y control de versiones basada en Git. La principal razón de usar Bitbucket frente a otras herramientas de control de versiones es que se integra con Jira de manera que puedas trabajar con las dos herramientas a la vez permitiendo por ejemplo crear nuevas ramas asociadas a tareas desde la interfaz de Jira. Tanto Jira como Bitbucket son productos de la empresa Atlassian[2].

- **Sourcetree**
  Sourcetree[38] es una herramienta que simplifica la manera de interactuar con los repositorios de Git. Proporciona una interfaz gráfica con la que se pueden manejar los repositorios de manera muy intuitiva. Además de simplificar la manera en la que se interactúa con los repositorios, también se integra con Bitbucket.

- **Confluence**
  Confluence[4] es una herramienta de documentación colaborativa. Se ha usado para documentar lo que se ha visto necesario en el desarrollo (como se verá en el seguimiento, principalmente para explicar el mapeo de entidades de la Suite a Gtfs y documentar el proceso de selección de un validador para los *feeds* Gtfs).

3.2. Herramientas de desarrollo

- **diagrams.net**
  diagrams.net[8] es una página web que permite crear diagramas de manera intuitiva y con bastantes elementos personalizables. Ha sido la opción escogida para realizar diagramas explicativos como el de la figura 4.2.

- **Astah**
  Astah[1] es una herramienta de modelado que permite crear diagramas de clases, secuencia, casos de uso, paquetes, etc. Se ha elegido usar esta herramienta debido a que se tiene experiencia con ella por su uso en asignaturas que se han cursado en el grado.

- **Visual Studio (versión de 2019)**
  Visual Studio[41] es un entorno de desarrollo integrado. Es compatible con múltiples lenguajes de programación, tales como C++, C#, Visual Basic .NET, etc. En el caso de este proyecto se ha utilizado para desarrollar aplicaciones con el framework .NET Core 3.1[25]. Se ha escogido este como IDE por varias razones:

  - Es el elegido por la mayoría de los desarrolladores de la Suite a la hora de hacer backend.
  - Puede conectarse con un servidor de SonarQube que analice el código para que se tenga las reglas que se ha de seguir a la hora de desarrollar.
  - Gran cantidad de plugins y documentación sobre este IDE, lleva siendo usado muchos años en sus diferentes versiones.

- **Postman**
  Postman[37] es una plataforma para construir y usar APIs. En este caso se ha usado para realizar pruebas durante el desarrollo antes de realizar tests de otro tipo.

3.3. Tecnologías utilizadas en el desarrollo backend

- **C#**
  C# [43] es un lenguaje de programación multiparadigma desarrollado y estandarizado por la empresa Microsoft como parte de su plataforma .NET. Su sintaxis básica deriva de C/C++ y utiliza el modelo de objetos de la plataforma .NET, similar al de Java, aunque incluye mejoras derivadas de otros lenguajes.

- **.NET Core[25]**
  .NET[11] es una plataforma de desarrollo (lo que es lo mismo que decir una colección

de lenguajes y bibliotecas que pueden trabajar conjuntamente para crear aplicaciones)gratis y de código abierto que permite desarrollar aplicaciones multiplataforma. .NET 5+ (incluido .NET Core[25]) se ejecuta en cualquier plataforma. Con .NET se pueden crear aplicaciones de todo tipo: microservicios (como es el caso de este proyecto), servicios Cloud, aplicaciones móviles, web, IoT, etc.

- **Entity framework Core**
  Entity Framework Core[23] es un asignador de base de datos de objeto moderno para .NET. Admite consultas LINQ[24], seguimiento de cambios, actualizaciones y migraciones de esquemas. EF Core funciona con una gran variedad de bases de datos, incluidas SQL Database (en el entorno local y Azure), SQLite, MySQL, PostgreSQL y Azure Cosmos DB. Durante el proyecto, no se tienen que crear nuevos datos persistentes, pero si ha sido necesario añadir nuevas consultas para optimizar el proceso de exportación, eso ha hecho necesario comprender de manera básica como funciona Entity Framework.

- **Microsoft SQL Server Management Studio 18** Es un entorno integrado para administrar cualquier infraestructura de SQL. Se ha utilizado para visualizar los datos en el entorno local sin tener que estar realizando peticiones con Postman constantemente y así hacer pruebas y comprobar que los resultados obtenidos se corresponden con los esperados antes de la realización de los test.

- **XUnit**[45] Librería utilizada para hacer testing unitario en el framework .NET.

3.4. Herramientas de validación

A la hora de validar los feed Gtfs generados se han usado varias herramientas:

- **Reflect foursquare gtfs validator**[14]
- **Gtfsvtor**[29]
- **Feed validator**[31]
- **Mobility data gtfs validator**[28]

3.5. Otras herramientas

- **LaTeX**[42] es un sistema de composición de textos, orientado especialmente a la creación de libros, documentos científicos y técnicos que contengan fórmulas matemáticas. LaTeX está formado por un gran conjunto de macros de *TeX*[44]. Es muy utilizado para la composición de artículos académicos, tesis y libros técnicos, dado que la calidad tipográfica de los documentos realizados con LaTeX es comparable a la de una editorial científica de primera línea.

- **Overleaf**[36] es un editor colaborativo de LaTeX basado en la nube que se utiliza para escribir, editar y publicar documentos científicos. Ha sido usado para escribir esta memoria, principalmente por el hecho de no tener que instalar ninguna herramienta para ser capaz de escribir documentos con *LaTeX*.

3.6. Despliegue

Aunque en este proyecto no haya que hacer un despliegue como tal (el despliegue se realiza de manera automática), se van a mencionar las herramientas que se usan:

- **Docker**[9]: automatiza la implementación de aplicaciones como contenedores portátiles y autosuficientes (se incluyen todas las dependencias en el propio contenedor) que se pueden ejecutar en la nube o localmente. De esta manera, los contenedores Docker permiten aislar el software de su entorno y garantizan que funcione. El fichero de texto que contiene las instrucciones a partir de las que se crea una imagen Docker se llama Dockerfile[10]. Lo que llamamos contenedor Docker es una instancia de una imagen Docker en ejecución.
- **Kubernetes**[21] es un software de orquestación de código abierto que ofrece una API para controlar la forma y el lugar en que se ejecutarán los contenedores. Permite ejecutar contenedores y cargas de trabajo de Docker y dar solución a algunas de las complejidades de funcionamiento al escalar varios contenedores implementados en varios servidores.

  Con Kubernetes, podemos organizar un clúster de máquinas virtuales y programar los contenedores para que se ejecuten en esas máquinas según los recursos de proceso disponibles y los requisitos de recursos de cada contenedor. Los contenedores se agrupan en pods, que es la unidad operativa básica de Kubernetes.

Capítulo 4

Diseño

4.1. Introducción al diseño

En esta sección se va a hacer la descripción detallada del diseño de nuevas funcionalidades dentro de microservicios existentes de la Suite.

A la hora de realizar los diagramas las funcionalidades ya implementadas antes del comienzo de este proyecto y que se usan se representarán como componentes de caja negra.

Primero se van a enumerar los microservicios de la Suite que ha sido necesario usar y se dará una descripción superficial de que hace cada uno de ellos, se hará una pequeña explicación de los principios a seguir a la hora de diseñar la nueva funcionalidad, y por último se mostrará el diseño detallado de la nueva funcionalidad para cada uno de los microservicios.

Por si todavía no ha quedado claro como es posible automatizar el proceso de exportación, vamos a ver las posibilidades que Google ofrece a las compañías de transporte para subir sus *feeds*:

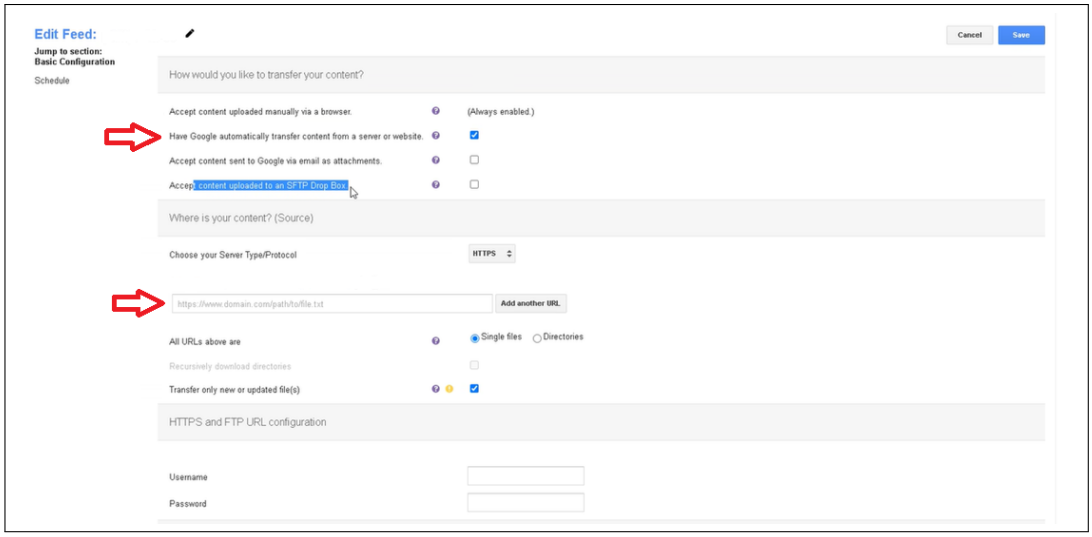

Figura 4.1: Interfaz que Google proporciona a las compañías de transporte para subir sus feeds Gtfs

Como se puede ver en la figura 4.1, el cliente puede indicar a Google una URL desde la que cargar los *feeds* (segunda flecha de color rojo) e indicar que transfiera los feed automáticamente desde esa URL (primera flecha de color rojo). Como se puede ver en la interfaz (en la linea superior a la indicada por la primera línea roja), también es posible para los usuarios subir los *feeds* manualmente.

Viendo estas opciones, es posible subir los feeds generados cada cierto tiempo a una URL accesible por Google, y que Google automáticamente comience a usarlos (tras haber pasado por una validación que hace la propia compañía). En nuestro caso, como dirección a la que subir los *feeds*, se ha elegido un repositorio de Nexus. Esto queda representado en la figura 4.2.

Figura 4.2: Diagrama subida de *feeds* a Nexus

4.2. Microservicios de la Suite a usar o modificar

Microservicios en los que se va a tener que implementar nueva funcionalidad:

- **ms-import-export**: como el propio nombre indica, es el encargado de hacer la importación de datos al sistema. Este microservicio será en el que se va a implementar la generación/exportación de los *feeds* Gtfs, y el único necesario en caso de que el usuario quiera exportar *feeds* manualmente.

Microservicios necesarios para cargar datos de exportación:

- **ms-topology**: como el propio nombre indica es el que trabaja con todos los datos relacionados con la topología de cada organización como podrían ser las paradas, las rutas, los puntos de los que se compone una ruta,etc.
- **ms-timetable**: es el microservicio que trabaja con todos los datos relacionados con los horarios de cada organización como pueden ser las horas de pase por parada, los viajes que se realizan, etc.

Además del microservicio en el que se desarrollará la funcionalidad correspondiente a la exportación manual y los necesarios para cargar datos de exportación para la parte automatizada será necesario que el primero de estos interactúe con otros que se encargan de proveer datos sobre las organizaciones y los usuarios, de ejecutar tareas programadas y de subir los *feeds* a un repositorio.

4.3. Principios a seguir en diseño de microservicios

A la hora de diseñar microservicios, es conveniente seguir el enfoque de diseño guiado por el dominio (*Domain Driven Design*). Este enfoque es diferente al que tomamos normalmente en el que se hace un solo modelo de dominio para representar todo el sistema. En el enfoque del *Domain Driven Design*, dividimos ese dominio del sistema global en *subdominios* en función de las áreas del dominio global que tratan *conceptos diferentes*.

Como bien dice el refrán: *"largo es el camino de la enseñanza por medio de teorías; pero breve y eficaz por medio de ejemplos"*, seguro que se entiende mejor en que consiste el *Domain Driven Design* con un ejemplo: en el caso del dominio global del sistema de la Suite se puede diferenciar el subdominio de la topología de los demás porque trata conceptos diferentes de los demás subdominios posibles como son las paradas, las rutas, etc. En las figuras 6.1 y 6.2 se puede ver una separación inicial de parte del modelo de dominio global de las aplicaciones de la Suite en subdominios en función de los conceptos diferentes que se tratan.

El otro aspecto que quiero tratar en esta sección es la comunicación entre microservicios. Cada microservicio tiene su propia base de datos, y es el único capaz de acceder a ellos. Cuando un microservicio necesita acceder a datos de otro, ha de hacerlo a través de peticiones a la api de este segundo.

4.4. Diseño detallado

En esta parte van a mostrarse los diagramas de diseño que se ha creído convenientes hacer. En caso de considerarse necesario, también se hará una descripción de algunas partes de los mismos.

Los diagramas mostrados en este apartado se corresponden con el diseño detallado del desarrollo hecho en el microservicio que se encarga de la importación y la exportación.

4.4.1. Arquitectura de la API

El diseño de la API es realmente sencillo, permite obtener el feed de la organización a la que pertenezca el usuario que hace la llamada, obtener el mismo feed pero con una validación incluida, subir *feeds* de una determinada organización a un repositorio y programar los jobs que se encargan de subir los *feeds* a un repositorio.

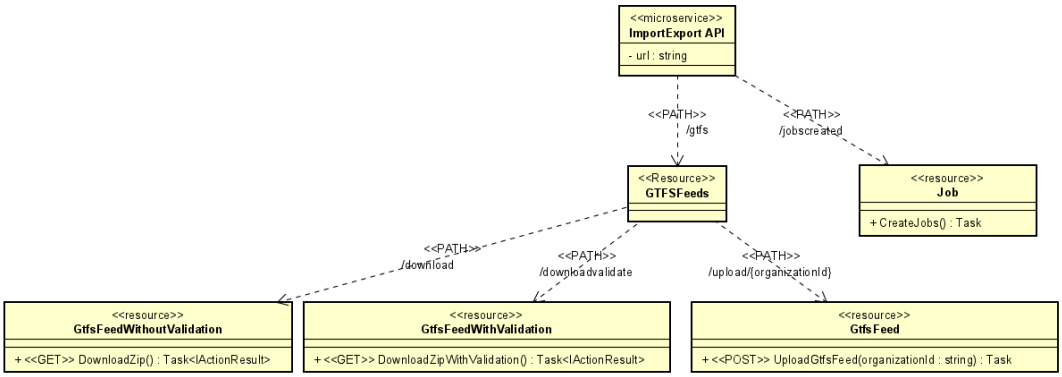

Figura 4.3: Arquitectura de la API

4.4.2. Estructura de paquetes

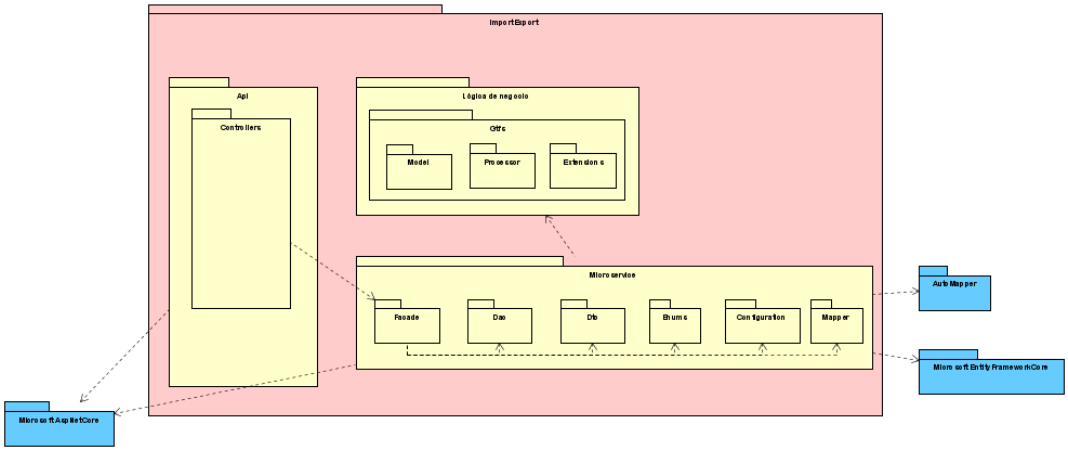

Figura 4.4: Estructura de paquetes

Respecto a la estructura de paquetes y las dependencias entre ellos, vemos como se dividen en 3 principales: la parte de API, la de microservicio y la de lógica de negocio (hay otro paquete principal para los tests pero se ha obviado en la parte de diseño).

Se puede ver por las dependencias como se utiliza el **patrón fachada**, por el que se proporciona una interfaz simple a un sistema más o menos complejo. De esta manera la mayoría de los métodos de la API consisten en una sola llamada a una interfaz de la fachada, y ya en la implementación de ese método se trata toda la complejidad (en las flechas que van de la fachada a los demás subpaquetes se puede ver representado como la fachada esconde más complejidad).

En el paquete que se corresponde con la lógica de negocio, iría toda la funcionalidad correspondiente con por ejemplo procesar ficheros de entrada, o las clases que se encargan de escribir los ficheros que luego se exportan, toda la parte de "cómputo" más pesada.

4.4.3. Controladores

En el caso de los controladores de la API la funcionalidad correspondiente a la exportación manual y la exportación que incluye una validación de los *feeds* será accessible por todos los clientes que tengan el permiso de exportación necesario.

La parte correspondiente tanto a la subida automática de los *feeds* como a la programación de los jobs solo será accesible para un usuario administrador o desde las apis. Como veremos más tarde, la parte de programación de los jobs que arrancan la subida de ficheros a un repositorio se ejecutará sin necesidad de que un usuario tenga que intervenir.

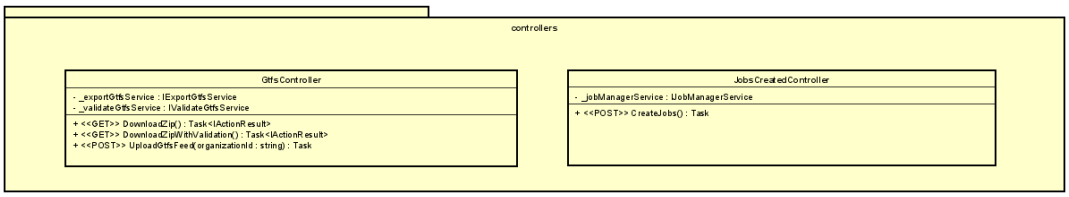

Figura 4.5: Controladores de la API

4.4.4. Detalle lógica de negocio

En el siguiente diagrama se muestra todo el diseño correspondiente a la parte de la exportación que dadas unas entidades que se corresponden con las especificaciones Gtfs, escribe los ficheros que forman el feed que tiene la información de esas entidades.

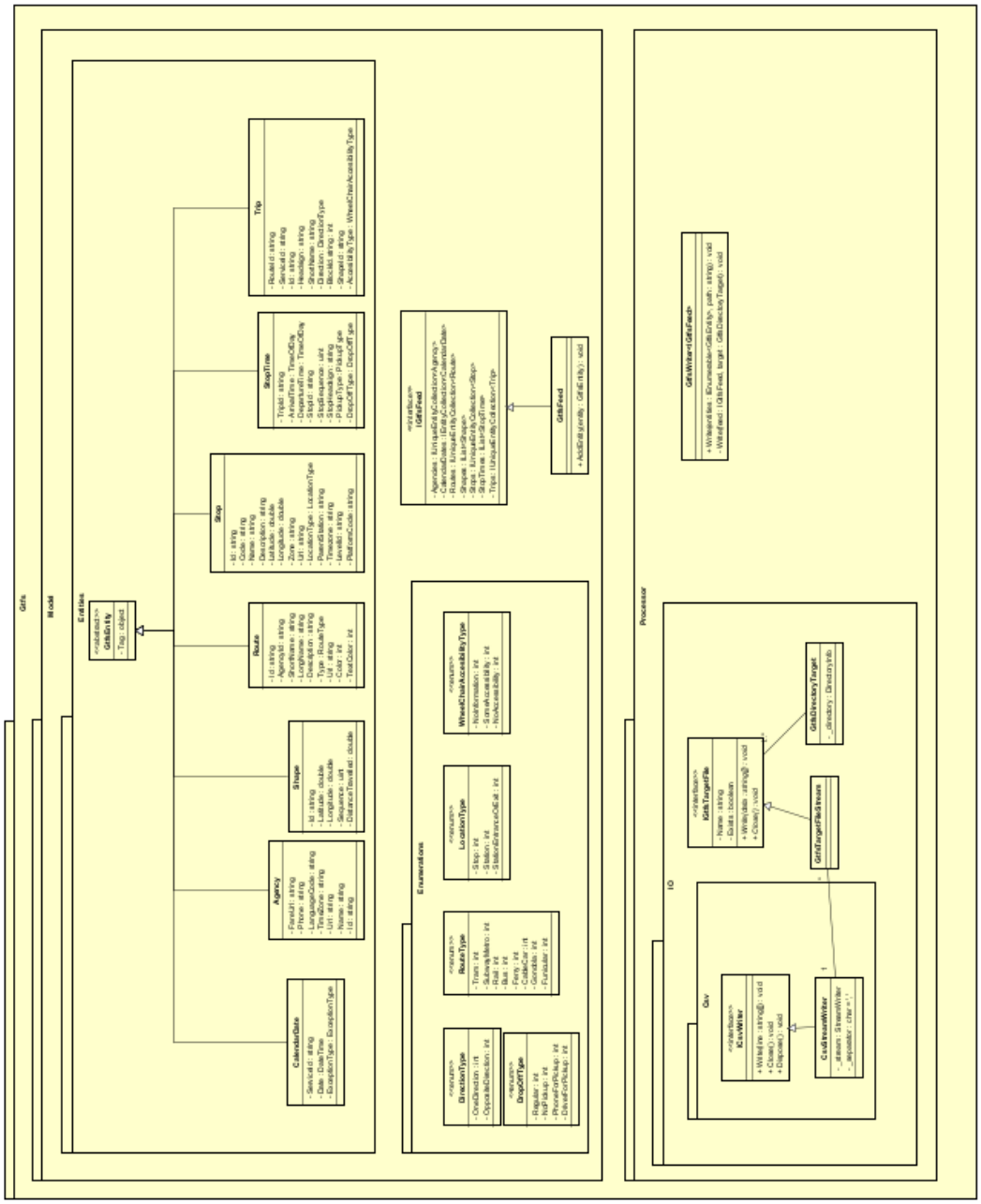

Figura 4.6: Parte de lógica de negocio de la exportación

4.4.5. Arquitectura de microservicios

En la figura 4.8 se pueden ver los aspectos que más nos interesan del microservicio. En la figura 4.9 se muestra el diseño detallado de cada uno de los servicios mostrados en la figura 4.8.

Por si el lector considera que hay muchos elementos en la figura 4.8, la figura 4.7 muestra las dependencias internas entre los propios servicios de la fachada.

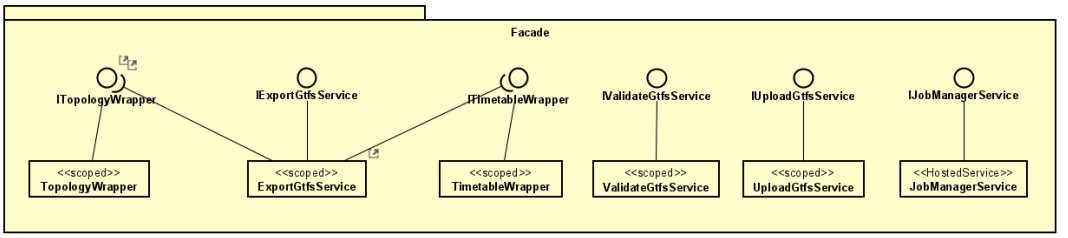

Figura 4.7: Dependencias internas del microservicio

Hay varios aspectos importantes que se pueden observar en la figura 4.8 en la que se muestran los elementos fundamentales que van a hacer posible el proceso de exportación:

- Los controladores solo conocen los métodos de la fachada.
- Hay varios **wrappers**, que encapsulan todas la funcionalidad de pedir datos a otros microservicios.
- El servicio más importante, que es el de exportación, hace principalmente uso de estos wrappers y de los mappers. De esta manera el proceso de creación de los *feeds* explicado de manera general consiste en obtener las entidades de la Suite que se correspondan con entidades Gtfs y pasarlas por un mapeo para luego escribirlas.
- Se puede observar que hay otros dos servicios, el que hace la validación que como se puede ver en el diseño detallado recibe un feed y produce una validación que añade al feed recibido. El otro es el encargado de programar los jobs que se van a ejecutar en segundo plano.
- En la figura 4.9 se puede ver cuáles son las Dtos que se van a obtener de cada uno de los otros microservicios. A partir de esas Dtos, se hará un procesamiento en caso de ser necesario y luego se mapearán a las entidades Gtfs correspondientes.
- El servicio encargado de tratar con los jobs esta marcado con el estereotipo **HostedService**[26]. Eso quiere decir que ejecuta una tarea (para generar los jobs que sean necesarios) en segundo plano que comienza en el momento de arranque del microservicio, y que si no ha finalizado antes, lo hará cuando el microservicio se detenga.

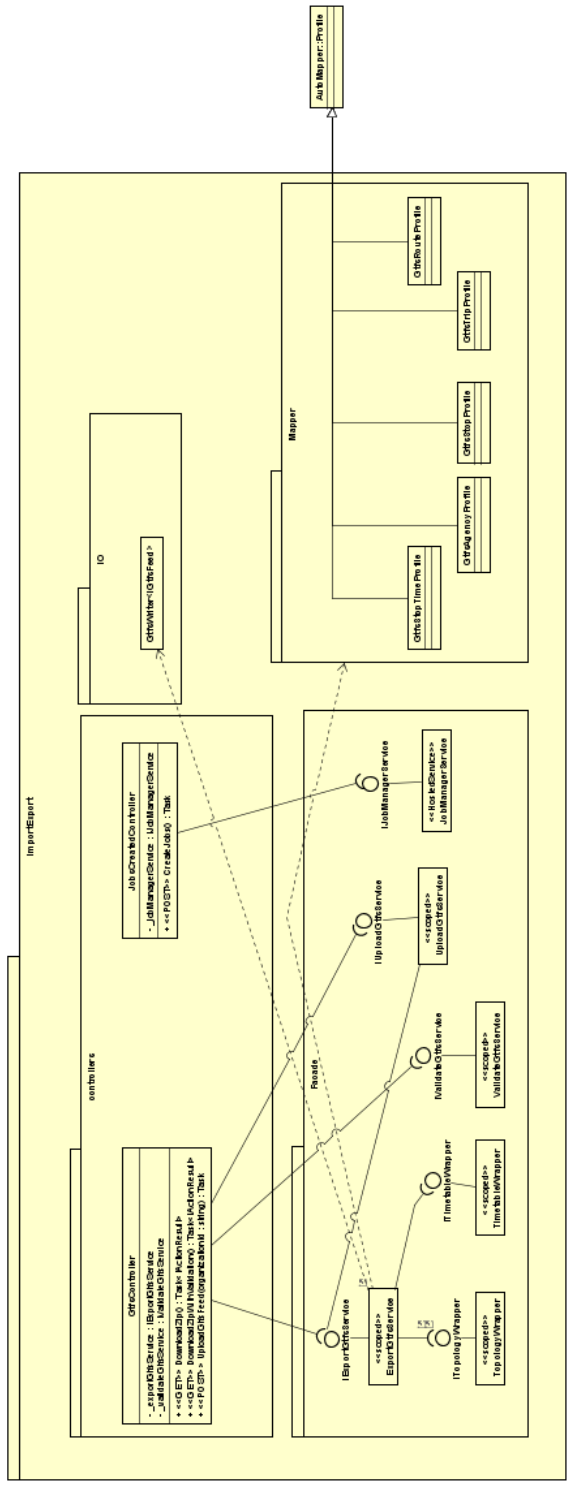

Figura 4.8: Arquitectura del microservicio explicada de manera superficial

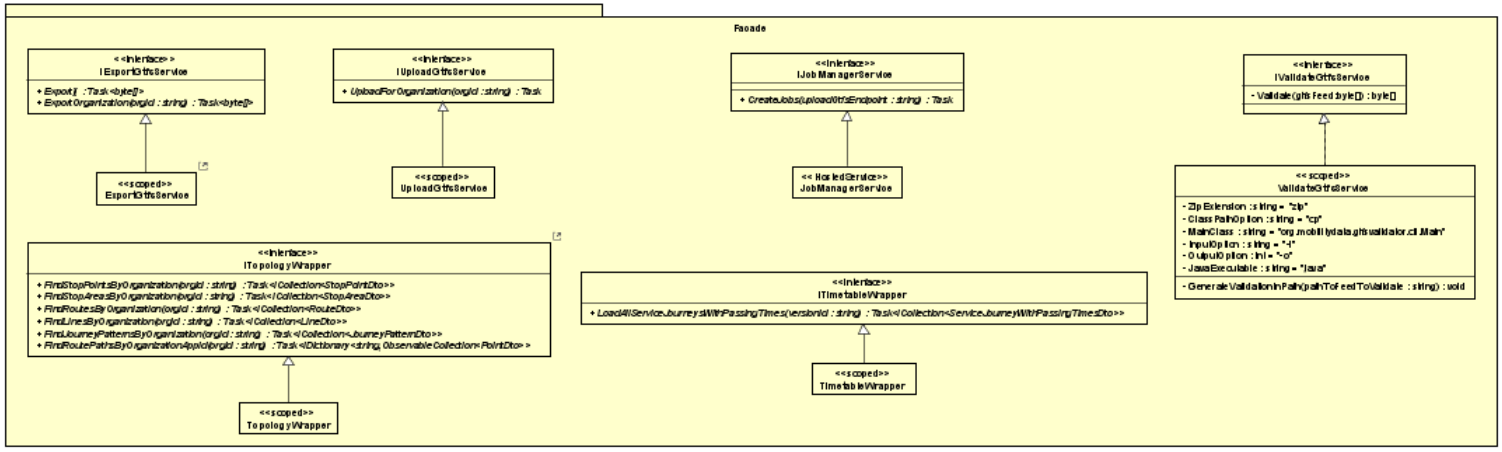

Figura 4.9: Diseño detallado con los métodos de las fachadas

Capítulo 5

Pruebas

5.1. Testing con Xunit

Para cada clase implementada se desarrollan los test unitarios correspondientes que prueban sus métodos. Los test unitarios permiten aislar el método que se está probando de la demás funcionalidad que se use que no pertenezca a la misma clase. De esta manera, al simular el comportamiento de las clases utilizadas aseguramos que si los resultados obtenidos son incorrectos es por la lógica de la clase que estamos probando, y no por la de otras clases.

En el caso de nuestro proyecto se ha utilizado la librería XUnit[45] para hacer tests unitarios. Podría haberse aplicado la filosofía TDD pero no se ha hecho simplemente porque no es la manera en la que trabaja el equipo de desarrollo de la Suite.

Aparte de realizar test unitarios, también se han realizado test integrados. Lo que se prueba con estos es que las conexiones entre los componentes es la adecuada, indicando la respuesta que se tiene que dar ante una petición a una determinada dirección web.

5.2. Testing con validadores de Gtfs

Como se verá en la sección 6 en la que se comenta el seguimiento del proyecto, durante los sprints 7 y 8 ha sido necesario realizar pruebas con diferentes validadores Gtfs.

En este apartado se presentan los resultados de validar con estas diferentes herramientas *feeds* generados con información de tránsito de Salamanca y Reus:

- **Salamanca**

Validador	Resultado esperado	Resultado obtenido
Feedvalidator[31]	Ningún error, feed válido.	Error de tipo de ruta inválido.
Gtfsvtor[29]	Ningún error, feed válido.	Ningún error, feed válido.
gtfs-validator[28]	Ningún error, feed válido.	Fichero de shapes. Se repiten números de secuencia para la misma shape.

Tabla 5.1: Tabla resultados obtenidos validación feed Salamanca

- **Reus**

Validador	Resultado esperado	Resultado obtenido
Feedvalidator[31]	Ningún error, feed válido.	Error de tipo de ruta inválido.
Gtfsvtor[29]	Ningún error, feed válido.	Ningún error, feed válido.
gtfs-validator[28]	Ningún error, feed válido.	Fichero de shapes. Se repiten números de secuencia para la misma shape.

Tabla 5.2: Tabla resultados obtenidos validación feed Reus

El error que proporciona el primer validador es incorrecto respecto al estándar actual, el tipo de ruta usado en los *feeds* es válido. Por lo tanto ese error es incorrecto. Aparte de eso, este primer validador parece bastante desactualizado del estándar actual. El tercer validador proporciona un error real y que indica que hay un error a la hora de generar ese fichero (además este validador es el usado por Google, indicador de que posiblemente es el más fiable de los tres).

Tras haber detectado el error que marca el tercer validador, se ha corregido la manera en la que se generaban los "shapes" (tarea 11.1 del sprint 6.7). Tras hacer este cambio, la validación de los ficheros realizada con el tercer validador no detecta ningún error.

5.3. Pruebas sobre endpoints

Para las pruebas sobre los endpoints debemos de tener en cuenta que no todos los usuarios han de ser capaces de acceder a la exportación de datos. En la siguiente tabla se definen una serie de usuarios y lo que han de ser capaces de hacer.

Usuario/rol	Permisos	Descripción	Código
Usuario sin rol	Ningún permiso	No debería de ser capaz de hacer nada relacionado con la exportación.	0
Usuario sin rol	Permisos de exportación	Debería de ser capaz de exportar ficheros de forma manual.	1
Rol administrador	Tiene todos los permisos	Debería de ser capaz de trabajar tanto con exportación manual como automática	2

Tabla 5.3: Roles y códigos asociados para pruebas con endpoints

Endpoint	Rol	Resultado esperado	Resultado obtenido entorno local	Resultado obtenido entorno despliegue
/gtfs/download	0	Error 403, not authorized	Error 403, not authorized	Error 403, not authorized
/gtfs/download	1, 2	Código 200, Cuerpo de respuesta es un array de bytes que reprensentan feed.	Código 200, Cuerpo de respuesta es un array de bytes que reprensentan feed.	Código 200, Cuerpo de respuesta es un array de bytes que representan feed.
/gtfs/downloadvalidate	0	Error 403, not authorized	Error 403, not authorized	Error 403, not authorized
/gtfs/downloadvalidate	1, 2	Código 200, Cuerpo de respuesta es un array de bytes que reprensentan feed.	Código 200, Cuerpo de respuesta es un array de bytes que reprensentan feed.	Código 200, Cuerpo de respuesta es un array de bytes que representan feed.
/gtfs/upload/organizationId	0	Error 403, not authorized	Error 403, not authorized	Error 403, not authorized.
/gtfs/upload/organizationId	1, 2	Código 202, feed subido a repositorio configurado.	Código 202, feed subido a repositorio configuradod	Código 202, feed subido a repositorio configurado.
/jobscreated	0, 1	Error 403, not authorized	Error 403, not authorized.	Error 403, not authorized.
/jobscreated	2	Código 202. Jobs creados en DB.	Código 202. Jobs creados en DB.	Código 202. Jobs creados en DB.

Tabla 5.4: Tabla resultados obtenidos de llamadas a endpoints

Capítulo 6

Seguimiento del proyecto

Tal y como dicta el proceso de trabajo SCRUM, el seguimiento del proyecto se realizará cada sprint. Al comienzo de cada sprint, se definirá lo que es conocido como *"Sprint Backlog"*, que contendrá todas las tareas que se pretenderán realizar a lo largo de ese Sprint. A cada tarea se le asignará un número de *"Story Points"* teniendo en cuenta la opinión de todos los miembros del equipo.

En el caso de nuestro proyecto el número de *"Story Points"* a asignar ha de ser uno de los de esta lista: *0.5, 1, 2, 3, 5*. Cuando a una de las tareas se la asignen el máximo de *"Story Points"* posibles eso no quiere decir de manera rigurosa que la tarea lleve 40 horas realizarla, si no que realmente no estamos muy seguros de cuanto puede costar llevarla a cabo, pero de seguro es bastante costosa. A la hora de listar las tareas, la mayoría estarán relacionadas con alguna historia de usuario, en ese caso se asociarán con esta.

Cuando el sprint finalice, a cada tarea se le asignará su estado; ya sea completado, en desarrollo o pendiente de comenzar, de manera que en el siguiente sprint, si su estado no es completado, se tenga en cuenta antes de asignar las nuevas tareas.

6.1. Sprint 1

En el primer sprint, lo principal que va a hacer el desarrollador será formarse en las tecnologías que va a usar y aprender cómo funciona el microservicio en el que se va a implementar toda la funcionalidad de exportación de GTFS (ms-import-export).

Se pretende implementar la parte de escritura de los ficheros que forman el *feed* Gtfs dadas las entidades definidas que establece el propio estándar, esto conllevará una parte de investigación ya que para implementar esa parte se pretende usar el código ya desarrollado y testeado de algún repositorio público. Se ha decidido hacer esto en vez de implementar toda esta funcionalidad de nuevo ya que si no, el desarrollador perdería bastante tiempo entre implementación y pruebas. De esta manera haremos que el desarrollo se centre en la parte más importante y costosa de mapeo de entidades de la Suite a entidades Gtfs.

Aparte de la tarea de escritura de los ficheros, se pretende mapear las organizaciones del sistema a las entidades *Agency* de Gtfs. Se estima la duración de este sprint en unas 48 horas debido a las tareas de investigación y al tiempo necesario de aprendizaje que tendrá que destinar el desarrollador.

Nºtarea	Historia de usuario	Descripción de la tarea	Story points	Estado
T-01		Integrar un GtfsWriter, investigar cual es el adecuado, comprobar que funciona correctamente y añadirlo junto con unas pruebas al microservicio	2	Completada
T-02	**HU-001**	Se deben mapear las organizaciones de la ITS Suite al modelo Agency de GTFS. Se debe actualizar la documentación de Confluence explicando el mapeo.	2	En desarrollo
T-03	**HU-002**	Generar GTFS conjunto o por departamento en función de parámetro de configuración.	2	Completada

Tabla 6.1: Tabla planificación sprint 1

6.1.1. Retrospectiva de sprint

Este sprint se esperaba dedicar unas 48 horas a la parte de desarrollo, el tiempo dedicado ha sido de 48 horas como se esperaba, sin embargo no se ha podido completar la tarea correspondiente a la parte de exportar las organizaciones de la ITS Suite debido a que faltaba

añadir un campo obligatorio (URL de la organización) en la entidad a partir de la cual se generan las agencias de los feed Gtfs.

Tras valorar varios repositorios que implementen un *GtfsWriter*, se ha decidido usar el repositorio GTFS[30]. No todas las clases ni paquetes del repositorio son necesarios, los incluidos estarán indicados en la sección de diseño (4) de este documento. Se trata de un repositorio testeado y que ya ha sido usado por otros repositorios. De cualquier manera tras integrarlo en nuestro propio microservicio se han hecho pruebas unitarias y todo funciona como se esperaba.

El desarrollo realizado en este sprint ha sido menos de lo esperado debido a que se ha tenido que lidiar con el uso de nuevas herramientas y lenguaje de programación. Esto es algo que ya se esperaba (Riesgo 1), el aprendizaje suele ser algo que ocurre de manera exponencial. Se confía en que en los siguientes sprints el desarrollo realizado se acerque más al esperado.

A continuación se muestra lo que serían los diagramas de dominio realizados durante la fase de análisis para dar una perspectiva conceptual a los lectores de que es lo que se trata en cada microservicio. El diagrama de dominio ya está dividido por las clases que estarían en cada microservicio.

Es importante recordar que la funcionalidad correspondiente a este proyecto se corresponde a funcionalidad del microservicio de exportación, de ahí que las clases que se ven en el diagrama del microdominio de importación y exportación sean las correspondientes con el estándar Gtfs. Aunque no se haya incluido por evitar una relación de una entidad hacia todas, la gran mayoría de las entidades presentes en el modelo de dominio estarán relacionadas con la organización a la que pertenecen.

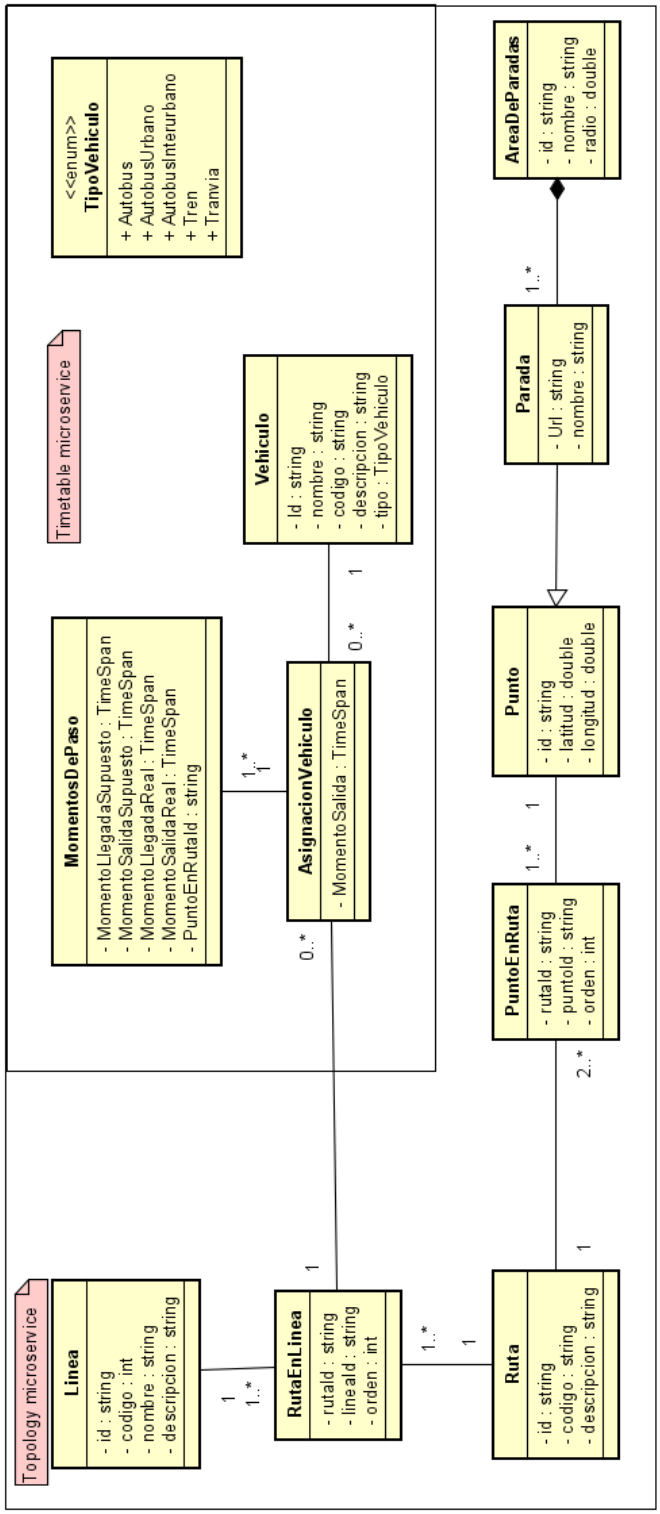

Figura 6.1: Diagrama de dominio en análisis de microservicios topología y horarios

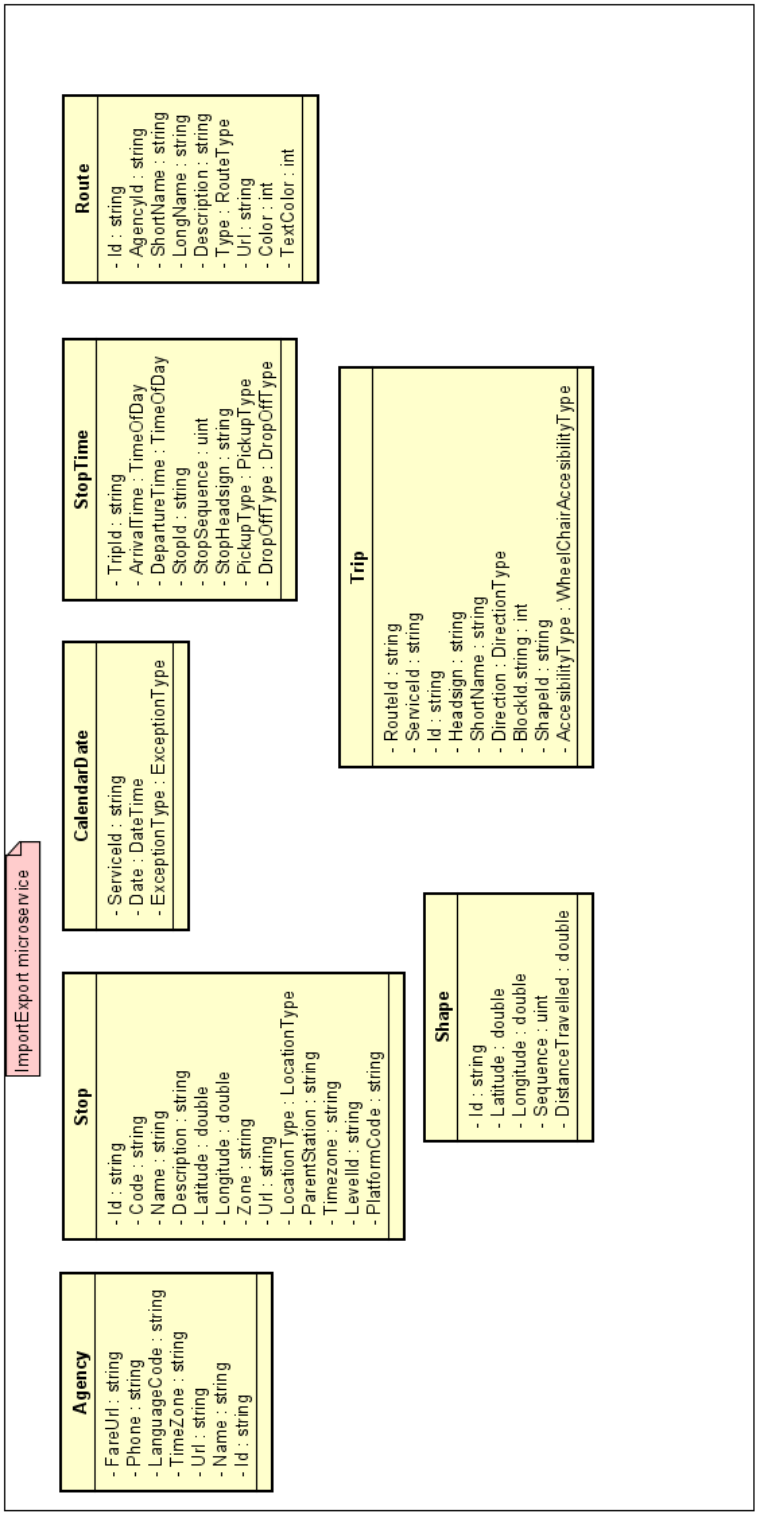

Figura 6.2: Diagrama de dominio en análisis de microservicio de importación y exportación de datos

6.2. Sprint 2

Nº Tarea	Historia de usuario	Descripción de la tarea	Story points	Estado
T-02	**HU-001**	Se deben mapear las organizaciones de la ITS Suite al modelo Agency de GTFS. Se debe actualizar la documentación de Confluence explicando el mapeo.	1	Completada
T-04	**HU-003**	Se deben mapear los StopPoints y StopAreas de la ITS Suite al modelo Stops de GTFS. Se debe actualizar la documentación de Confluence explicando el mapeo.	1	Completada
T-05	**HU-004**	Se deben mapear las Lines y Routes de la ITS Suite al modelo Route de GTFS. Se debe actualizar la documentación de Confluence explicando el mapeo.	2	En desarrollo

Tabla 6.2: Tabla planificación sprint 2

6.2.1. Retrospectiva de sprint

Este sprint se esperaba dedicar unas 32 horas a las tareas. El tiempo dedicado ha sido de unas 40 horas. Al igual que pasó en el anterior sprint, el hecho de que el desarrollador haya tenido que invertir parte de su tiempo en aprender cosas nuevas ha hecho que el tiempo dedicado a realizar las tareas haya sido más del esperado.

Al igual que se comentó en el sprint anterior, el aprendizaje es algo que ocurre de manera exponencial, y se espera que en las próximas semanas el desarrollo avance más rápidamente.

6.3. Sprint 3

En este sprint, se dedicarán horas a la implementación de nuevas funcionalidades y a la investigación de como se deberían de implementar tareas futuras viendo como funcionan los microservicios que se encargan de la gestión de los horarios.

Se tratarán tareas correspondientes a varias historias de usuario, principalmente las correspondientes a la HU-004, la cual no se pudo completar el sprint pasado por falta de tiempo, esta tarea no debería de necesitar mucho trabajo, ya que lo único que falta de implementar para completarla son algunos test de fachada que comprueben que el contenido de los ficheros generados a partir de las entidades *routes* sea el esperado. Que el contenido sea el esperado ya se ha probado con pruebas con Postman, pero el desarrollo de estos test sigue siendo necesario por si en el futuro se produce un cambio en la implementación y queremos asegurarnos de que el resultado final sigue siendo el que ha de ser.

Además de esta historia de usuario, se implementará toda la funcionalidad correspondiente a la HU-005 (eso incluirá sus correspondientes pruebas) y se solucionará un bug encontrado este sprint sobre la HU-003. Si se completan todas estas tareas en la duración del sprint, mientras se espera a la corrección de estas, se comenzará la implementación de la funcionalidad correspondiente a las HU 6 y 8.

Nºtarea	Historia de usuario	Descripción de la tarea	Story points	Estado
T-05	**HU-004**	Se deben mapear las Lines y Routes de la ITS Suite al modelo Route de GTFS. Se debe actualizar la documentación de Confluence explicando el mapeo.	2	Completada
T-06	**HU-005**	Se deben mapear los RouteLinks y Links de la ITS Suite al modelo Shapes de GTFS. Se debe actualizar la documentación de Confluence explicando el mapeo.	1	Completada
T-07	**HU-003**	Cuando no se generan StopAreas, no incluir el ParentStation	1	Completada

Tabla 6.3: Tabla planificación sprint 3

6.3.1. Retrospectiva de sprint

En este sprint, las estimaciones de tiempo han estado un poco por debajo del coste real. Las tareas se han realizado antes de lo esperado puesto que algunas de ellas ya estaban prácticamente finalizadas del sprint anterior. Tal y como se indicó al principio del sprint, al finalizar estas tareas se ha comenzado a investigar como realizar la funcionalidad correspondiente a las historias de usuario **HU-006 y HU-008**, pero esto no se ha tenido en cuenta como tiempo invertido ya que todavía no se ha comenzado su desarrollo.

En total se han dedicado unas 30 horas, cuando se esperaba dedicar unas 32.

6.4. Sprint 4

En este sprint se va a intentar realizar toda la implementación correspondiente a generar las entidades CalendarDates y Trips (HU-006). Puede extrañar que solo se haya asignado esta tarea a realizar para todo el sprint, pero es justificado ya que su complejidad incrementa bastante respecto a la de las tareas hechas hasta ahora.

Los horarios (los momentos en los que un vehículo llega o sale de un punto en concreto) de la ITS Suite están versionados, me explico, de la manera en la que está diseñado el sistema, una versión de horarios tiene pares de fechas de inicio-fin que indican los días en los que esta activa esa versión, y los horarios que tiene esa versión, son los que se aplican para cada día que está activa.

Por ejemplo, si se quisiera tener unos horarios entre semana, y otro los fines de semana (algo bastante común), deberíamos de crear una versión de horarios que tenga los horarios que hay entre semana y indicar que esa versión tiene como pares de fechas inicio-fin todos los lunes y viernes de cada semana, y otra que tenga los horarios que hay los fines de semana y indicar que esa versión tiene como pares de fecha de inicio-fin todos los sábados y domingos de cada semana.

Al igual que los horarios están versionados, también lo están algunos elementos de la topología. El hecho de que estas entidades de la Suite estén versionadas dificulta mucho el mapeo de estas entidades que representan parte de los horarios en el estándar Gtfs.

Para simplificar un poco esta tarea se ha decidido que de momento solo se generen los viajes del sistema suponiendo que todos usan la misma versión de topología, pero diferentes versiones de horarios. La complejidad añadida de tener que tratar con diferentes versiones de topología se tratará en *sprints* futuros o se dejará como algo a realizar como objetivo futuro (cambiar rutas o paradas sobre la topología de una organización es algo que pasa raramente).

Este sprint va a ser un poco distinto, ya que coincide con la semana santa (el día 14 y 15 de abril no son laborables), y por lo tanto habrá menos tiempo para desarrollar.

NºTarea	Historia de usuario	Descripción de la tarea	Story points	Estado
T-08	**HU-006**,**HU-008**	Se deben mapear los horarios de la ITS Suite al modelo de Trips, y CalendarDates de GTFS. Se importarán datos de un mes a futuro, y se deberán tener en cuenta las diferentes planificaciones de horarios que existan. Aunque en esta tarea sólo se considerarán aquellas que vayan contra la topología actual. Se debe actualizar la documentación de Confluence explicando el mapeo.	5	En proceso

Tabla 6.4: Tabla planificación sprint 4

6.4.1. Retrospectiva de sprint

Este sprint se esperaba dedicar "unas 40 horas" a las tareas asignadas (tal y como se explico al principio del apartado de seguimiento, el número máximo de "Story Points" que se pueden asignar a una tarea es de 5, y cuando se asigna ese número a una tarea quiere decir que no se sabe de manera certera cuanto va a costar realizarla).

Como ya se había visto en la planificación del sprint, la complejidad de la tarea asignada es bastante grande, y no ha dado tiempo a finalizarla a pesar de que se le han dedicado unas 48 horas.

El desarrollo de test integrados en los que se trabaja con versiones no es algo que se haya hecho mucho en la Suite, y eso hace que el desarrollador haya tenido problemas con esta última parte. De cualquier manera, no solo hay conclusiones negativas, el desarrollador ha aprendido a trabajar con entidades versionadas y se espera que en el siguiente sprint sea capaz de terminar la tarea asignada.

6.5. Sprint 5

En este sprint la única tarea a realizar que se corresponde a este proyecto es la pendiente del sprint anterior.

La tabla de planificación es por tanto la misma que la del sprint anterior reduciendo el número de "Story Points" asignados.

NºTarea	Historia de usuario	Descripción de la tarea	Story points	Estado
T-08	**HU-006**,**HU-008**	Se deben mapear los horarios de la ITS Suite al modelo de Trips, y CalendarDates de GTFS. Se importarán datos de un mes a futuro, y se deberán tener en cuenta las diferentes planificaciones de horarios que existan. Aunque en esta tarea sólo se considerarán aquellas que vayan contra la topología actual. Se debe actualizar la documentación de Confluence explicando el mapeo.	3	En proceso

Tabla 6.5: Tabla planificación sprint 5

6.5.1. Retrospectiva de sprint

Este sprint se esperaba dedicar 24 horas a la tarea pendiente. Al final del sprint se han acabado dedicando 28 horas a la finalización de la tarea pendiente consiguiendo completarla con éxito.

6.6. Sprint 6

Este sprint tiene como objetivo principal terminar de mapear toda las entidades correspondiente mapear las entidades necesarias de la Suite a los Gtfs StopTimes. Aparte de ello, el desarrollador ha observado que se puede obtener una mejora considerable en el tiempo de ejecución de la exportación si se cambia la manera en la que se exportan los StopPoints de la ITS Suite. Antes para cada StopPoint que se iba a mapear a un Stop de Gtfs hacía falta hacer una petición al microservicio de Topology, y eso retrasaba bastante el tiempo

de respuesta (el número de StopPoints puede ser muy grande). Lo que el desarrollador ha propuesto ha sido añadir un endpoint nuevo en el microservicio de Topology que devuelva todos los StopPoints que pertenecen una organización dado el identificador de esta. Se lo ha comentado al resto de desarrolladores de la Suite en la planificación del sprint y se ha creado una nueva tarea para hacer esto.

Este sprint tendrá en día laborable menos, ya que el Día del Trabajador se celebrará el día 2 de Mayo. De cualquier manera, esto se ha tenido en cuenta a la hora de asignar las tareas, así que no debería de ser un problema a la hora de finalizar las tareas.

Nºtarea	Historia de usuario	Descripción de la tarea	Story points	Estado
T-09	**Mejora HU-003**	Se debe crear un nuevo endpoint en el microservicio de Topología para obtener los StopPoints por organización y usar ese nuevo endpoint en el microservicio ImportExport en lugar del anterior que cargaba cada StopPoint por su identificador.	1	Completada
T-10	**HU-007**	Se deben mapear los horarios de la ITS Suite al modelo de StopTimes de GTFS. Se debe actualizar la documentación de Confluence explicando el mapeo.	2	En proceso

Tabla 6.6: Tabla planificación sprint 5

6.6.1. Retrospectiva de sprint

Este sprint se esperaba dedicar unas 40 horas al desarrollo. En total se ha dedicado unas 45 horas para completar el desarrollo. La última tarea se ha marcado como "En proceso" a pesar de que el desarrollo haya sido completado y probado debido a que este sprint los revisores no han tenido el tiempo necesario como para corregir por completo la funcionalidad implementada y darle el aprobado necesario para "mergearla" a la rama principal. Esta tarea no se añadirá por tanto a la planificación del siguiente sprint.

6.7. Sprint 7

Este sprint tiene como objetivo principal la validación de los *feeds* generados. En el sprint anterior se finalizaron las tareas correspondientes a la generación de los ficheros de carácter obligatorio en un feed Gtfs, por tanto en este se pretende empezar a realizar pruebas con los *feeds* generados con distintas herramientas.

NºTarea	Historia de usuario	Descripción de la tarea	Story points	Estado
T-11	**HU-010**	Generar feed Gtfs de Reus, y probar que sea correcto. Corregir los problemas que surjan.	3	Completada
T-11.1	**HU-010**	Cuando se construyen los shapes de una ruta, cada uno de los puntos que describen el shape deben pertenecer a la ruta en su conjunto, y no a cada uno de los links que la forman.	2	Completada

Tabla 6.7: Tabla planificación sprint 7

6.7.1. Retrospectiva de sprint

La tabla de planificación de este sprint es un poco diferente a las demás. La segunda fila ha sido añadida a lo largo del sprint cuando el desarrollador se ha dado cuenta de los errores cometidos.

Este sprint se tenía pensado dedicar unas 32 horas. Al final se ha dedicado sobre unas 48 horas. Esto se debe al coste de la implementación de la solución a los fallos, ya que en primera instancia se desconocía la existencia de un servicio en el microservicio de topología que ya tuviera la funcionalidad de asociar a una ruta todos los puntos que la componen de manera ordenada hecha, y el desarrollador comenzó a implementarla en vez de reusarla.

En este punto, es importante recordar algo, los errores encontrados, son errores de planteamiento, y no de implementación, es decir, lo que se había implementado funcionaba correctamente (en caso contrario fallarían los test que tiene que pasar cada nueva funcionalidad), el fallo estaba en que la forma en la que se pensaba que tenía que generar las *"shapes"* de los Gtfs era incorrecta.

Durante este sprint, como consecuencia de tener que validar los *feeds* generados, se ha tenido que investigar sobre que herramientas de validación hay disponibles. Para la validación de los *feeds* de Reus, se ha acabado usando las siguientes: FourSquareITP[14], Gtfsvtor[29] y FeedValidator[31].

6.8. Sprint 8

El objetivo principal de este sprint es analizar las herramientas utilizadas en las pruebas del sprint anterior para validar los *feeds* Gtfs, elegir la que se considere mejor y más completa, e implementar la funcionalidad que nos permite usarlas en nuestro microservicio de exportación. La principal razón de que se implemente una validación para los *feeds* Gtfs es (aparte de asegurar que los *feeds* están bien formados, obviamente) evitar pérdidas de tiempo en corregir errores que nos trasladaría Google tras hacer su propia validación de los ficheros (por experiencias previas podemos asegurar que Google puede tardar hasta dos días en validar los *feeds*).

NºTarea	Historia de usuario	Descripción de la tarea	Story points	Estado
T-12	**HU-010**	Evaluar herramienta de validación de GTFS a utilizar. Probar herramientas disponibles y generar un excel valorando: calidad de validación, requisitos, versión online, versión offline, posibilidad de inclusión en imagen docker de ImportExport. Generar un informe con cada herramienta. Finalmente, elegir el validador que se considere más correcto.	1	Completada
T-13	**HU-010**	Incluir herramienta de validación para los Gtfs generados.	3	Completada
T-13.1	**HU-010**	Implementación de servicio que use la herramienta de validación.	2	Completada
T-13.2	**HU-010**	Inclusión de herramienta y sus dependencias en imagen Docker	0.5	Completada

Tabla 6.8: Tabla planificación sprint 8

6.8.1. Retrospectiva de sprint

Al igual que en el anterior sprint, y como se puede observar en la tabla 6.9, la tarea **T-13** se ha dividido en dos subtareas, así se hace más simple la explicación de lo que hay que hacer. En este sprint ha sido necesario dedicar unas 48 horas, a pesar de que la estimación inicial era de 32.

El exceso de tiempo dedicado se debe en su mayoría a tener que probar varios validadores y tener que lidiar con herramientas que en algunos casos están muy desactualizadas, cuyo proceso de instalación puede ser tedioso y que apenas tienen documentación. Los resultados del análisis de los validadores existentes se pueden ver en la imagen B.1. Tras un pequeño debate sobre cuál es el validador que más nos conviene usar, se escogió usar el **gtfs-validator**[28]. Para el desarrollo de la tarea 17.1 hay que tener en cuenta que al final el validador elegido consiste en un ejecutable que se usa desde línea de comandos. Usando la clase Process[22] podemos iniciar un proceso que se encarge de usar esta herramienta. Para poder arrancar el ejecutable con la clase Process necesitaremos su localización.

El único problema que queda entonces es asegurarse de que la localización de los ejecutables sea siempre la misma (no es lo mismo un despliegue en local, que con contenedores Docker), de ahí el por qué de la subtarea 17.2. Para hacer esto partimos de la imagen base que se indica en el repositorio del validador elegido[27] y copiamos los ejecutables de esta imagen en el directorio de trabajo que tendrá el contenedor Docker.

Sería algo como esto:

```
FROM ghcr.io/mobilitydata/gtfs-validator:latest AS gtfsvalidator;
FROM mcr.microsoft.com/dotnet/core/sdk:3.1-buster;
WORKDIR /app;
COPY . .;
COPY --from=gtfsvalidator *.jar .;
ENTRYPOINT ["dotnet","run"];
```

Repito que no son exactamente estas líneas las que se usarían, esto solo es una versión simple del dockerfile (los dockerfile se crean de manera automática por medio de scripts,y hay que tener en cuenta más cosas), las únicas líneas que se conservarían identicamente son la primera y la quinta. Esto solo sirve para dar una visión general al lector de como podemos usar partes de imágenes externas y añadirlas a nuestra imagen docker.

En la primera línea nos traemos la imagen base del validador elegido y en la quinta copiamos todos los jars de esta imagen a nuestro directorio de trabajo establecido en la tercera línea. De esta manera los ejecutables del validador van a estar siempre en */app* y resolvemos nuestro problema.

Llegados a este punto, el desarrollador se ha enfrentado a la decisión de si finalizar el proyecto a defender aquí y dejar la automatización de la exportación como trabajo futuro o continuar con la segunda parte de automatizar el proceso de exportación. Se ha decidido seguir con la implementación y se ha añadido un sprint más (**Sprint 9**) aunque no estuviera planificado inicialmente.

6.9. Sprint 9

Tras el desarrollo de los sprints anteriores, tenemos la capacidad de generar *feeds* Gtfs con la información de la Suite, y somos capaces de pasarles una validación. Esta parte sería la correspondiente a lo primero que se menciona en el primer párrafo de la sección 1.3 en la que se presentan los objetivos y se habla de que las organizaciones deberían de ser capaces manualmente de generar *feeds* Gtfs.

El objetivo de este sprint, sería la segunda parte de lo que se habla en esa sección, automatizar la subida de los *feeds* a Google Transit. Para ello será necesario realizar dos tareas.

Nºtarea	Historia de usuario	Descripción de la tarea	Story points	Estado
T-14	**HU-011**	Crear endpoints y funcionalidad que permitan subir un feed Gtfs dado a un repositorio de Nexus. El repositorio ha de ser configurable por proyecto.	5	En proceso
T-15	**HU-011**	Programación diaria de generación de GTFS. Configurar un job para que se invoque diariamente en hora configurable al endpoint de import-export que genera el fichero Gtfs en Nexus.	1	En proceso

Tabla 6.9: Tabla planificación sprint 9

6.9.1. Retrospectiva de sprint

Este sprint se esperaba dedicar unas 48 horas a las tareas programadas. A la hora de valorar el tiempo que iban a llevar las tareas no se ha tenido en cuenta todo el desarrollo necesario y los cambios en la funcionalidad presente que iba a suponer. En el caso de la primera tarea hay que tener en cuenta que lo correcto en la funcionalidad a implementar es que permita subir cualquier tipo de fichero a un repositorio que siga el formato "Raw"de nexus (y no que solo sirva para subir nuestros *feeds* Gtfs). Así, se crea funcionalidad no

solo útil para el desarrollo de esta tarea, si no para otras personas que puedan querer subir ficheros en el futuro.

Tener que implementar todo esto ha provocado cambios que no se habían tenido en cuenta y cambios en funcionalidad ya implementada. Si a todo esto le añadimos que hasta ahora no se había hecho desarrollo en el microservicio que tiene toda la funcionalidad de trabajar con repositorios de Nexus, el tiempo a dedicar en la primera tarea aumenta. Respecto a la segunda tarea, aunque la mayor parte está implementada, no puede ser terminada al depender de la primera (los jobs a configurar en la segunda tarea acaban llamando a un endpoint que sube los *feeds* Gtfs a Nexus).

Al final del sprint, el tiempo dedicado a las tareas ha sido de unas 60 horas.

6.10. Semana añadida de manera extraordinaria

Para completar la implementación, ha sido necesario añadir unos días (desde el martes 14 de junio hasta el viernes 17). En estos días el único objetivo era completar las tareas que habían quedado en proceso en el final del sprint anterior. El tiempo dedicado durante estos 4 días ha sido de unas 32 horas. La estimación inicial también era de 32 horas (4 días de trabajo a 8 horas por día).

Con la finalización de esas dos tareas pendientes, se ha completado la segunda parte de la funcionalidad principal de la que hablábamos en un principio, y habríamos automatizado la generación de los *feeds* Gtfs y su subida a una URL de la que puedan ser extraídos por las aplicaciones necesarias como por ejemplo Google Transit.

6.11. Conclusiones del seguimiento

A lo largo del proyecto ha sido necesario en muchos casos dedicar más tiempo del estimado a las tareas. Sin embargo, ha sido posible completar la parte del objetivo principal del proyecto (la exportación manual) dentro del plazo estimado en un principio. Para la parte más opcional de automatizar la exportación ha sido necesario añadir un sprint y medio más, y aunque haya sido necesario extender los tiempos de realización del proyecto, se considera que en este caso merece la pena invertir un poco más de tiempo a cambio de una funcionalidad mucho más completa.

Al finalizar el proyecto estos son los tiempos estimados y empleados en cada uno de los sprints:

Sprint	Fecha de inicio	Fecha de fin	Tiempo estimado	Tiempo dedicado
1	08/02/2022	21/02/2022	48	48
2	22/02/2022	07/03/2022	32	40
3	08/03/2022	21/03/2022	32	30
4	22/03/2022	04/04/2022	40	48
5	05/04/2022	18/04/2022	24	28
6	19/04/2022	02/05/2022	40	45
7	03/05/2022	16/05/2022	32	48
8	17/05/2022	30/05/2022	32	48
9	31/05/2022	14/06/2022	48	60
Días extra	14/06/2022	17/06/2022	32	32
TOTAL			360	383

Tabla 6.10: Tiempo estimado y dedicado para cada sprint

6.11.1. Gestión de los riesgos

Respecto de los riesgos presentados en la sección 2.3, los que se han cumplido han sido el riesgo 2.3 que era prácticamente inevitable y el riesgo 2.4.

En el caso del primero se había establecido un plan de mitigación que se empezó a aplicar desde la primera semana y ha durado hasta el final (el desarrollador se ha formado durante todo el desarrollo del proyecto). Podemos decir que en el momento de finalización del proyecto se ha reducido el impacto de este riesgo (en vez de ser medio, pasa a ser bajo) y la probabilidad se mantiene como alta (seguro que todavía faltan muchas cosas por aprender). De acuerdo con lo visto en la asignatura de "Planificación y gestión de proyectos", si se decide seguir con el proyecto, lo conveniente es mantener el plan de mitigación y ofrecer planes de aprendizaje al desarrollador.

En el caso del segundo también se había establecido un plan de mitigación que consistía en comenzar antes de lo necesario el proyecto, disponiendo de esta manera de tiempo para añadir un sprint más. En principio este sprint extra se iba a usar en caso de ser necesario para

completar la parte de exportación manual, pero como por suerte se ha conseguido finalizar esa parte en los 8 sprints previstos, se ha usado para automatizar la exportación. En el momento de finalizar el proyecto, el impacto del riesgo a pasado a ser bajo (ya se ha finalizado el proyecto) y lo mismo pasa con la probabilidad; no sería por tanto necesario mantener el plan de mitigación.

El resto de riesgos vistos en el apartado 2.3 no se han cumplido. En caso de continuar con el proyecto, será necesario monitorizarlos cada semana.

6.11.2. Costes reales

En la sección 2.5 se realizó el análisis de costes y se calculó el presupuesto inicial (6811,2 €).

Concepto	Coste
383 horas laborables	7246,36 €

Tabla 6.11: Costes reales del proyecto

En el momento de finalizar el proyecto se han gastado 435,12 €(un 6,3 %) más de lo planeado en un principio. En términos porcentuales, la desviación por tanto es pequeña y entra dentro de lo común en proyectos de este tipo. Además, es importante recordar que en el cálculo del presupuesto inicial no estaba incluida la parte de automatización, mientras que si lo está incluida en el cálculo del presupuesto final.

Capítulo 7

Conceptos sobre el despliegue

7.1. Introducción

En este proyecto no se ha realizado un despliegue de forma manual como tal porque en el sistema de la suite se realiza de manera automática. Lo que se pretende explicar en esta sección es un concepto muy importante a tener en cuenta al desplegar microservicios en un sistema basado en una arquitectura cloud.

7.2. Service Discovery Pattern

Cuando una aplicación quiere usar una API suele servirse de una configuración local para saber donde están alojados sus servicios (llamamos servicio a una instancia que se está exponiendo de una API). Si más aplicaciones dependen de la misma API, cada una tendrá su propia configuración para determinar donde están los servicios de esa API que necesita.

En una aplicación basada en microservicios que se ejecuta en la nube, que las aplicaciones sepan donde están alojados sus servicios es mucho más complicado, porque las IPs se asignan de manera dinámica y las localizaciones y número de servicios están cambiando constantemente.

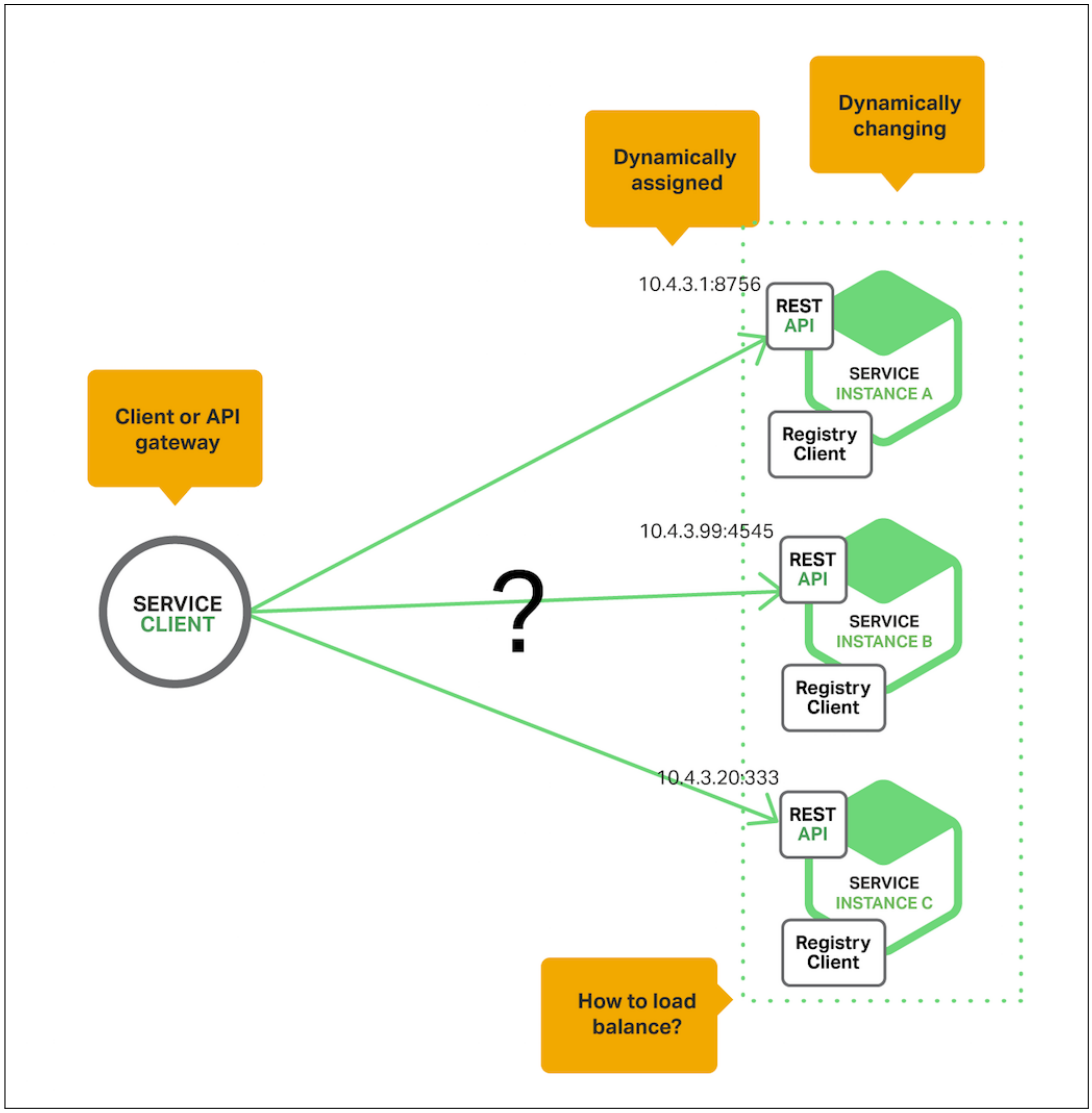

Figura 7.1: Localización de servicios en soluciones Cloud. Imagen extraída de [35]

El patrón **Service Discovery** permite resolver este problema, haciendo que sean los mismos servicios cuando se inician los que se registren en un "registro de servicios", y que cuando la aplicación quiera hacer uso de un servicio lo haga a través del "registro de servicios". Además del registro inicial, los servicios han de mandar señales conocidas como "heartbeat" al "registro de servicios" para que este sepa que siguen disponibles. Cuando la instancia de un servicio termina, el propio servicio notifica al "registro de servicios" para que lo borre. Lo que se llama "registro de servicios" suele implementarse como una base de datos que almacena los identificadores de los servicios disponibles junto con su dirección.

Dependiendo de donde se encuentre el "balanceador de carga", hay dos variantes para este patrón:

7.2.1. Descubrimiento del lado del cliente

En esta variante, el cliente es el responsable de determinar las localizaciones de los servicios y de balancear la carga entre ellos. Para ello la implementación local del balanceador se comunicará con el registro de servicios y luego desde la propia aplicación se elegirá el servicio a utilizar.

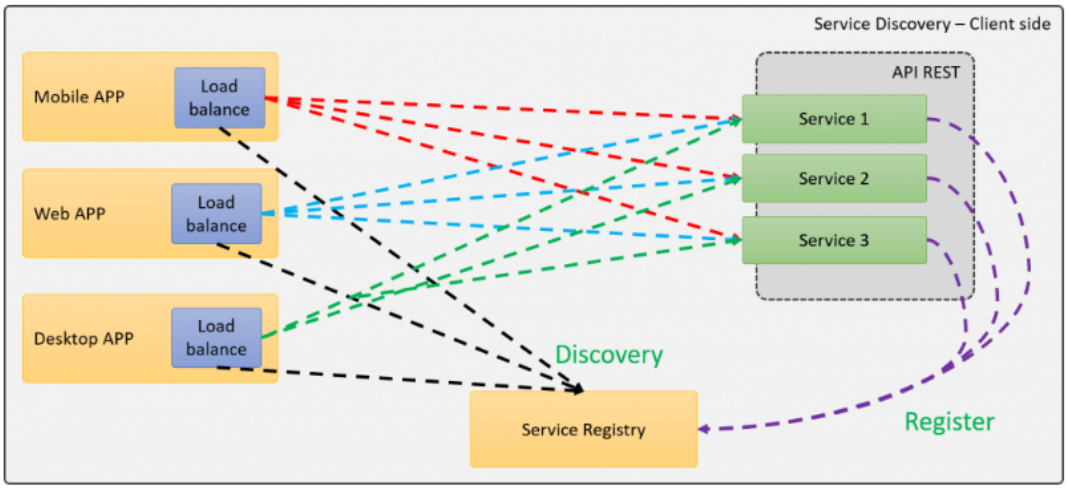

Figura 7.2: Patrón Service Discovery. Variante descubrimiento del lado del cliente. Imagen extraída de [46]

Tal y como se ha explicado, los servicios se registran en el servicio de registros (flecha morada que sale de cada servicio),y los balanceadores de carga de cada cliente se comunican con el registro de servicios para obtener el servicio a utilizar (flecha negra que sale de cada cliente) y luego envían sus peticiones al servicio elegido (flechas rojas, azules y verdes que salen de cada cliente).

La principal ventaja de utilizar el descubrimiento del lado del cliente es que se evita tener un único punto de fallo (si falla el balanceador de carga del cliente móvil, el de la aplicación de escritorio seguirá funcionando, son independientes). Las principales desventajas son que hay que implementar un balanceador de carga por cliente y que los clientes son los que se tienen que tener la lógica de descubrimiento de servicios.

7.2.2. Descubrimiento del lado del servidor

En esta segunda variante, el balanceador de carga es un componente independiente de los clientes. Los clientes hacen peticiones al balanceador de carga y este es el que se comunica con el registro de servicios para usar uno u otro.

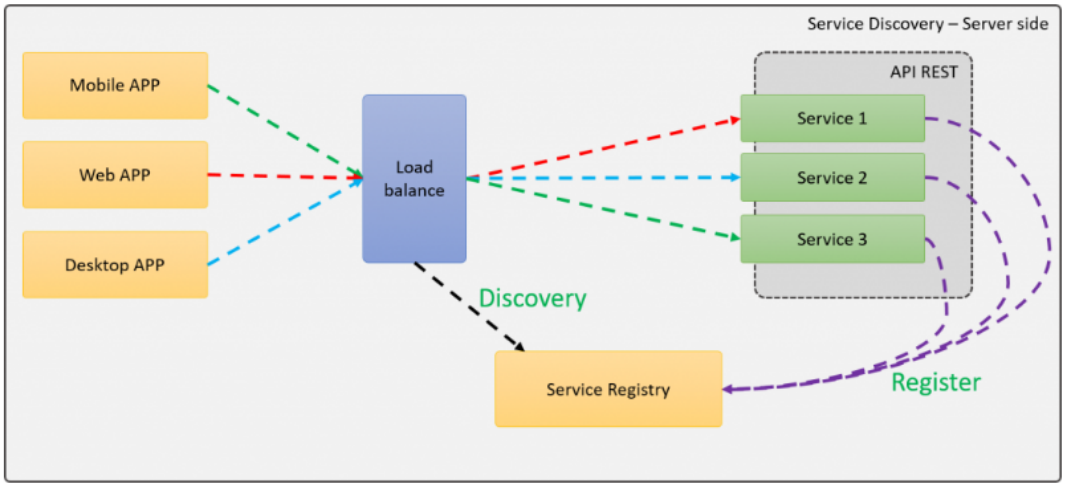

Figura 7.3: Patrón Service Discovery. Variante descubrimiento del lado del servidor. Imagen extraída de [46]

La parte buena de usar esta versión es que se desacoplan las funcionalidades y no se implementa la misma funcionalidad en cada cliente. La mala es que si no se configura de manera adecuada puede ser un punto de fallo único que deje sin servicios a todos los clientes.

Para utilizar el patrón de descubrimiento de servicios, hay varias herramientas que podemos usar. A continuación se mencionan algunas de las más populares.

En el caso del registro de servicios:

- Netflix Eureka[32]. Es el usado por Netflix.
- Consul[7]. Provee una API que permite a los clientes registrar y configurar servicios.

En el caso de los balanceadores de carga:

- Ribbon[33]. Parte del stack tecnológico de Netflix OSS. Permite realizar el balanceo de cargar por medio de Eureka.
- Nginx[34]. Es un servidor web de código abierto que, pero también es usado como proxy inverso, cache de HTTP, y balanceador de carga.

Capítulo 8

Conclusiones

Principales conclusiones del proyecto:

- Se ha desarrollado una herramienta que permite a las empresas de transporte que están suscritas a los servicios de la Suite exportar sus datos de transporte estáticos en forma de *feeds* que siguen la especificación Gtfs.
- Se ha automatizado el proceso de exportación, de manera que desde el momento en el que las organizaciones depositan los datos necesarios(topología y horarios) en el sistema, el sistema es capaz de generar los *feeds* automáticamente sin necesidad de tener más interacción con el usuario.
- Se ha integrado la posibilidad de subir los *feeds* generados a repositorios de Nexus.
- Se ha integrado la validación de *feeds* Gtfs estáticos en el sistema de la Suite. De esta manera podemos asegurar que nuestros *feeds* son correctos desde el punto de vista de la especificación Gtfs y que los datos del sistema son coherentes.
- Se han realizado pruebas de la funcionalidad desarrollada tanto en entorno local como en entornos reales.
- Por el hecho de tener que realizar nuevas funcionalidades en varios microservicios existentes de la Suite, se ha adquirido cierto nivel de conocimiento sobre este sistema que en algunos casos tiene bastante complejidad.
- Se han obtenido conocimientos sobre el estándar Gtfs en su variante estática, C# y .NET Core.
- Se ha visto la importancia de pasos previos como el diseño y la planificación antes del comienzo de un proyecto con cierto nivel de complejidad en el que se trabajarán tanto con estándares como tecnologías desconocidas.

8.1. Objetivos no cumplidos

- HU-009, ser capaz de exportar las tarifas del sistema.

Hay dos razones importantes por las que no se ha implementado esta historia de usuario: la primera es su coste (se estima que sea de unos 5 story points, es decir, unas 40 horas), y la segunda y más importante es que en los ficheros que componen los *feeds*, las tarifas no son algo que haya que incluir de manera obligatoria para que sean válidos (al contrario que los ficheros generados que se corresponden con otras historias de usuario).

Por estas dos razones, la HU-009 es la que tenía asignada una menor prioridad. Hubiera sido posible haber realizado esta historia de usuario a cambio de la automatización de la generación de los *feeds* o añadiendo más tiempo de trabajo al planificado en un principio.

8.2. Líneas de trabajo futuras

Hay varias cosas en las que todavía se puede seguir trabajando:

- Implementar la HU-009, para aquellas organizaciones que tienen información tarifaria, generar los ficheros correspondientes en sus *feeds*.
- **Completar los *feeds* con diferentes versiones de entidades de topología.** Como ya se comentó durante el seguimiento de los sprints, los *feeds* generados actualmente utilizan la versión de topología actual del sistema para generar los *feeds*. En los casos más normales (prácticamente siempre) la información topológica de una organización no cambia, pero en el caso de que pasase, a nuestros *feeds* faltaría añadirles esa información de próximas versiones. Una solución rápida a este problema es la que esta implementada actualmente, en la que cada vez que se activa una versión de topología, se generan unos nuevos *feeds* automáticamente y se suben al repositorio correspondiente, de esta manera la información permanece actualizada. Sin embargo, el estándar Gtfs recomienda que sus ficheros sean válidos por al menos 7 días, y si se suceden varias versiones de topología distintas en días consecutivos, este *workaround* no cumpliría con esa recomendación.
- **Creación de feeds Gtfs Realtime.** Aunque no forme parte de este proyecto, y más bien sería uno nuevo, la especificación Gtfs no solo es usada en los *feeds* estáticos, si no que también tiene es usada en una extensión de estos primeros llamada **Gtfs Realtime**[18] que permite que las empresas de transporte público brinden a los usuarios información en tiempo real sobre las interrupciones en sus servicios (estaciones cerradas, líneas que no funcionan, retrasos importantes, etc.), la ubicación de sus vehículos y los horarios de llegada esperados. Quizás en el futuro y tras haber realizado un estudio de mercado para ver si a los clientes les puede interesar esta nueva funcionalidad, podría utilizarse la información en tiempo real de la Suite para generar estos nuevos *feeds*.

Bibliografía

[1] Astah. Herramienta para hacer diagramas. https://astah.net/. (marzo 2022).

[2] Atlassian. Atlassian como empresa de software. https://www.atlassian.com/es. (febrero 2022).

[3] Atlassian. Bitbucket como servicio de alojamiento de código y control de versiones. https://www.atlassian.com/es/software/bitbucket. (febrero 2022).

[4] Atlassian. Confluence herramienta para documentación. https://www.atlassian.com/es/software/confluence. (febrero 2022).

[5] Atlassian. Software jira para seguimiento de proyectos. https://www.atlassian.com/es/software/jira. (febrero 2022).

[6] BikeShareMap. Vista de bikesharemap. https://bikesharemap.com/#/3/-60/25/. (febrero 2022).

[7] Consul. Consul para service discovery. https://www.consul.io/. (junio 2022).

[8] Diagrams. Diagrams herramienta para hacer diagramas online. https://www.diagrams.net/. (marzo 2022).

[9] Docker. Docker como herramienta usada para contenerizar aplicaciones. https://www.docker.com/. (mayo 2022).

[10] Docker. Dockerfile como archivo usado para crear imagenes docker. https://docs.docker.com/engine/reference/builder/. (mayo 2022).

[11] Dotnet. Dotnet como herramienta multiplataforma. https://dotnet.microsoft.com/en-us/. (abril 2022).

[12] Editores de Zoho. Que es kanban. https://www.zoho.com/sprints/kanban.html?src=leftpanel. (febrero 2022).

[13] Editores de Zoho. Que es un *Scrum Board*. https://www.zoho.com/sprints/what-is-a-scrum-board.html. (febrero 2022).

[14] FourSquareITP. Herramienta reflect para validar feeds gtfs. https://reflect.foursquareitp.com/. (mayo 2022).

[15] Google. Archivo calendar.txt del estándar gtfs. `https://developers.google.com/transit/gtfs/reference?hl=es#calendartxt`. (febrero 2022).

[16] Google. Conceptos básicos sobre google transit planner. `https://support.google.com/transitpartners/answer/1111471`. (febrero 2022).

[17] Google. Especificación gtfs, punto especialmente importante para leer por los lectores. `https://developers.google.com/transit/gtfs/reference?hl=es`. (febrero 2022).

[18] Google. Gtfs realtime como extensión de los feeds gtfs estáticos. `https://developers.google.com/transit/gtfs-realtime`. (junio 2022).

[19] Google. Portland se convierte en primera ciudad en google transit trip planner. `https://googleblog.blogspot.com/2006/09/happy-trails-with-google-transit.html`. (febrero 2022).

[20] Google Maps. Vista de maps. `https://www.google.com/maps/?hl=es`. (febrero 2022).

[21] Kubernetes. Kubernetes para orquestación de contenedores. `https://kubernetes.io/es/`. (mayo 2022).

[22] Microsoft. Clase process. `https://docs.microsoft.com/es-es/dotnet/api/system.diagnostics.process?view=net-6.0`. (mayo 2022).

[23] Microsoft. Entityframework como framework orm para .net. `https://docs.microsoft.com/es-es/ef/core/get-started/overview/first-app?tabs=netcore-cli`. (febrero 2022).

[24] Microsoft. Linq para hacer consultas en formato sql. `https://docs.microsoft.com/es-es/dotnet/csharp/programming-guide/concepts/linq/introduction-to-linq-queries`. (marzo 2022).

[25] Microsoft. Sobre el framework dotnet. `https://docs.microsoft.com/es-es/dotnet/core/introduction`. (febrero 2022).

[26] Microsoft. Tareas en segundo plano en .net. `https://docs.microsoft.com/es-es/aspnet/core/fundamentals/host/hosted-services?view=aspnetcore-6.0&tabs=visual-studioe`. (junio 2022).

[27] MobilityData developers. Imagen base de gtfs-validator. `https://github.com/MobilityData/gtfs-validator/pkgs/container/gtfs-validator`. (mayo 2022).

[28] Múltiples contribuidores de GitHub. Dotnet como herramienta multiplataforma. `https://github.com/MobilityData/gtfs-validator`. (mayo 2022).

[29] Múltiples contribuidores de GitHub. Validador de feeds gtfs. `https://github.com/mecatran/gtfsvtor`. (mayo 2022).

[30] Múltiples usuarios de GitHub. Gtfswriter usado para escribir las entidades gtfs en ficheros. `https://github.com/itinero/GTFS`. (marzo 2022).

[31] Múltiples usuarios de GitHub. Validador de feeds gtfs. `https://github.com/google/transitfeed/wiki/FeedValidator`. (mayo 2022).

[32] Netflix developers. Netflix eureka. `https://github.com/Netflix/eureka`. (junio 2022).

[33] Netflix developers. Netflix ribbon. `https://github.com/Netflix/ribbon`. (junio 2022).

[34] Nginx. Nginx en service discovery. `https://www.nginx.com/`. (junio 2022).

[35] Nginx developers. Sobre el service discovery pattern. `https://www.nginx.com/blog/service-discovery-in-a-microservices-architecture/`. (junio 2022).

[36] Overleaf. Overleaf como editor online de latex usado. `https://es.overleaf.com/`. (febrero 2022).

[37] Postman. Postman herramienta usada para probar apis. `https://www.postman.com/`. (febrero 2022).

[38] Sourcetree. Sourcetree ofrece interfaz gráfica para git. `https://www.sourcetreeapp.com/`. (febrero 2022).

[39] Trabajadores anónimos con el puesto de desarrollador back-end en España. Sueldos de desarrollador back-end en españa. `https://www.glassdoor.es/Sueldos/desarrollador-back-end-sueldo-SRCH_KO0,22.htm`. (abril 2022).

[40] Visual Studio. Página de descargas de visualstudio. `https://visualstudio.microsoft.com/es/downloads/`. (febrero 2022).

[41] VisualStudio. Visual studio ide. `https://visualstudio.microsoft.com/es/`. (febrero 2022).

[42] Wikipedia. Latex como herramienta usada en redacción de la memoria. `https://es.wikipedia.org/wiki/LaTeX`. (febrero 2022).

[43] Wikipedia. Sobre el lenguaje de programación c#. `https://es.wikipedia.org/wiki/C_Sharp`. (febrero 2022).

[44] Wikipedia. Tex como sistema de tipografía. `https://es.wikipedia.org/wiki/TeX`. (febrero 2022).

[45] Xunit. Xunit como herramienta de testing usada en test unitarios. `https://xunit.net/`. (febrero 2022).

[46] Óscar Blancarte. Explicación del service discovery pattern. `https://www.oscarblancarteblog.com/2018/09/24/service-discovery-pattern-para-microservicios/`. (junio 2022).

Apéndice A

Resumen de enlaces adicionales

Aunque todos los incluidos en la bibliografía tengan importancia, los enlaces de más interés en este Trabajo Fin de Grado son:

- Lo más importante de todo, especificación de feeds Gtfs estáticos`https://developers.google.com/transit/gtfs/reference?hl=es`[17]
- URL de primer validador utilizado: `https://github.com/google/transitfeed/wiki/FeedValidator`[31].
- URL segundo validador utilizado: `https://github.com/mecatran/gtfsvtor`[29].
- URL tercer validador utilizado: `https://github.com/MobilityData/gtfs-validator`[28].

Apéndice B

Imágenes auxiliares

B.1. Imágenes auxiliares

Herramienta	Calidad(sobre 10)	Requisitos	Online	Offline	Documentación	Facilidad de uso	Posibilidad de instalación en docker	Dependencias	Pasos	Desventajas	Observaciones extra
https://github.com/mecatran/gtfsvtor	7	NO	SI	SI	6	9	SI	Java 8+, gradle, git	gradle build(la primera vez tras bajar repo), gradle run --args (Path al feed en un zip)	No valida ciertas cosas como por ejemplo que no se repita la misma clave de shape con distintas secuencias	
https://github.com/MobilityData/gtfs-validator.git	9	NO	No	SI	10(https://github.com/MobilityData/gtfs-validator/blob/master/RULES.md)	9	SI	Java 11	java -jar (nombre del .jar) -i (path al feed en un zip) -o (nombre del directorio para output)		Permite validar Gtfs dada una url, output en html y json
https://github.com/google/transitfeed/wiki/FeedValidatorIde.es(-v2)	5	NO	No	SI	6	10	SI	Python 2.7(segun el repo ha sido testeado en python 2.4 y 2.5 solo)	python (Path a feedvalidator.py) -i (Path a feed gtfs en zip) -o (nombre fichero salida).html	No da cómo válidos los tipos extendidos	

Figura B.1: Tabla de análisis de validadores Gtfs