Universidad de Valladolid

ESCUELA DE INGENIERÍA INFORMÁTICA

GRADO EN INGENIERÍA INFORMÁTICA
Mención en Ingeniería del Software

Implementación y explotación de un sistema de *APIs* para el consumo de *KPIs*

Alumno: Juan Medina-Bocos Lorenzo

Tutor Académico: Benjamín Sahelices

Tutor Empresa: Rafael González

A mi padre, a mi madre, a mis hermanas y a mis amigos

Agradecimientos

Gracias a mi familia por ayudarme siempre con todo y estar ahí para mí.

Gracias a todos mis amigos, en especial a los *Lame9*, que me han acompañado durante estos años de carrera y han conseguido convertirla en una experiencia única y bonita.

A Diego, que desde secundaria me ha acompañado en mi día a día aconsejándome y acompañándome. Gracias por estar siempre ahí.

A Rebe, por ser una de las personas más importantes para mí, estar ahí siempre que lo he necesitado y apoyarme durante tantos años.

Gracias a Benjamín por gestionar este proyecto y al resto de profesores que me han enseñado el mundo de la informática durante la carrera.

Por último, gracias a Rafa y a Jairo por haberme ayudado en todo momento con las dudas del proyecto.

Resumen

El objetivo de este proyecto es diseñar e implementar un sistema de *APIs* basado en la arquitectura *API-Led* para la explotación de *KPIs*.

Este sistema está formado por diversas *APIs* divididas en tres capas que se encargarán de extraer los datos de todas las *APIs* de la empresa en *MuleSoft* y se los entregará al usuario tratados para su posterior explotación en *PowerBI*.

Con estas métricas obtenidas el usuario podrá gestionar de forma sencilla los tiempos de respuesta, el número de llamadas o la tasa de error entre otras cosas de todas las aplicaciones de la empresa.

Este proyecto posibilitará mejorar la arquitectura de *APIs* en la empresa permitiendo tomar mejores decisiones a la hora de gestionar las *APIs*.

Palabras clave: *APIs*, *KPIs*, *MuleSoft*, *PowerBI*, Arquitectura *API-Led*, Métricas.

Abstract

The objective of this project is to design and implement an *API* system based on the *API-Led* architecture for the exploitation of *KPIs*.

This system, consisting of several *APIs* divided into three layers, will extract the data on all the company's *APIs* in *MuleSoft* and deliver them to the user for later use in *PowerBI*.

With these metrics obtained, the user will be able to easily manage response times, the number of calls or the error rate, among other things, of all the company's applications.

This project will allow the improvement of the *API* architecture in the company, leading to better decisions when managing *APIs*.

Keywords: *APIs*, KPIs, *MuleSoft*, *PowerBI*, *API-Led* Architecture, Metrics.

Índice general

Lista de Figuras

Lista de Tablas

Capítulo 1

Introducción

Hoy en día la mayor parte de servicios informáticos se tienen que comunicar con bases de datos o con otros servicios para poder aportar una funcionalidad plena y completa. Una gran cantidad de empresas importantes en el mercado como *Amazon*, plataformas de *streaming* como *Netflix* y redes sociales como *Facebook* permiten a desarrolladores externos visualizar sus productos mediante la creación de *APIs* públicas[3].

Pese a que bastantes empresas ya se han pasado a los sistemas de *APIs* para agilizar los servicios internos o para permitir a terceros acceder a sus contenidos aún existen empresas que están en proceso de actualización para hacer uso de estas aplicaciones en su día a día.

El objetivo de este proyecto es diseñar, implementar, lanzar y gestionar un sistema de *APIs* que se encargue de recoger toda la información posible de los diversos servicios que proporciona el sistema de *MuleSoft* para su futura explotación por parte de la empresa.

Para ello el proyecto se dividirá en dos partes:

1. Desarrollo de un sistema de *APIs* que permita extraer métricas de la plataforma de MuleSoft.
2. Explotación de *KPIs* mediante el uso de la herramienta *PowerBI*.

1.1. Motivación

La motivación y objetivos personales para realizar el trabajo son los siguientes:

- Acercarme más al desarrollo de *APIs*, experimentando con sus herramientas de creación, ya que desde que entré en el mundo laboral en NTT Data es un tema que cada día me está llamando más la atención.

- Aprender acerca de la aproximación que hace *MuleSoft* sobre los sistemas de *APIs* mediante la arquitectura *API-Led*. MuleSoft es una de las principales empresas en el mercado para el desarrollo de APIs,y el esquema que propone me parece bastante interesante debido a las ventajas que proporciona.
- Tener unos conocimientos básicos acerca del funcionamiento de *PowerBI*, una herramienta bastante utilizada para realizar estudios de medidas.
- Apoyar a la empresa realizando un trabajo sobre los conocimientos que he ido adquiriendo durante este tiempo y facilitar una posible incorporación a la misma debido a que estos meses he estado muy a gusto con mis compañeros de trabajo y me gustaría apoyar a la compañía con la realización de este proyecto.

Este proyecto ha sido realizado en convenio con la empresa *NTT Data España*. Para ello, el alumno ha contado con la ayuda de un tutor dentro de la empresa, Rafael González, quien ha ayudado en el desarrollo del proyecto, así como un tutor de la Universidad que ha ayudado con la organización de la memoria.

1.2. Estructura de la memoria

Este documento se estructura de la siguiente forma:

Capítulo 2 Marco conceptual: Introduce los conceptos teóricos necesarios para la comprensión del proyecto así como las tecnologías que han sido usadas durante su desarrollo.

Capítulo 3 Planificación: Explica la metodología seguida para el desarrollo del proyecto y las fases de que se compone, así como un análisis de costes y posibles riesgos.

Capítulo 4 Análisis y diseño: Describe los requisitos, el modelo de dominio, los casos de uso y las decisiones de diseño que se han tomado para el sistema de *APIs*.

Capítulo 5 Implementación y pruebas: Detalla el proceso de implementación del proyecto y las pruebas que se han realizado.

Capítulo 6 Demostrador *PowerBI*: Introduce los diversos gráficos que se han usado para la explotación de los datos extraídos por el sistema.

Capítulo 8 Conclusiones: Describe cómo ha ido la resolución del proyecto.

Bibliografía: Cita las fuentes que se han utilizado para consultar información en el desarrollo del proyecto.

Capítulo 2

Marco conceptual

En este capítulo se tratarán algunos conceptos necesarios para el entendimiento de este proyecto, así como las tecnologías utilizadas. Además, se detallará la metodología seguida durante el trascurso del proyecto y las fases en que se ha dividido.

2.1. Contexto del proyecto

Para entender el proyecto primero es preciso saber cuáles son las metas principales que se deben cumplir al finalizarlo. Estas metas son:

1. Realizar un sistema de *APIs* que permita extraer medidas sobre las aplicaciones de una plataforma.
2. Monitorizar esas medidas mediante *dashboards* para extraer *KPIs* que permitan una mejor gestión y conocimiento sobre dichas aplicaciones.

Se podría decir que el proyecto se cimienta sobre dos conceptos importantes: las *APIs* y los *KPIs*.

2.1.1. *APIs*

Una *API* (del inglés, *Application Programming Interface*) es un conjunto de definiciones y protocolos que permite a diferentes aplicaciones comunicarse entre sí y compartir información y funcionalidades. Las *APIs* permiten la comunicación entre varias aplicaciones sin la necesidad de conocer la implementación de estas[6].

Existen *APIs* que son públicas y están abiertas a cualquier desarrollador para que este pueda trabajar con ellas en un proyecto propio, *APIs* que están abiertas únicamente a colaboradores y *APIs* que son usadas de forma interna por empresas y tienen la función de ayudar al negocio y facilitar la colaboración entre equipos.

Las *APIs* tienen numerosas ventajas frente a las aplicaciones extremo a extremo; algunas de las más importantes son:

- Poder reciclar las *APIs* existentes en nuevas aplicaciones. En caso de que ya tengas un proyecto que haga uso de *APIs*, a la hora de hacer un nuevo proyecto parecido al anterior puedes hacer uso de las *APIs* creadas en el primer proyecto para apoyar el desarrollo del segundo. Esto permite reducir costes de tiempo y recursos en el nuevo proyecto.
- Competitividad en el mercado. Cuando una empresa de una industria publica crea un sistema de *APIs*, el resto de la industria intenta hacer lo mismo ya que los desarrolladores independientes pueden hacer uso de esas *APIs* para sus propias aplicaciones y aumentar los ingresos de aquellas empresas que hayan publicado su sistema de *APIs*.
- Dar uso a datos sin utilidad. Numerosas empresas tienen cantidades enormes de datos que no pueden ser usados de forma útil. Publicar *APIs* con esos datos permite a otros equipos de la empresa, o a usuarios externos (en caso de *APIs* públicas) utilizar esos datos y poder generar beneficios.
- Sencillez de uso. Hace unos años las empresas desarrollaban complicados sistemas para integrar sus aplicaciones. Hoy en día, mediante el uso de *APIs*, una tecnología pensada para su uso en Internet y que la mayor parte de los profesionales de software están familiarizados con esta, una empresa que quiera conectar con el mayor número de clientes tendría que pensárselo dos veces antes de optar por el uso de *APIs*.

2.1.2. *KPIs*

Un *KPI* (del inglés, *Key Performance Indicator*) es un indicador o valor cuantitativo que se pueden medir, comparar y monitorear, con el fin de exponer el desempeño de los procesos y trabajar en las estrategias de un negocio. Se trata de medir los resultados en números para aportar información real a las decisiones, no solo percepciones y conjeturas.[19]

La medición de *KPIs* permite a una empresa:

- Obtener información valiosa y útil.
- Medir determinadas variables y resultados a partir de dicha información.
- Realizar comparativas con la información obtenida.
- Tomar decisiones oportunas.
- Tener una visión global de un sistema.

En este proyecto el objetivo principal de la monitorización de los *KPIs* es obtener una visión global del sistema, conociendo la información de cada grupo de negocio, entorno y aplicación. Conociendo y comparando estos *KPIs* podemos tomar decisiones que afecten al rendimiento del resto del sistema de aplicaciones. Al tener un gran número de aplicaciones, sin esta monitorización los cambios realizados serían tediosos de analizar generando costes que podrían haberse ahorrado.

2.2. Contexto tecnológico

Para el desarrollo del proyecto se han utilizado una gran variedad de tecnologías y herramientas diferentes que procedo a explicar a continuación.

2.2.1. Lenguaje de programación

Para la implementación de las *APIs* se hace uso de *Java* para organizar y dar funcionalidad a los conectores de *Mule* que actúan como pequeños fragmentos de código. Estos conectores, por dentro, hacen uso de *Java*.

Para la implementación de las diversas *APIs* se hace uso de diversos conectores que proporciona el Anypoint Studio, estos conectores están implementados en Java. Sin embargo, para organizar estos conectores se hace uso de lenguaje XML.

2.2.2. Lenguaje de modelado

RAML es un lenguaje de definición de *APIs*. Estos lenguajes permiten crear un documento que especifique correctamente todos los aspectos a definir en una *API*. Así, podemos tener controlado todo en un documento que nos permitirá generar la documentación de la *API*, crear casos de prueba, implementar un *mock* para poder acelerar el desarrollo y definir el esqueleto de nuestra aplicación.[8]

Dentro del proyecto, este lenguaje ha sido usado para el modelado de las *APIs* dentro del *Design Center*, permitiendo hacer pruebas sin necesidad de tener la implementación realizada.

2.2.3. Entornos de desarrollo

Anypoint Design Center es un entorno de desarrollo que consta de las siguientes herramientas:[16]

- *API Designer* permite crear especificaciones de *API* en varios lenguajes de modelado y crear fragmentos de *API* en lenguaje *RAML*. Estos fragmentos pueden ser publicados en el *Anypoint Exchange* para su posterior uso.
- *Flow Designer* permite la creación de aplicaciones *Mule* para integrar sistemas en flujos de trabajo.

En este proyecto se ha usado el *API Designer* para realizar el diseño de las *APIs*.

Anypoint Studio es el *IDE* de *MuleSoft* basado en *Eclipse* para diseñar, implementar y probar aplicaciones de *Mule*.[16]

Algunas de las características de este *IDE* son:

- Ejecución instantánea de su aplicación *Mule* dentro de un *runtime* local.
- Editores visuales para configurar archivos de especificación de *API* y dominios *Mule*.
- Actualización de cambios de una aplicación local en ejecución para probar estos cambios sin tener que relanzar toda la aplicación.
- Integración con *Exchange* para importar plantillas, ejemplos, definiciones y otros recursos de su organización *Anypoint Platform*.
- Marco de pruebas unitarias integrado.
- Soporte integrado para desplegar en *CloudHub*.

En este proyecto se ha usado *Anypoint Studio* y ha sido empleado para la implementación de las *APIs* y la realización de las pruebas en local.

2.2.4. Control y gestión de *APIs*

Anypoint API Manager es un componente del *Anypoint Platform* que permite gestionar, controlar y proteger las *APIs*.[16]

Anypoint Monitoring proporciona visibilidad de las integraciones de las aplicaciones proporcionando información de los flujos y componentes de *Mule* de las aplicaciones.[16]

2.2.5. Control de versiones

SourceTree[17] es una herramienta que nos permite interactuar con los repositorios de *GitHub* de forma más sencilla.

Esta herramienta se ha utilizado debido a que es la que se utiliza en mi equipo de la empresa para llevar a cabo el control de versiones. Como resultado, ya estaba familiarizado con su funcionamiento, lo que ha optimizado el proceso de control de versiones en general.

2.2.6. Análisis de Datos

PowerBI[10] es un sistema predictivo, inteligente y de gran apoyo, capaz de traducir los datos (simples o complejos) en gráficas, paneles o informes.[7]

PowerBI ha sido usado para la explotación de los datos obtenidos por el sistema de *APIs* en forma de diversas gráficas y tablas.

2.2.7. Pruebas

Postman es una plataforma que nos permite construir y usar *APIs* simplificando los pasos del ciclo de vida de estas con el fin de crear mejores *APIs.*[14]

Postman ha sido usado para efectuar las llamadas a las APIs con el fin de probar la aplicación tanto al final del proyecto como durante la implementación de este.

2.2.8. Documentación

Overleaf [13] es una herramienta de publicación y redacción colaborativa en línea que facilita y agiliza el proceso de redacción, edición y publicación de documentos. *Overleaf* nos proporciona un editor *LaTeX* fácil de usar con colaboración en tiempo real y la salida totalmente compilada producida automáticamente en segundo plano a medida que escribe.[21]

Overleaf ha sido usado para la escritura de la memoria.

2.3. Metodología de trabajo

Para este proyecto se ha elegido optar por la metodología modelo de cascada. En esta metodología el proyecto se divide en distintas fases secuenciales[2], donde el equipo puede pasar a la siguiente fase solo cuando se haya completado la anterior, permitiendo cierta superposición entre las fases. Además, este modelo permite volver a una etapa anterior del proyecto en caso de que exista algún fallo en etapas posteriores del proyecto.[23]

Se ha elegido esta metodología ya que el proyecto debe ser realizado de forma secuencial, esto significa que no se puede realizar la implementación sin haber hecho antes un diseño del sistema. Además, el modelo de cascada es una metodología muy sencilla de entender y fácil de aplicar a este proyecto dadas sus características.

Para usar esta metodología el proyecto debe pasar por las siguientes seis fases:

1. **Estudio de Factibilidad**: En esta fase se evalúa si la realización del proyecto será favorable. Dentro de este estudio se realizarán tres estudios de factibilidad distintos:

   a) **Factibilidad técnica**: Intenta descubrir si se poseen las tecnologías necesarias para llevar a cabo el proyecto.

   b) **Factibilidad legal**: Se pretende comprobar si el proyecto incumple alguna normativa estatal, autonómica o municipal.

   c) **Factibilidad de tiempo**: Se trata de analizar si el tiempo que se tiene planificado para llevar a cabo el proyecto es viable.

2. **Requisitos de usuario**: En esta fase se analizan los requisitos funcionales, no funcionales y de información del proyecto sin entrar en detalles técnicos de implantación.

3. **Análisis del sistema**: En esta fase se identifican algunas de las funcionalidades que tendrá la aplicación.

4. **Diseño del sistema**: En esta fase se realiza una descripción de la estructura del sistema que servirá como base para su construcción.

5. **Implementación**: En esta fase se realiza la implementación del sistema sobre la base del diseño establecido en la anterior etapa.

6. **Pruebas**: En esta fase se realiza una batería de pruebas para verificar el correcto funcionamiento del sistema.

7. **Mantenimiento**: Una vez la aplicación está ya desplegada, en esta fase se realizan correcciones de errores, mejoras de rendimiento y mejoras en general del *software*

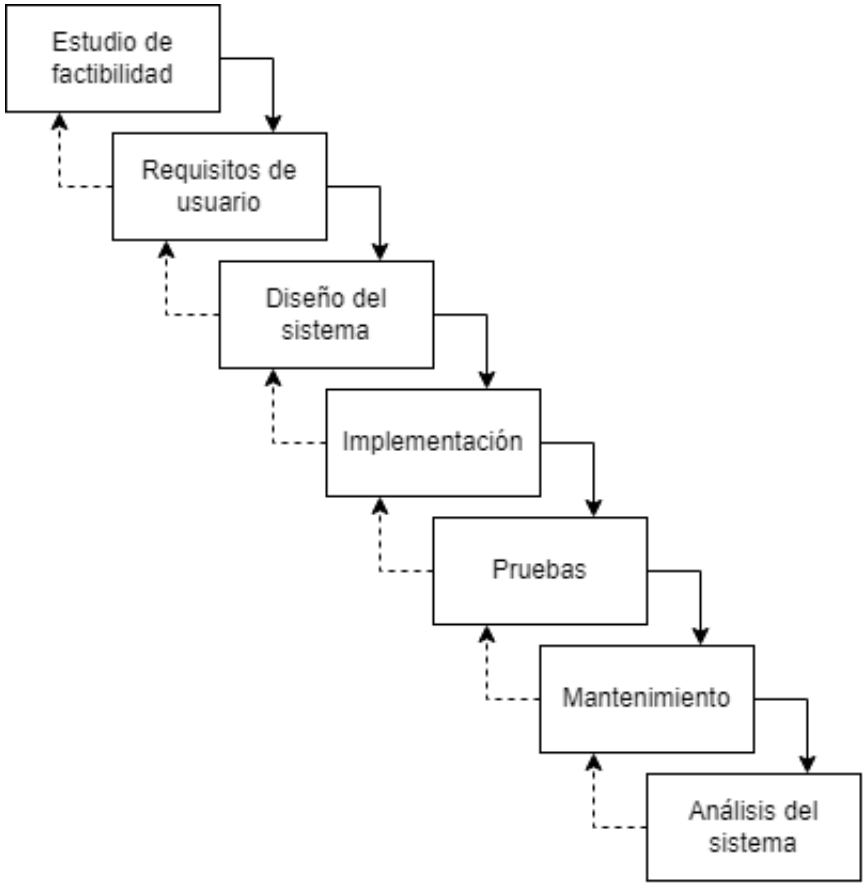

Figura 2.1: Modelo de cascada

Capítulo 3

Planificación

Este capítulo describirá la planificación seguida para el desarrollo del proyecto, haciendo un análisis de los costes y los posibles riesgos que se puedan presentar.

3.1. Costes

En este apartado se tratarán los costes que ha conllevado el desarrollo del proyecto en su totalidad.

El proyecto se inició el 14 de Noviembre de 2022 y se estima el final para el 3 de Marzo de 2023; un total de 16 semanas de trabajo, 109 días. Trabajando de lunes a viernes a media jornada (4 horas), hace un total de 320 horas de trabajo totales de duración del proyecto. Quitando días de fiesta o no laborales se estiman unas 300 horas útiles de trabajo. Esta estimación de tiempo será usada para la aproximación de los costes.

3.1.1. Costes de personal

El proyecto ha sido realizado por una única persona. Asumiendo que su salario es el correspondiente a un ingeniero informático *junior* - 30 € por hora - Multiplicando el salario por las horas estimadas del proyecto se estima un coste por este concepto de 9.000 €.

Puesto	**Coste**
Ingeniero informático *junior*	9000 €
Total	**9000 €**

Tabla 3.1: Costes de personal

3.1.2. Costes de *hardware*

Los costes de *hardware* corresponden al coste del equipo físico usado para el desarrollo del proyecto.

Sin embargo, dado que los equipos usados no han sido comprados ni usados únicamente para este proyecto es preciso estimar el coste inputable a este proyecto. A tal efecto, se ha decidido asignar una vida útil de 4 años para los equipos informáticos. Por tanto, el porcentaje de tiempo que un equipo informático ha sido usado para el proyecto frente a su tiempo de vida útil total es de 7,47 %.

Equipo	**Coste**	**7,47 % del coste**
Ordenador portátil *Dell Latitude* 5420	1.112,11 €	71,06 €
Ratón óptico *Logitech* G203	47,99 €	3,58 €
Monitor *Asus VG248QE*	174,40 €	13,03 €
Total	**1334,50 €**	**87,67 €**

Tabla 3.2: Costes de *hardware*

3.1.3. Costes de *software*

Para este proyecto el único software de pago que ha sido usado es la licencia de MuleSoft que posee NTT DATA. Debido a que se desconoce el precio de esa licencia ya que dicha información es confidencial y no podemos tener un dato exacto de este coste.

Sin embargo, podemos estimar su coste en unos 100000 €, esta estimación ha sido realizada usando como referencia el coste de otras licencias de herramientas para gestión de *APIs*, un ejemplo de estas licencias es el servicio que proporciona Google cuyo coste ronda los 300000 €.

Herramienta	**Coste**
MuleSoft	100000 €
Total	**100000 €**

Tabla 3.3: Costes de software

3.1.4. Otros costes

Estos costes son los relacionados con el entorno de trabajo, por ejemplo el agua, la electricidad, etc.

Para el cálculo de estos costes se tomará en cuenta la duración del proyecto y la época en la que este ha sido desarrollado, es decir, invierno.

Servicio	**Coste**
Agua	30 €
Calefacción	55 €
Electricidad	40 €
Internet	50 €
Total	**175 €**

Tabla 3.4: Otros costes

3.1.5. Total de costes

El coste total del proyecto corresponde a la suma de todos los costes.

Tipo	**Coste**
Costes de personal	3606 €
Costes de *hardware*	87,67 €
Costes de *software*	100000 €
Otros costes	175 €
Total	**103868,67 €**

Tabla 3.5: Total de costes

3.2. Análisis de riesgos

Según el *PM-BOK* un riesgo es: *un evento o condición incierta que, si ocurre, tendrá un efecto positivo o negativo sobre al menos un objetivo del proyecto, llámese tiempo, costo, alcance o calidad*[11]

Mediante el análisis de riesgos se pretende identificar, analizar y diseñar un plan de actuación para los riesgos antes de que ocurran con el fin de prevenir o limitar el impacto de estos riesgos en nuestro proyecto.[18] Para ello, primero debemos de cuantificar el impacto que puede tener un riesgo en un proyecto definiendo dos magnitudes: la probabilidad y el impacto.

- Probabilidad: Se corresponde con la capacidad que tiene el riesgo de ocurrir. Ya que es muy difícil asignar un valor exacto a la probabilidad de que ocurra un riesgo, dividiremos las probabilidades en tres niveles según su rango de probabilidades.

Probabilidad	**Rango**
Alta	Probabilidad de ocurrir mayor del 50 %
Media	Probabilidad de ocurrir entre el 10 y el 49 %
Baja	Probabilidad de ocurrir menor del 10 %

Tabla 3.6: Intervalos de la probabilidad de los riesgos

- Impacto: Se corresponde con las consecuencias que tendría el riesgo en caso de ocurrir. Al igual que antes, se divide el impacto en tres niveles según su nivel de impacto al presupuesto del proyecto.

Impacto	**Rango**
Alto	Impacto al presupuesto mayor del 30 %
Medio	Impacto al presupuesto entre el 10 y el 29 %
Bajo	Impacto al presupuesto menor del 10 %

Tabla 3.7: Intervalos del impacto de los riesgos

3.2.1. Riesgos

Estos son los principales riesgos que han sido identificados y podrían afectar al proyecto.

R001	Fallo en la estimación del proyecto
Descripción	Debido a que la estimación es una aproximación, es posible que no se cumplan las fechas señaladas; si no se sigue la estimación afectará a la duración final del proyecto.
Probabilidad	Media.
Impacto	Medio.
Acciones de mitigación	Hacer una estimación realista y añadir algunos días de margen.

Tabla 3.8: R001 - Fallo en la estimación

R002	Desconocimiento de las tecnologías
Descripción	Debido a que el proyecto se desarrolla con tecnologías nuevas para el alumno, es posible que afecte a la duración final del proyecto.
Probabilidad	Media.
Impacto	Bajo.
Acciones de mitigación	Solicitud y lectura de documentación y material formativo sobre las tecnologías que se van a utilizar.

Tabla 3.9: R002 - Desconocimiento de las tecnologías

R003	Incompatibilidad de versiones
Descripción	Las versiones de las herramientas y tecnologías utilizadas podrían no ser compatibles entre sí o con el equipo en el que se desarrolla el proyecto.
Probabilidad	Baja.
Impacto	Medio.
Acciones de mitigación	Comprobación de las dependencias de versiones y la compatibilidad de las herramientas con el equipo antes de dar inicio al desarrollo.

Tabla 3.10: R003 - Compatibilidad de versiones

R004	Disponibilidad del desarrollador
Descripción	Por motivos personales o de salud el desarrollador puede no encontrarse disponible durante cierto tiempo, lo que afectaría a la duración final del proyecto.
Probabilidad	Baja
Impacto	Medio.
Acciones de mitigación	Hacer una estimación realista y añadir algunos días de margen.

Tabla 3.11: R004 - Disponibilidad del desarrollador

R005	Indisponibilidad de la conexión a internet
Descripción	En caso de fallo de la conexión a internet, la duración de muchas fases del desarrollo se verá comprometida.
Probabilidad	Baja
Impacto	Medio.
Acciones de mitigación	Tener una conexión alternativa como los datos móviles.

Tabla 3.12: R005 - Indisponibilidad de la conexión a internet

R006	Pérdida del desarrollo
Descripción	Pérdida de fragmentos de código o documentación en desarrollo. Esto implicaría tener que repetir el desarrollo de lo que se ha perdido y afectaría a la duración del proyecto.
Probabilidad	Baja
Impacto	Alto.
Acciones de mitigación	Tener un repositorio y una copia de seguridad en la nube de todo lo que se vaya desarrollando.

Tabla 3.13: R006 - Pérdida del desarrollo

Capítulo 4

Análisis y Diseño

En este capítulo se tratarán las fases previas a la implementación del proyecto. Estas fases son la toma de requisitos, el análisis del sistema y el diseño del sistema.

4.1. Requisitos

Se conoce como requisito a una necesidad documentada sobre el contenido, forma o funcionalidad de un producto o servicio estableciendo qué debe hacer el sistema, pero no cómo hacerlo. Existen tres categorías de requisitos[12]:

- Requisitos funcionales: Describen las funciones que debe realizar el proyecto.
- Requisitos no funcionales: Describen las restricciones o condiciones que impone el cliente al proyecto.
- Requisitos de información: Describen la información que debe almacenar y gestionar el proyecto.

Para obtener los requisitos, mi tutor de la empresa y yo nos reunimos para hacer una toma de requisitos. Los requisitos que han sido identificados en esta reunión son:

4.1.1. Requisitos funcionales

Los requisitos funcionales obtenidos en la toma de requisitos han sido agrupados en la Tabla 4.1.

Código	Requisito	Descripción
RF-01	Visualizar *Assets*	El sistema permitirá visualizar toda la información de los recursos mediante *dashboards*.
RF-02	Visualizar *NumberAPIs*	El sistema permitirá visualizar toda la información del número de *APIs* mediante *dashboards*.
RF-03	Visualizar *Policies*	El sistema permitirá visualizar toda la información de las políticas mediante *dashboards*.
RF-04	Visualizar *Contracts*	El sistema permitirá visualizar toda la información de las subscripciones mediante *dashboards*.
RF-05	Visualizar *Manager*	El sistema permitirá visualizar toda la información del *API Manager* mediante *dashboards*.
RF-06	Visualizar *Monitoring*	El sistema permitirá visualizar toda la información del *Runtime Monitoring* mediante *dashboards*.
RF-07	Filtrado entorno	El sistema permitirá filtrar por entorno.
RF-08	Filtrado *API*	El sistema permitirá filtrar por *API*.
RF-09	Filtrado *Business Group*	El sistema permitirá filtrar por *business group*.
RF-10	Fichero *KPIs*	El sistema devolverá un fichero con toda la información de los *KPIs*.

Tabla 4.1: Requisitos funcionales

4.1.2. Requisitos no funcionales

Los requisitos no funcionales obtenidos en la toma de requisitos han sido agrupados en la Tabla 4.2.

Código	Requisito	Descripción
RNF-01	Seguridad	Todas las *APIs* del sistema estarán securizadas mediante *API Key*.
RNF-02	*Internet*	Todas las *APIs* del sistema estarán expuestas a Internet.

RNF-03	*HTTP*	Todas las *APIs* del sistema usarán el protocolo HTTP.
RNF-04	*REST*	Todas las *APIs* del sistema usarán arquitectura *REST*.
RNF-05	Errores	Todas las *APIs* del sistema tendrán los errores capturados.
RNF-06	*API-Led*	El diseño de la arquitectura cumplirá con la filosofía *API-Led*.
RNF-07	Tiempo de respuesta	Las *APIs* de la capa de experiencia no deben tener tiempos de respuesta superiores a los 1 minuto.

Tabla 4.2: Requisitos no funcionales

4.1.3. Requisitos de información

En este sistema no se almacena ninguna información por lo que no existe ningún requisito de este tipo.

4.2. Modelo de dominio

En el modelo de dominio se describen las distintas entidades, sus atributos, papeles y relaciones con el fin de representar un vocabulario y los conceptos clave del proyecto.

- ***APIs*:** Una *API* (del inglés, *Application Programming Interface*) es un conjunto de definiciones y protocolos que permite a diferentes aplicaciones comunicarse entre sí y compartir información y funcionalidades.[24] Las *APIs* permiten la comunicación entre varias aplicaciones sin necesidad de conocer la implementación de estas.[15]
- ***KPIs*:** Un *KPI* (del inglés, *Key Performance Indicator*) es un indicador o valor cuantitativo que se pueden medir, comparar y monitorear, con el fin de exponer el desempeño de los procesos y trabajar en las estrategias de un negocio. Se trata de medir los resultados en números para aportar información real a las decisiones, no sólo percepciones y conjeturas.[19]
- ***Endpoint*:** Un *endpoint* es una pasarela que conecta los procesos del servidor de la aplicación con una interfaz externa. En otras palabras, es la dirección a la que se envían las peticiones.[1] Una *API* puede tener múltiples *endpoints* y cada uno realizaría llamadas diferentes a uno o varios servidores.

- ***REST*:** es un estilo arquitectónico que establece un conjunto de reglas y restricciones para el diseño de sistemas distribuidos y servicios web. Está basado en el protocolo *HTTP* y utiliza los métodos y verbos del protocolo para realizar operaciones en recursos web.

- ***API REST*:** Es una interfaz de programación de aplicaciones (API o API web) diseñada siguiendo los principios y limitaciones de la arquitectura *REST (Representational State Transfer)*. Estas *APIs* proporcionan una forma estructurada y estandarizada de comunicación entre diferentes aplicaciones y servicios a través de la web permitiendo a los desarrolladores enviar solicitudes a través de los métodos *HTTP* como *GET*, *POST*, *PUT* y *DELETE* para interactuar con los recursos de estos servicios.

4.3. Casos de uso

En esta sección se generará el diagrama de casos de uso y se efectuarán en profundidad los casos de uso hallados en el diagrama.

4.3.1. Diagrama de casos de uso

Un diagrama de casos de uso, en el contexto de ingeniería del software, representa a un sistema o subsistema como un conjunto de interacciones que se desarrollarán entre casos de uso y sus actores en respuesta a un evento que inicia un actor principal.[22]

Este diagrama representa de forma esquemática las interacciones entre los actores y el sistema desarrollado para hacer uso de las diferentes funcionalidades implementadas.

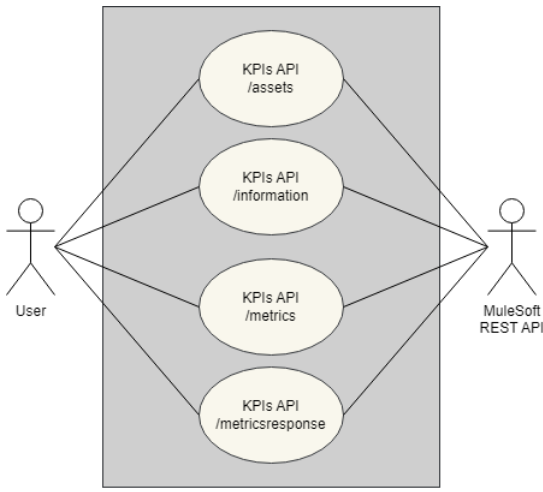

Figura 4.1: Diagrama de casos de uso

4.3.2. Descripción de los casos de uso

A continuación se expresa con mayor detalle la información de cada caso de uso mostrado en la sección anterior.

Este primer caso de uso es la actuación del sistema cuando el usuario desea obtener la información de las *APIs* que se encuentran dentro de la plataforma de *MuleSoft*.

Identificador	CU-01 *API KPis /assets*
Actor principal	Usuario.
Descripción	El sistema devuelve la información de los *Assets*.
Secuencia normal	1. El usuario solicita la información de los *Assets*. 2. El sistema recoge la información del sistema *API REST* de *MuleSoft*. 3. El sistema formatea la información obtenida. 4. El sistema devuelve la información al usuario.
Secuencia alternativa	2.1. Si el sistema no puede recoger la información del sistema *API REST* de *MuleSoft* devuelve el fallo por pantalla y el caso de uso termina.
Postcondiciones	1. La información que se devuelve al usuario está en formato *CSV*.

Tabla 4.3: CU-01. *API KPIs /assets*.

Este segundo caso de uso es la actuación del sistema cuando el usuario desea obtener la información de las políticas y contratos del resto de *APIs* dentro de la plataforma de *MuleSoft*.

Identificador	CU-02 *API KPis /information*
Actor principal	Usuario.
Descripción	El sistema devuelve la información de las políticas y contratos de las *APIs*.
Secuencia normal	1. El usuario solicita la información del *API Information*. 2. El sistema recoge la información del sistema *API REST* de *MuleSoft*. 3. El sistema formatea la información obtenida. 4. El sistema devuelve la información al usuario.
Secuencia alternativa	2.1. Si el sistema no puede recoger la información del sistema *API REST* de *MuleSoft* devuelve el fallo por pantalla y el caso de uso termina.

Postcondiciones	1. La información que se devuelve al usuario está en formato *CSV*.

Tabla 4.4: CU-02. *API KPIs /information*.

Este tercer caso de uso es la actuación del sistema cuando el usuario desea inciar una descarga de la información de las métricas de las *APIs* dentro de la plataforma de *MuleSoft*.

Identificador	CU-03 *API KPis /metrics*
Actor principal	Usuario.
Descripción	El sistema inicia una descarga de información de las métricas.
Secuencia normal	1. El usuario solicita la información del *API Metrics*. 2. El sistema inicia una descarga asíncrona de la información del sistema *API REST* de *MuleSoft*. 3. El sistema devuelve el identificador para acceder a esa información.
Secuencia alternativa	2.1. Si el sistema no inicia la descarga de información del sistema *API REST* de *MuleSoft* devuelve el fallo por pantalla y el caso de uso termina.
Postcondiciones	1. El identificador es único.

Tabla 4.5: CU-03. *API KPIs /metrics*.

Este cuarto caso de uso es la actuación del sistema cuando el usuario desea obtener la descarga generada en el tercer caso de uso.

Identificador	CU-04 *API KPis /metricsresponse*
Actor principal	Usuario.
Descripción	El sistema devuelve la información de la descarga de métricas.
Precondiciones	1. Se ha iniciado el caso de uso CU-03 anteriormente.
Secuencia normal	1. El usuario solicita la información de la descarga de métricas. 2. El sistema recoge la información de la descarga de métricas. 3. El sistema formatea la información obtenida. 4. El sistema devuelve la información al usuario.
Secuencia alternativa	2.1. Si el sistema no ha finalizado la descarga de métricas avisa al usuario y el caso de uso termina.

Postcondiciones	1. La información que se devuelve al usuario está en formato *CSV*.

Tabla 4.6: CU-04. *API KPIs /metricsresponse.*

4.4. Diseño de la arquitectura

El diseño de la arquitectura es una descripción de la estructura del sistema para su posterior implementación. Este diseño nos permite generar un prototipo de como queremos que sea la versión final del proyecto y nos ayuda a modularizar el sistema en unidades más pequeñas que tengan una función similar. Esta etapa de diseño nos sirve para poder empezar el etapa de implementación partiendo de una base solida que nos permita agilizar dicha etapa.

4.4.1. Arquitectura *API-Led*

Para poder entrar en el diseño de la arquitectura primero se debe hablar del enfoque que tiene *MuleSoft* sobre las integraciones punto a punto.

MuleSoft intenta buscar una nueva forma de realizar estas integraciones basado en la generación de *assets* reutilizables en una arquitectura de tres capas que facilita la utilización y la reutilización de las *APIs*. Las capas en las que se divide la arquitectura son[9]:

- **Capa de experiencia:** Dedicada a la presentación de la información reutilizando la lógica implementada en las capas inferiores. Generalmente sirven a las aplicaciones cliente.
- **Capa de procesos:** Procesa los datos obtenidos en la capa de sistema y aplican la lógica correspondiente para cubrir las distintas necesidades de negocio. Es donde se construyen y coordinan los procesos de negocio.
- **Capa de sistema:** Conjunto de *APIs* y conectores que se conectan con los sistemas de orígenes de datos.

Este enfoque de la arquitectura tiene bastantes beneficios en comparación con el la arquitectura convencional. Algunos de estos beneficios son:

- **Escalabilidad**: La arquitectura *API-Led* permite añadir nuevas *APIs* en cualquiera de las capas, sin afectar al resto de *APIs* de esa capa o de otras capas. Esto permite que cualquiera pueda conectarse a una de las *APIs* sin afectar al resto de conexiones.
- **Reducción de la carga de trabajo de *IT***: A la hora de generar nuevos sistemas, la arquitectura *API-Led* permite reutilizar *APIs* de otros sistemas que puedan ser de utilidad para el nuevo, reduciendo los costes de nuevos proyectos.

4.4.2. Aplicación de la arquitectura

Para adaptar la arquitectura *API-Led* a la aplicación se realizaron en la fase de diseño diversas arquitecturas que cumplieran con los requisitos de la aplicación.

Para explicar estos diseños se irá describiendo en cada capa el número de *APIs* que incluye y la función que realizan.

Diseño inicial

Este primer diseño consistía en:

- **Capa de Experiencia (1 API)**: Esta *API* se encargaría de devolver al usuario los datos en formato *CSV* con la información necesaria para trabajar con *PowerBI*. Solo existe una única *API* en esta capa ya que en principio el proyecto solo va a ser usado por un único usuario.
- **Capa de Proceso (3 APIs)**: Estas *APIs* se encargarían de juntar la información obtenida en la capa de sistema.
- **Capa de Sistema (5 APIs)**: Estas *APIs* se encargarían de llamar al sistema *RESTful* de *MuleSoft* para obtener las medidas a devolver al usuario y tratarlas para mandárselas a la capa de proceso.

Este primer diseño fue descartado ya que incumplía ciertas condiciones del diseño. Algunas de estas razones fueron:

- Las *APIs* de la capa de proceso únicamente juntaban información proveniente de la capa de sistema y en ciertos casos algunas *APIs* de proceso solamente servían como intermediario entre la capa de experiencia y la capa de sistema, siendo redundantes en el diseño.
- Las *APIs* de la capa de sistema hacían también parte del trabajo de las capas de proceso ya que hacían tratamiento de los datos que vienen del sistema *RESTful* de *MuleSoft*.

En definitiva, en este diseño la capa de proceso no tenía un uso definido y podría ser prescindible en su totalidad dejando entonces únicamente una arquitectura con dos capas que no cumple con la arquitectura *API-Led*.

Diseño intermedio

Este segundo diseño era algo más sencillo que el anterior ya que pretendía eliminar la implementación de una capa. Este segundo diseño consistía en:

- **Capa de Experiencia (1 API)**: Esta *API* se encargaría de devolver al usuario los datos en formato *CSV* con la información necesaria para trabajar con *PowerBI*. Solo existe una única *API* en esta capa ya que en principio el proyecto solo va a ser usado por un único usuario.
- **Capa de Proceso (5 APIs)**: Estas *APIs* se encargarían de tratar y combinar la información obtenida en la capa de sistema para pasarla a la capa de experiencia.
- **Capa de Sistema (0 APIs)**: Estas *APIs* serían las propias del sistema *RESTful* de *MuleSoft* entrando únicamente en esta capa aquellas llamadas usadas en la capa de proceso para obtener la información.

Las razones por las que este diseño se descartó fueron:

- No existe como tal una capa de sistema propia del proyecto, y el uso del propio sistema *RESTful* de *MuleSoft* como capa de experiencia impide tener control acerca de la capa.
- Al contrario que en el prototipo anterior, la capa de proceso es la que está actuando en este caso como capa de sistema.

La idea principal de este diseño es que al usar el sistema *RESTful* de *MuleSoft* como capa de sistema permite ahorrarse la implementación de la capa de sistema con la pega de que no tienes control de esa capa. Además, hacer esto aumenta el trabajo de la capa de proceso ya que es la encargada de hacer las llamadas a *MuleSoft*.

Diseño final

Este último diseño está basado en el anterior pero añadiendo realmente la capa de sistema. Este diseño consiste en:

- **Capa de Experiencia (1 API)**: Esta *API* se encargaría de devolver al usuario los datos en formato *CSV* con la información necesaria para trabajar con *PowerBI*. Solo existe una única *API* en esta capa ya que en principio el proyecto solo va a ser usado por un único usuario.
- **Capa de Proceso (3 APIs)**: Estas *APIs* se encargarían de tratar y combinar la información obtenida en la capa de sistema para pasarla a la capa de experiencia.
- **Capa de Sistema (4 APIs)**: Estas *APIs* se encargarán de recibir la información de *MuleSoft* y devolverla toda a las *APIs* de proceso para su posterior tratado. Cada una de estas *APIs* llamaría a una parte diferente del sistema de *MuleSoft*.

Este diseño cumple con la arquitectura *API-Led* y permite hacer una división de las *APIs* de forma coherente. En la Figura 4.2 se puede ver en profundidad este diseño y los diversos *endpoints* que lo componen.

Algunos de los beneficios que tiene este diseño final son:

- Escalabilidad: En caso de que un usuario quisiera acceder a toda la información de *MuleSoft* podría acoplarse directamente a nuestro sistema haciendo uso de nuestra capa de sistema.
- División coherente de las capas: En este diseño cada una de las capas tiene un objetivo especifico cumpliendo con la arquitectura *API-Led*.
  - La capa de experiencia se encarga de trasformar la información a formato *CSV* y devolvérsela al usuario.
  - La capa de proceso se encarga de combinar datos y tratarlos para devolver únicamente las medidas necesarias.
  - La capa de sistema se encarga de extraer toda la información del sistema *RESTful* de *MuleSoft* y devolverla sin reducir la información obtenida.

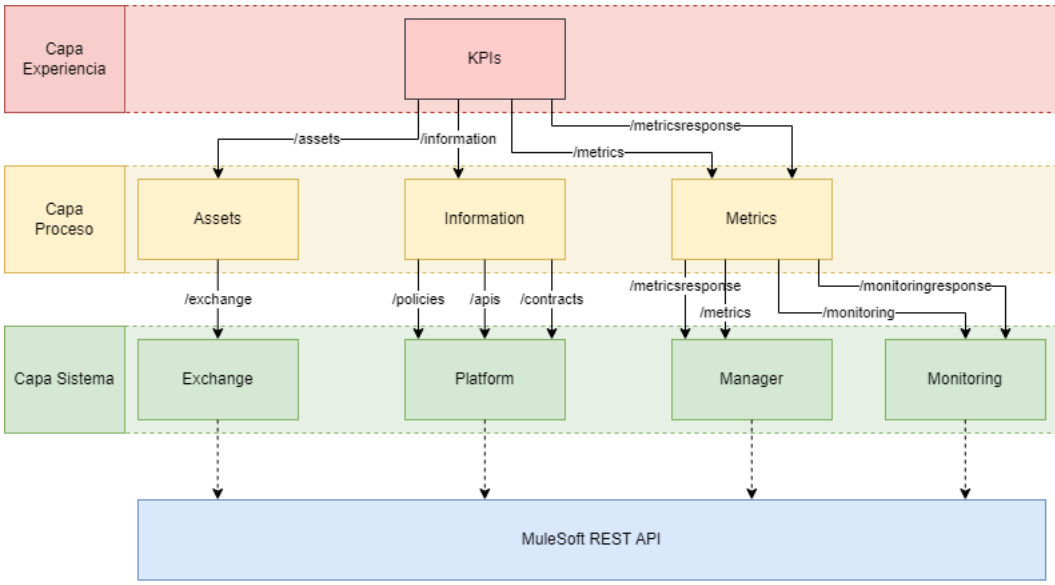

Figura 4.2: Diseño final del sistema

Capítulo 5

Implementación y pruebas

En este capítulo trataremos la implementación y las pruebas de cada una de las *APIs*, desde su modelado en el *Design Center* hasta su despliegue en *CloudHub*.

La implementación de las *APIs* siguen un proceso que se divide en los siguientes pasos:

1. Modelado en el *Design Center*.
2. Implementación en el *Anypoint Studio*.
3. Pruebas en el *Anypoint Studio*.
4. Despliegue en el *Runtime Manager*.
5. Gestión en el *API Manager*.

Tanto el primero como los dos últimos pasos se realizan de forma similar para todas las *APIs*. Por tanto, en este capítulo comentaremos primero cómo se realiza el modelado de una *API* en general, después iremos por cada una de las capas de la arquitectura comentando la implementación y las pruebas realizadas a cada una de las *APIs* de esa capa, y comentaremos finalmente se comentará el despliegue y la gestión de las *APIs* de forma general.

Es importante añadir que en todas las APIs, el único tipo de pruebas que han sido realizadas han sido pruebas de integración mediante Postman. Para ello se ha realizado una colección dentro de Postman con cada una de las APIs y sus diferentes pruebas. Para realizar una prueba, se realiza una llamada al endpoint en local y se comprueba que el resultado obtenido es el mismo que el resultado deseado al realizar esa llamada en condiciones especificas. Además, pruebas como la validación de credenciales para poder tener acceso a las APIs han sido realizadas en todas las APIs.

5.1. Modelado de las *APIs*

El modelado se realiza en el *Design Center*, ya que es la aplicación de *MuleSoft* que está dedicada a realizar el modelado de las *APIs* para la posterior implementación. Además, nos permite realizar pruebas para comprobar el correcto funcionamiento de los *endpoints* sin necesidad de implementarlos.

En el *Design Center* primero definimos los *endpoints* de nuestra *API* y en cada uno de estos *endpoints* definimos qué métodos *HTTP* queremos que realice y el formato que buscamos en la respuesta.

En este proyecto todos los *endpoints* hacen uso de la operación *GET*, ya que es la que se usa por defecto para leer información. En nuestro caso no tenemos la intención de crear (*POST*), modificar (*PUT* o *PATCH*) o borrar (*DELETE*) información dentro de la plataforma de *MuleSoft* por lo que no necesitamos hacer uso de otros métodos *HTTP*.

Por último, en cada una de las *APIs*, para definir el formato que buscamos en la respuesta de un *endpoint* haremos uso de un fichero *RAML* en el que definiremos cómo es la respuesta. Este fichero se acompaña con otro fichero *JSON* en el que introduciremos un ejemplo de la respuesta.

Por último, el *Design Center* se encargará de comprobar que el modelo de la aplicación no tenga fallos de sintaxis y comprueba que el formato de la respuesta y el formato del ejemplo coincidan. Acto seguido publicaremos el *API* en el *Exchange* para su posterior lanzamiento.

5.2. Implementación y pruebas de la capa de sistema

Como ya se comentó en el capítulo 4, esta capa tiene como objetivo sacar toda la información posible del sistema *API REST* de la plataforma de *MuleSoft*, intentando no modificar o eliminar ningún valor devuelto por la plataforma.

5.2.1. *API Exchange*

Está *API* tiene la función de sacar toda la información que contenga el apartado *Exchange* dentro de la plataforma de *MuleSoft*.

Implementación

Para implementar esta *API* se va recorriendo cada uno de los *Business Group* realizando una llamada al sistema *API REST* de *MuleSoft* para que recoja toda la información que, una vez formateada con la estructura que buscamos, es devuelta por el *API*.

En caso de que solo se requiera la información de un *Business Group* específico, se realiza una comprobación previa de los parámetros de la consulta para únicamente realizar la petición al sistema de *MuleSoft* con el *Business Group* específico.

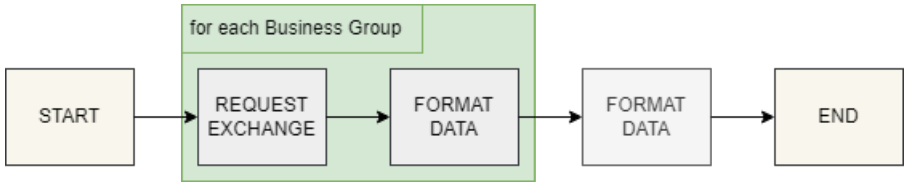

Figura 5.1: Implementación *API Exchange*

Pruebas

Para las pruebas en local se creará una batería de pruebas que comprobará todos los posibles casos. Las pruebas realizadas en esta aplicación son:

- En caso de no introducir un *Business Group* la aplicación devuelve la información de todos los *Business Group*.
- En caso de introducir un *Business Group* la aplicación devuelve solamente la información de ese *Business Group*.
- El formato de la respuesta es el esperado.

Todas estas pruebas han sido superadas de manera correcta.

5.2.2. *API Platform*

En el caso de esta *API* tenemos tres *endpoints* diferentes: */apis*, */policies* y */contracts*.

Implementación

La implementación del *endpoint /apis* se realiza obteniendo y recorriendo todos los *Business Group*. Dentro de cada uno de estos se obtienen todos los *Environments* y se recorren sacando todas las *APIs* que se encuentren dentro de ese cada uno de ellos, para finalmente, efectuar un recuento del número de *APIs*.

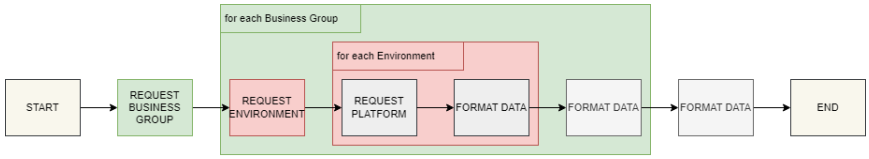

Figura 5.2: Implementación *API Platform /apis*

En el caso de los *endpoints /policies* y */contracts* la implementación se realiza de la misma manera. La novedad es que dentro de cada un de los *Environments* de estos *endpoints*, en lugar de contar únicamente las *APIs*, se va recorriendo cada una de ellas obteniendo las *policies* o los *contracts* según el *endpoint* al que se llame.

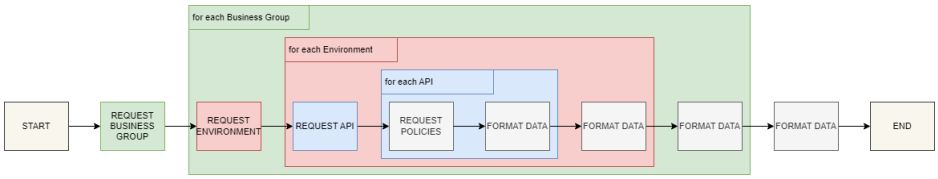

Figura 5.3: Implementación *API Platform /policies*

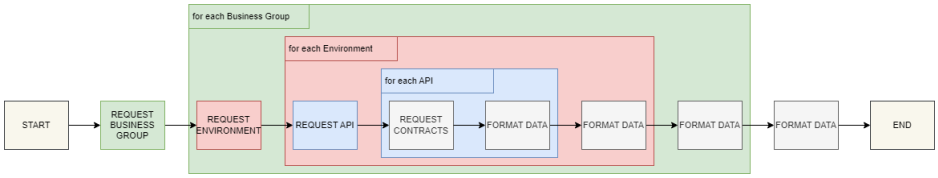

Figura 5.4: Implementación *API Platform /contracts*

Al igual que en la anterior *API*, en caso de que solo se quiera obtener la información de un especifico *Business Group* o *Environment*, se realiza una comprobación de los parámetros de la consulta y no se efectúan las llamadas para obtener todos los *Business Group* o los *Environment*.

Pruebas

Para las pruebas en local se realiza una batería de pruebas por cada uno de los *endpoints*. No obstante, dentro de cada batería, las pruebas que se realizarán serán las mismas, independientemente del *endpoint* al que se ataque.

- En caso de no introducir un *Business Group* la aplicación devuelve la información de todos los *Business Group*.

- En caso de no introducir un *Environment* la aplicación devuelve la información de todos los *Environment*.
- En caso de introducir un *Business Group* la aplicación devuelve solamente la información de ese *Business Group*.
- En caso de introducir un *Environment* la aplicación devuelve solamente la información de ese *Environment*.
- El formato de la respuesta es el esperado.

Todas estas pruebas han sido superadas de manera correcta.

5.2.3. *API Manager* y *API Monitoring*

Estas dos *APIs*, a pesar de llamar a dos puntos distintos del sistema *API REST* de *MuleSoft*, se implementan de forma muy parecidas.

Implementación

En ambos casos lo más importante de la implementación es que el flujo principal se desarrolla de forma asíncrona. Esto significa que al realizar la llamada a cualquiera de estas *APIs* se iniciará una descarga de los datos de *MuleSoft* en segundo plano.

La necesidad de que la descarga de los datos se realice de forma asíncrona viene motivada por los dilatados tiempos de respuesta que se producen al implementar la llamada de forma síncrona (como ocurre en el resto de *APIs*), llegando en algunos casos a los 20 minutos de espera. Además, el hecho de que *CloudHub* (La plataforma en la nube de *MuleSoft* donde están desplegadas las *APIs*) no permite tiempos de respuesta superiores a los 5 minutos impedía que la respuesta de estas *APIs* pudiera ser entregada de manera síncrona.

Para obtener la *asincronía* de estas *APIs* se ha usado un *Object Store*, un conector de *Mule* que permite guardar información dentro del *API* y que esta información no sea borrada al finalizar el flujo de la llamada. Para usar el *Object Store* es necesario una clave y un valor. En nuestro caso la clave sera un identificador único que será devuelto al usuario para poder obtener la información que ha solicitado. Por otro lado, el valor será la descarga en cuestión cuando esta haya terminado y, mientras tanto, un mensaje indicando que la descarga está en proceso.

En cuanto al proceso que se ejecuta dentro de las *APIs*, ambas tienen dos *endpoints* que actúan de forma parecida.

- ***/manager*** o ***/monitoring*** que se encargan de realizar la llamada asíncrona para la descarga de la información y devuelven el identificador único con el que obtener la información.

  - ***Input***: *Business Group*, *Environment* y Identificador.
  - ***Output***: Identificador.

- ***/managerresponse*** o ***/monitoringresponse*** que se encargan de devolver la información una vez descargada o de avisar al usuario de que la descarga está en curso en otro caso.

  - ***Input***: Identificador.
  - ***Output***: Información.

La parte asíncrona se encuentra dentro del primer *endpoint* y se encarga en primer lugar de obtener todos los *Business Group* y recorrer cada uno de ellos. Dentro de cada uno de los *Business Group* se obtienen y recorren todos los *Environments* extrayendo todas las *APIs* que ahí se encuentran. Toda la información de estas *APIs* se obtienen del sistema *API REST* de *MuleSoft*.

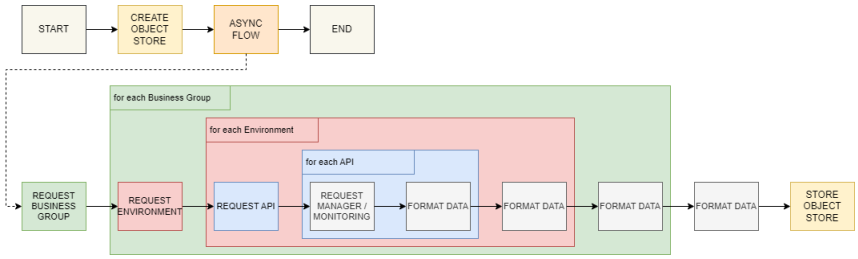

Figura 5.5: Implementación *API Manager* o *API Monitoring /manager* o */monitoring*

Figura 5.6: Implementación *API Manager* o *API Monitoring /managerresponse* o */monitoringresponse*

Pruebas

En estas *APIs* las pruebas que se realizan a cada uno de los *endpoints* son diferentes. Por lo que realizaremos una batería diferente por cada *endpoint*.

Pruebas para los *endpoints /manager* y */monitoring*:

- En caso de no introducir un *Business Group* la aplicación devuelve la información de todos los *Business Group*.

- En caso de no introducir un *Environment* la aplicación devuelve la información de todos los *Environment*.
- En caso de introducir un *Business Group* la aplicación devuelve solamente la información de ese *Business Group*.
- En caso de introducir un *Environment* la aplicación devuelve solamente la información de ese *Environment*.
- En caso de no introducir Identificador la aplicación no realiza la llamada.
- El formato de la respuesta es el esperado.

Pruebas para los *endpoints /managerresponse* y */monitoringresponse*:

- En caso de no introducir Identificador la aplicación no realiza la llamada.
- En caso de introducir Identificador que no existe la aplicación indica que no existe.
- En caso de introducir Identificador y la descarga de información no haya terminado la aplicación indica que está en proceso.
- En caso de introducir Identificador y la descarga de información haya terminado la aplicación devuelve la salida esperada.

Todas estas pruebas han sido superadas de manera correcta.

5.3. Implementación y pruebas de la capa de proceso

Estas *APIs* se encargan de obtener la salida de la capa de sistema y devolver a la capa de experiencia solamente lo que nos interese de cada una de las respuestas reduciendo o generando campos nuevos en los *JSON* según sea necesario.

5.3.1. *API Assets*

Esta *API* se encarga de reducir la respuesta obtenida por el *API Exchange*, eliminando bastantes campos que no son necesarios para obtener *KPIs*.

Implementación

Para implementar este *API* únicamente se realiza una llamada al *API Exchange* de la capa de sistema y se le da formato a la respuesta haciendo uso de la función *map* que nos

permite dar formato a los elementos de un *array* de objetos *JSON*. Esta respuesta se devuelve también en formato *JSON*.

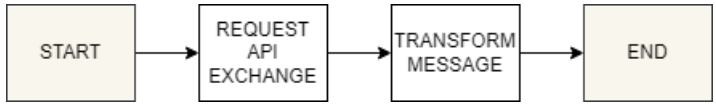

Figura 5.7: Implementación *API Assets*

Pruebas

Las pruebas realizadas a esta *API* han sido:

- En caso de no introducir un *Business Group* la aplicación devuelve la información de todos los *Business Group*.
- En caso de introducir un *Business Group* la aplicación devuelve solamente la información de ese *Business Group*.
- El formato de la respuesta es el esperado.

Todas estas pruebas han sido superadas de manera correcta.

5.3.2. *API Information*

Este *API* se encarga de obtener toda la información de cada uno de los *endpoints* del *API Platform* y de combinar esta información así como reducir la innecesaria.

Implementación

Para su implementación primero se llama a los tres *endpoints* del *API Platform* y se guarda la salida de cada uno de ellos en variables. Acto seguido, esta información es recorrida y es combinada y reducida por el *API* para, posteriormente, devolverla en formato *JSON*.

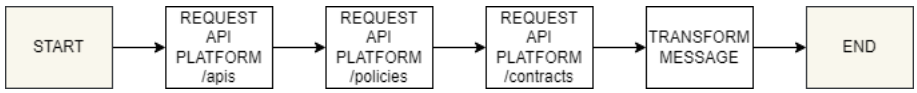

Figura 5.8: Implementación *API Information*

Dado que en esta *API* se realizan tres peticiones seguidas, los tiempos de respuesta son algo altos, llegando a los 30 segundos en caso de sacar la información de todos los datos. Sin embargo, ya que no es un tiempo excesivo no ha sido introducida *asincronía* en esta *API* como ocurría en *API Manager* o en *API Monitoring*.

Pruebas

Las pruebas realizadas a esta *API* han sido:

- En caso de no introducir un *Business Group* la aplicación devuelve la información de todos los *Business Group*.
- En caso de no introducir un *Environment* la aplicación devuelve la información de todos los *Environment*.
- En caso de introducir un *Business Group* la aplicación devuelve solamente la información de ese *Business Group*.
- En caso de introducir un *Environment* la aplicación devuelve solamente la información de ese *Environment*.
- El formato de la respuesta es el esperado.

Todas estas pruebas han sido superadas de manera correcta.

5.3.3. *API Metrics*

Este *API* se encarga de llamar a *API Manager* y a *API Monitoring* tanto para dar inicio a una descarga de información como para obtener la información obtenida.

Implementación

En este caso tenemos dos *endpoints* al igual que en las *APIs* a las que llama:

- ***/metrics***, que se encarga de realizar las llamadas para dar inicio a la descarga de información. Además, genera un identificador único que envía como parámetro a estas llamadas y lo devuelve para poder obtener la información una vez esté lista.
  - ***Input***: *Business Group* y *Environment*.
  - ***Output***: Identificador.
- ***/metricsresponse***, que se encarga de devolver la información una vez descargada o, en su caso, de avisar al usuario de que la descarga está en curso. Para ello comprueba la salida de ambos *endpoints* de respuesta de las *APIs* de la capa de sistema y, una vez ambos hayan terminado, devuelve la información combinada y reducida de ambos *APIs*; en caso de que aún no esté preparada la información, informa de que la llamada sigue en curso.
  - ***Input***: Identificador.

- ***Output***: Información.

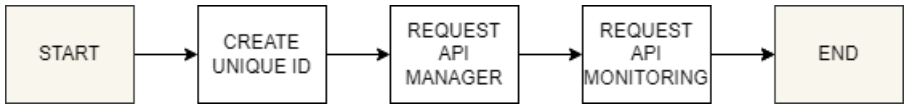

Figura 5.9: Implementación *API Metrics /metrics*

Figura 5.10: Implementación *API Metrics /metricsresponse*

Pruebas

En esta *APIs* las pruebas que se realizan para cada *endpoint* son:

Pruebas para el *endpoint /metrics*:

- En caso de no introducir un *Business Group* la aplicación devuelve la información de todos los *Business Group*.
- En caso de no introducir un *Environment* la aplicación devuelve la información de todos los *Environment*.
- En caso de introducir un *Business Group* la aplicación devuelve solamente la información de ese *Business Group*.
- En caso de introducir un *Environment* la aplicación devuelve solamente la información de ese *Environment*.
- El formato de la respuesta es el esperado.

Pruebas para el *endpoint /metricsresponse*:

- En caso de no introducir Identificador la aplicación no realiza la llamada.
- En caso de introducir Identificador que no existe la aplicación indica que no existe.
- En caso de introducir Identificador y la descarga de información no haya terminado la aplicación indica que está en proceso.
- En caso de introducir Identificador y la descarga de información haya terminado la aplicación devuelve la salida esperada.

Todas estas pruebas han sido superadas de manera correcta.

5.4. Implementación y pruebas de la capa de proceso

En esta capa solo hay una *API* y es la encargada de devolver la información al usuario con el formato deseado; en el caso de este proyecto, el formato deseado es en *CSV*.

5.4.1. *API KPIs*

Esta *API* consta de cuatro *endpoints* que se encargan de llamar a las tres *APIs* que se encuentran en la capa de proceso (*PAPIs*). Los *endpoints* que tenemos son: */assets*, */information*, */metrics* y */metricsresponse*.

Implementación

La implementación de todos los *endpoints* es igual, excepto el *endpoint* de */metrics* que únicamente se encarga de iniciar una descarga de información en *API Metrics*.

Figura 5.11: Implementación *API KPIs /metrics*

El resto de los *endpoints* siguen un mismo formato y se encargan de realizar llamadas a las *APIs* de la capa de proceso y aplanar la respuesta para pasarla a formato csv.

Figura 5.12: Implementación *API KPIs /assets, /information* y */metricsresponse*

La parte más interesante de esta *API* es el proceso de aplanado de la respuesta de las *PAPIs* para devolverla en formato *CSV*. Esto se hace ya que la respuesta de las *PAPIs* contienen *arrays* de objetos *JSON* anidados y no es posible formatearlo directamente. Para ello se opta por realizar un aplanado de estos *arrays*. Este proceso consiste en reducir la repuesta de las *PAPIs* a un único *array* con objetos *JSON* pero sin tener pérdidas en la información.

Por poner un ejemplo del aplanado, si nuestra entrada fuera esta:

```
{
    "Escuela": "Alfa",
    "Alumnos": [
        {
            "Nombre": "Juan",
            "Apellidos": "Medina-Bocos Lorenzo"
        },
        {
            "Nombre": "Miguel",
            "Apellidos": "Martínez Uriarte"
        }
    ]
},
{
    "Escuela": "Beta",
    "Alumnos": [
        {
            "Nombre": "Diego",
            "Apellidos": "Velázquez Lorca"
        }
    ]
}
```

Figura 5.13: Entrada aplanado.

La salida que daría nuestro aplanado sería la siguiente:

```
{
    "Escuela": "Alfa",
    "AlumnoNombre": "Juan",
    "AlumnoApellidos": "Medina-Bocos Lorenzo",
},
{
    "Escuela": "Alfa",
    "AlumnoNombre": "Miguel",
    "AlumnoApellidos": "Martínez Uriarte",
},
{
    "Escuela": "Beta",
    "AlumnoNombre": "Diego",
    "AlumnoApellidos": "Velázquez Lorca",
}
```

Figura 5.14: Salida aplanado.

Pruebas

Las pruebas realizadas a esta *API* se dividen según el *endpoint* al que se realiza la llamada. Las pruebas realizadas a los *endpoints /assets*, */information* y */metrics* han sido:

- En caso de no introducir un *Business Group* la aplicación devuelve la información de todos los *Business Group*.
- En caso de no introducir un *Environment* la aplicación devuelve la información de todos los *Environment*.
- En caso de introducir un *Business Group* la aplicación devuelve solamente la información de ese *Business Group*.
- En caso de introducir un *Environment* la aplicación devuelve solamente la información de ese *Environment*.
- El formato de la respuesta es el esperado.

Pruebas para el *endpoint /metricsresponse*:

- En caso de no introducir Identificador la aplicación no realiza la llamada.
- En caso de introducir Identificador que no existe la aplicación indica que no existe.
- En caso de introducir Identificador y la descarga de información no haya terminado la aplicación indica que está en proceso.
- En caso de introducir Identificador y la descarga de información haya terminado la aplicación devuelve la salida esperada.

Todas estas pruebas han sido superadas de manera correcta.

5.5. Lanzamiento

Para realizar el lanzamiento de las *APIs* en primer lugar tenemos que generar unos ficheros *JAVA* con las *APIs* compiladas desde *Anypoint Studio*. Después, desde el *Runtime Manager* se puede lanzar el fichero *JAVA* en *CloudHub*.

Este proceso no tiene mucha dificultad ya que solo requiere que la aplicación compile y funcione en local; es aquí donde a la aplicación se le asigna una *url* para poder atacarla.

5.6. Gestión y control de las *APIs*

Una vez desplegadas las aplicaciones, queremos poder gestionarlas y controlarlas haciendo uso de políticas y contratos.

5.6.1. Configuración en el *API Manager*

El *API Manager* es una funcionalidad que aporta *MuleSoft* para poder gestionar y controlar tus *APIs* asignándolas políticas de seguridad o contratos para limitar su uso, entre otras muchas cosas.

En nuestro caso, para poder aplicar políticas y contratos a nuestras *APIs* debemos primero vincular nuestra aplicación lanzada en el *Runtime Manager* con el *API Manager*. Para ello se hace uso del *Autodiscovery ID*.

En primer lugar, debemos generar una instancia dentro del *API Manager* que haga referencia a una *API* que existe en el *Exchange*. En nuestro caso, al final del modelado subimos todas las *APIs* al *Exchange*, así que vamos generando instancias de cada uno dentro del *API Manager*.

Después, debemos introducir la *ID* proporcionada por estas instancias en la implementación del *API* mediante un conector llamado *Autodiscovery*. Esto permitirá que nuestro *API* esté vinculada al *API Manager* una vez sea redesplegada.

5.6.2. Políticas

En nuestras *APIs* queremos introducir una política de seguridad, de modo que solo se permita la llamada si se cuenta con la clave de acceso. Esto se realiza desde el *API Manager* en el apartado de *Policies* y es la propia aplicación de *MuleSoft* la que se encarga de gestionar y controlar las políticas.

5.6.3. Contratos

Para poder acceder a nuestros *APIs* primero se debe solicitar el acceso; en caso de que este sea autorizado, se generarán unas claves que permitirán el acceso. Estas solicitudes de acceso se realizan desde el *Exchange*; sin embargo, puedes controlar los contratos generados en cada *API* desde el *API Manager*.

Capítulo 6

Visualización de datos mediante *PowerBI*

En este capítulo se hablará acerca del uso que le hemos dado a la información obtenida por nuestro sistema.

6.1. Objetivo

El objetivo principal de esta parte es realizar unos cuantos gráficos y tablas que permitan una visualización rápida de los datos que nos ayuden a tomar decisiones de cara al futuro. Por ejemplo, si observamos que los tiempos de respuestas de una *API* son muy altos o que un gran porcentaje de las llamadas realizadas se traducen en un fallo deberíamos de derivar esfuerzos de la empresa a tratar con estos posibles problemas para poder dar una mejor solución al cliente. Dicho esto, en nuestro caso queremos mostrar de forma detallada y sencilla los siguientes datos:

- Primer objetivo: Poder ver el número de *Assets* en el *Exchange* y quién los ha creado. Además, poder filtrar por grupo de negocio, tipo y creador.
- Segundo objetivo: Poder ver el número de Aplicaciones y poder filtrar por grupo de negocio y entorno.
- Tercer objetivo: Poder ver el recuento de políticas aplicadas a las *APIs* y poder filtrar por grupo de negocio, entorno y aplicación.
- Cuarto objetivo: Poder ver el número de contratos que tiene cada *API* y poder filtrar por grupo de negocio y por entorno.
- Quinto objetivo: Poder ver el promedio del número de llamadas y del número de errores de cada *API*, pudiendo filtrar por grupo de negocio, entorno y aplicaciones.

- Sexto objetivo: Poder ver los tiempos de respuesta de cada aplicación y poder filtrar por grupo de negocio, entorno y aplicación

6.2. Visualización

Para explotar los datos se ha creado un proyecto dentro de la herramienta *PowerBI* en el que en cada una de las páginas se cumple un objetivo de los citados anteriormente. En este pequeño trabajo de visualización, se han hecho uso de principalmente tres tipos de herramientas: gráficos de columnas apiladas, gráficos de columnas agrupadas, gráficos de anillos y tablas.

6.2.1. Gráficos de columnas apiladas

Este tipo de gráficos tienen como objetivo comparar valores numéricos de una variable categórica y la descomposición de cada una de las barras.[20]

Este gráfico ha sido usado para el primer y el cuarto objetivo ya que en ambos podemos dividir cada una de las barras entre quien las ha creado (primer objetivo) y las aplicaciones (cuarto objetivo). Además, este a este gráfico se le pueden aplicar filtros para que solo muestre la información que desea el usuario.

A continuación se añade una figura con un ejemplo de un gráfico de columnas apiladas generado con nuestros datos (primer objetivo). Se ha procedido a ocultar información confidencial de la empresa como nombres de usuarios o grupos de negocio.

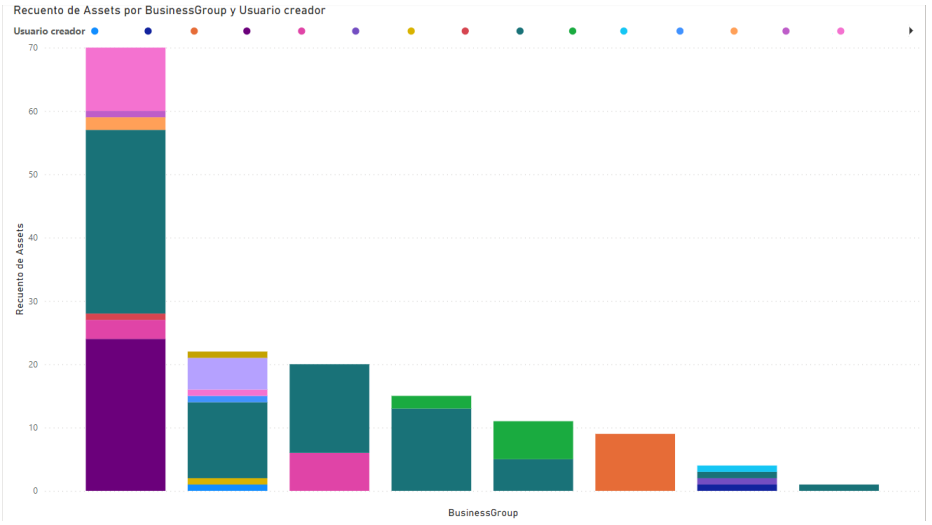

Figura 6.1: Gráficos de columnas apiladas

6.2.2. Gráficos de columnas agrupadas

Este tipo de gráfico sirve principalmente para ayudar a resumir los datos por grupos de casos.[5] Aunque principalmente este tipo de gráficos se usen para comparar variaciones en un intervalo de tiempo, sin embargo nos puede servir bastante bien para el quinto y el sexto objetivo.

En este caso cada uno de los grupos de columnas serán los grupos de negocios, entornos o aplicaciones dependiendo de la capa en la que estemos. En el quinto objetivo, en cada grupo de columnas habrá dos columnas, una que nos mostrará el promedio del número de llamadas al *API* o a los *APIs* y otra que nos mostrará el número de errores. Por otro lado, en el sexto objetivo habrá tres columnas: una que nos mostrará el tiempo máximo de respuesta, otra que nos mostrará el tiempo mínimo y otra que nos mostrará el tiempo medio.

Estos gráficos los acompaño con unas tarjetas que muestran de forma numérica el promedio del número de llamadas, el número de errores y la tasa de error en el caso del quinto objetivo y muestra los tres tiempos de respuesta en el sexto objetivo. Estos valores cambian en función del filtro que se aplique a la página.

A continuación se añade una figura con un ejemplo de un gráfico de columnas agrupadas generado con nuestros datos (sexto objetivo). Se ha procedido a ocultar información confidencial de la empresa como nombres de aplicaciones o grupos de negocio.

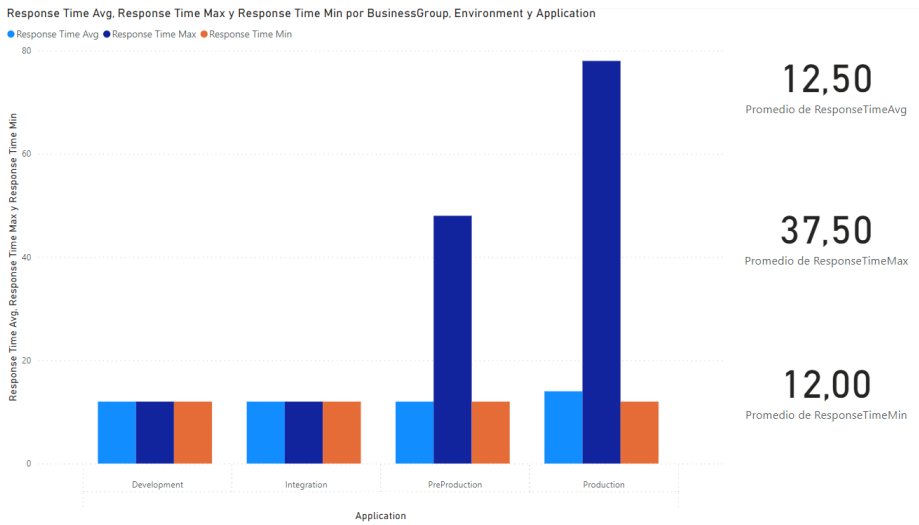

Figura 6.2: Gráficos de columnas agrupadas

6.2.3. Gráficos de anillos

Este tipo de gráficos sirven para mostrar la proporción frente al total de los datos.[4]

Ha sido usado para el segundo y tercer objetivo. En el segundo objetivo me interesa saber la proporción de aplicaciones que tiene cada uno de los grupos de negocio y dentro de cada grupo de negocio cuántas aplicaciones tiene cada entorno. Por otro lado, en el tercer objetivo me interesa saber la proporción de aplicaciones qué usan ciertas políticas *JWT* y poder ver qué aplicaciones las usan y cuáles no.

A continuación se añade una figura con un ejemplo de un gráfico de anillos generado con nuestros datos (segundo objetivo). Se ha procedido a ocultar información confidencial de la empresa como grupos de negocio.

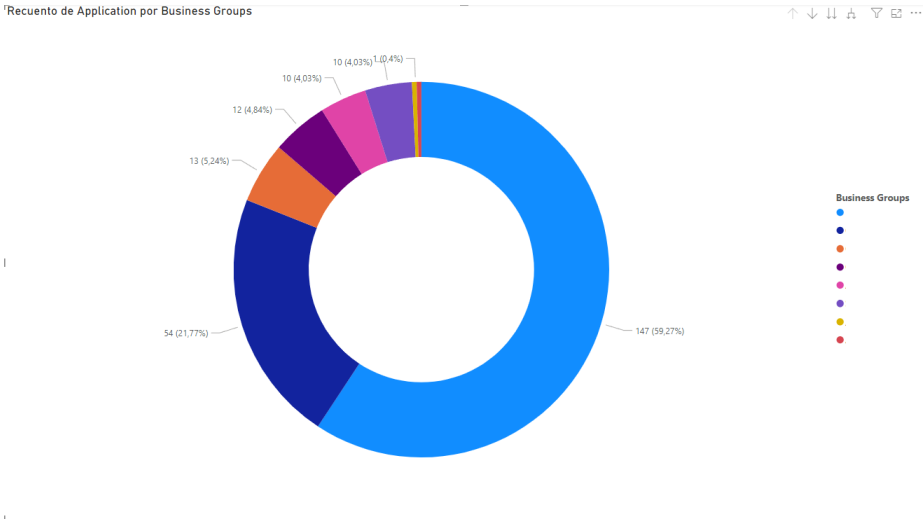

Figura 6.3: Gráficos de anillos

Capítulo 7

Conclusiones

Una vez finalizado el proyecto se puede hablar acerca de lo que se ha aprendido y como ha sido su desarrollo.

En primer lugar, se han cumplido todos los objetivos que se habían propuesto en la planificación del proyecto. El sistema de *APIs* ha sido implementado correctamente y funciona en su totalidad con unos tiempos de respuesta aceptables.

A pesar de que la realización de este proyecto me ha llevado más tiempo de lo esperado, el hecho de haber trabajado en la empresa en proyectos parecidos me ha facilitado el trabajo, sin embargo, conseguir que todo funcione de manera deseada ha requerido más tiempo y esfuerzo que el estimado al inicio del proyecto.

Al hacer este proyecto he advertido la importancia de las *APIs* en el mundo de la informática; sobre todo cómo han cambiado la forma de enviar información y cómo son prácticamente invisibles para muchas personas, que piensan que los datos se encuentran ya ubicados en las páginas web o que el proceso de extraer la información de bases de datos es inmediato y sencillo.

Desde que lo he descubierto con el trabajo, siento que el mundo de las APIs es una rama muy interesante de la informática, y haber podido realizar un proyecto de este tipo supone una gran ayuda para acercarme al desarrollo de APIs.

Por último, quisiera agradecer sinceramente al tutor de la empresa todo el tiempo que ha dedicado a resolver las dudas que me han ido surgiendo, ayudándome a enfocar el proyecto y la memoria para convertirlos en el trabajo de fin de grado que pone fin a mi etapa universitaria.

Bibliografía

[1] Amalia Taylor. ¿cómo crear puntos finales y por qué los necesita? https://appmaster.io/es/blog/como-crear-puntos-finales-y-por-que-los-necesita. Febrero 2023.

[2] Anastasia Stsepanets. Modelo de cascada (waterfall): qué es y cuándo conviene usarlo. https://blog.ganttpro.com/es/metodologia-de-cascada/. Febrero 2023.

[3] BBVA. Breve historia de las apis: del comercio electrónico a la era móvil. https://www.bbvaapimarket.com/es/mundo-api/breve-historia-de-las-apis-del-comercio-electronico-la-era-movil/. Marzo 2023.

[4] ChartJS. Doughnut and pie charts. https://www.chartjs.org/docs/latest/charts/doughnut.html#doughnut. Febrero 2023.

[5] IBM. Gráficos de barras agrupadas. https://www.ibm.com/docs/es/spss-statistics/28.0.0?topic=crosstabs-clustered-bar-charts. Febrero 2023.

[6] Jacobson, Daniel and Brail, Greg and Woods, Dan. Apis: A strategy guide. 2012.

[7] Johnna Cloded Menendez. ¿qué es power bi? https://www2.deloitte.com/es/es/pages/technology/articles/que-es-power-bi.html. Febrero 2023.

[8] Marco Antonio Sanz. Raml: La transformaciÓn digital llega a la definiciÓn de apis. https://openexpoeurope.com/es/raml-definicion-apis/. Febrero 2023.

[9] Matias Trionfetti. Conoce las ventajas mulesoft y su plataforma anypoint platform. https://s4g.es/blog/salesforce/mulesoft-que-es-y-ventajas-de-uso/. Febrero 2023.

[10] Microsoft. Powerbi. https://powerbi.microsoft.com/es-es/. Febrero 2023.

[11] Mike Cotterell y Bob Hughes. Software project management. 5.a ed. tata mcgraw hill education private limited. 2011.

[12] Northware. Requerimientos en el desarrollo de software y aplicaciones. https://www.northware.mx/blog/requerimientos-en-el-desarrollo-de-software-y-aplicaciones/. Febrero 2023.

[13] Overleaf. Overleaf - documentación. https://www.overleaf.com/learn. Febrero 2023.

[14] Postman. What is postman? https://www.postman.com/product/what-is-postman/. Mayo 2023.

[15] RED HAT. ¿qué es una api y cómo funciona? https://www.redhat.com/es/topics/api/what-are-application-programming-interfaces. Febrero 2023.

[16] SALESFORCE BLOG. ¿qué es mulesoft? ¿componentes de anypoint platform? https://www.nts-solutions.com/blog/salesforce-mulesoft-que-es.html#:~:text=Anypoint%20Design%20Center&text=Herramienta%20que%20permite%20la%20creacin,componentes%20que%20conforman%20una%20integracin. Febrero 2023.

[17] SourceTree. Sourcetree. https://www.sourcetreeapp.com. Febrero 2023.

[18] SOYPM. Gestión de riesgos pmbok 6. https://www.soypm.website/area-de-conocimiento/gestion-de-riesgos/. Febrero 2023.

[19] SYDLE. Kpis: ¿qué son, cuál es su importancia y cómo utilizarlos? https://www.sydle.com/es/blog/kpi-615de90225ce5d3ef29a5570#:~:text=Del%20ingls%2C%20el%20acrnimo%20KPI,las%20estrategias%20de%20un%20negocio. Febrero 2023.

[20] Tibco. ¿qué es un gráfico apilado? https://www.tibco.com/es/reference-center/what-is-a-stacked-chart. Febrero 2023.

[21] Universidad Carlos III de Madrid. Overleaf - editor online latex. https://www.uc3m.es/sdic/servicios/overleaf. Febrero 2023.

[22] Wikipedia. Diagrama de casos de uso. https://es.wikipedia.org/wiki/Caso_de_uso. Mayo 2023.

[23] Wikipedia. Metodología de desarrollo de software. https://es.wikipedia.org/wiki/Metodologa_de_desarrollo_de_software#Modelo_en_cascada. Febrero 2023.

[24] Yúbal Fernández. Api: qué es y para qué sirve. https://www.xataka.com/basics/api-que-sirve. Febrero 2023.