UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Desarrollo de una solución de integración para dar soporte a la gestión de pedidos E-commerce

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Alpuente I Parrilla, Pau

Tutor/a: Valderas Aranda, Pedro José

Cotutor/a externo: CABANES ASENSIO, CARLOS

CURSO ACADÉMICO: 2022/2023

Resumen

Actualmente, la integración de aplicaciones se ha convertido en un aspecto fundamental en el mundo de la tecnología. A medida que las empresas utilizan cada vez más sistemas y plataformas para su funcionamiento, la necesidad de conectar y sincronizar estos sistemas se ha vuelto crucial para garantizar una comunicación eficiente y un flujo de datos fluido. La integración permite a las organizaciones optimizar sus operaciones, mejorar la eficiencia y ofrecer una mejor experiencia a sus clientes.

En el ámbito específico de este trabajo, se abordará el desarrollo de un middleware para brindar soporte a una integración relacionada con E-commerce. Este middleware actuará como un intermediario entre el almacén y los diversos sistemas que deben recibir la actualización del estado de los pedidos. Al adoptar esta solución, se logrará una total desvinculación entre la aplicación y los sistemas existentes del cliente, permitiendo una comunicación más ágil y una gestión eficiente de la integración.

El middleware estará compuesto por diferentes APIs REST separadas por capas. Cada API tendrá un propósito específico, desde conectar con el almacén y orquestar la lógica de negocio hasta propagar la actualización del estado a los diferentes sistemas destinatarios. Esta arquitectura API-Led garantizará una comunicación más clara y organizada entre los sistemas, facilitando el desarrollo y mantenimiento del middleware.

En la memoria se detallarán todos los pasos seguidos en el desarrollo de la solución, adjuntando imágenes y tablas que permitan una mejor comprensión. Partiendo de un análisis del contexto, pasando por el diseño de las APIs, seguidamente por su implementación y la evaluación del funcionamiento. Finalmente, se presentarán las conclusiones obtenidas, destacando los logros alcanzados y los beneficios que aportará esta solución de integración.

Palabras clave: Integración, API, REST, middleware, E-commerce, capas, API-Led, flujos, transformaciones, peticiones

Resum

Actualment, la integració d'aplicacions s'ha convertit en un aspecte fonamental en el món de la tecnologia. A mesura que les empreses fan servir cada vegada més sistemes i plataformes per al seu funcionament, la necessitat de connectar i sincronitzar aquests sistemes s'ha tornat crucial per a garantir una comunicació eficient i un flux de dades fluït. La integració permet a les organitzacions optimitzar les seues operacions, millorar l'eficiència i oferir una millor experiència als seus clients.

En l'àmbit específic d'aquest treball, s'abordarà el desenvolupament d'un middleware per a brindar suport a una integració relacionada amb E-commerce. Aquest middleware actuarà com un intermediari entre el magatzem i els diversos sistemes que han de rebre l'actualització de l'estat dels comandes. En adoptar aquesta solució, s'aconseguirà una total desvinculació entre l'aplicació i els sistemes existents del client, permetent una comunicació més àgil i una gestió eficient de la integració.

El middleware estarà compost per diferents APIs REST separades per capes. Cada API tindrà un propòsit específic, des de connectar amb el magatzem i orquestrar la lògica de negoci fins a propagar l'actualització de l'estat als diferents sistemes destinataris. Aquesta arquitectura API-Led garantirà una comunicació més clara i organitzada entre els sistemes, facilitant el desenvolupament i manteniment del middleware.

En la memòria es detallaran tots els passos seguits en el desenvolupament de la solució, adjuntant imatges i taules que permetran una millor comprensió. Partint d'una anàlisi del context, passant pel disseny de les APIs, a continuació per la seua implementació i l'avaluació del funcionament. Finalment, es presentaran les conclusions obtingudes, destacant els assoliments aconseguits i els beneficis que aportarà aquesta solució d'integració.

Paraules clau: Integració, API, REST, middleware, E-commerce, capes, API-Led, fluxes, transformacions, peticions

Abstract

Currently, the integration of applications has become a fundamental aspect in today's technology world. As companies increasingly use multiple systems and platforms for their operations, the need to connect and synchronize these systems has become crucial to ensure efficient communication and smooth data flow. Integration allows organizations to optimize their operations, improve efficiency, and provide a better experience to their customers.

In the specific scope of this work, we will address the development of middleware to support E-commerce-related integration. This middleware will act as an intermediary between the warehouse and the various systems that need to receive order status updates. By adopting this solution, complete decoupling between the application and the client's existing systems will be achieved, enabling more agile communication and efficient integration management.

The middleware will be composed of different REST APIs separated by layers. Each API will serve a specific purpose, ranging from connecting to the warehouse and orchestrating business logic to propagating the status update to the different recipient systems. This API-Led architecture will ensure clearer and organized communication between the systems, facilitating the development and maintenance of the middleware.

The document will detail all the steps taken in the solution's development, including accompanying images and tables to enhance understanding. Starting from an analysis of the context, followed by the design of the APIs, moving on to their implementation, and concluding with an evaluation of their functionality. Finally, the obtained conclusions will be presented, highlighting the achievements and benefits that this integration solution will bring.

Key words: Integration, API, REST, middleware, E-commerce, layers, API-Led, flows, transformations, requests

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1

Introducción

La integración de aplicaciones se ha convertido en una necesidad cada vez más crítica en el mundo empresarial actual. La capacidad de conectar diferentes sistemas y aplicaciones para compartir información y automatizar procesos es esencial para mejorar la eficiencia y la eficacia de los procesos empresariales, y para mantenerse competitivo en un mercado en constante evolución.

Actualmente, el comercio electrónico (E-commerce) se está convirtiendo en una forma cada vez más popular de realizar compras, y ha llevado a muchas empresas a adaptar sus modelos de negocio para aprovechar esta oportunidad. Sin embargo, con la creciente demanda de compras en línea, también ha surgido la necesidad de integrar diversas aplicaciones y sistemas para poder ofrecer una experiencia de compra fluida y satisfactoria a los clientes.

En el siguiente trabajo se pretende desarrollar una solución de integración de aplicaciones para actualizar el estado de los pedidos E-commerce. Esta solución está basada en un caso de uso real en el cual está trabajando DISID Corporation. Para llevarla a cabo se implementarán nuevas funcionalidades a servicios REST ya creados por DISID y se integrarán entre ellos.

La idea principal es seguir la arquitectura API-Led, que se basa en 3 capas principales: experiencia, proceso y sistema. Esto permite escalar de manera más rápida y efectiva, ya que estas pueden ser combinadas y reutilizadas fácilmente para generar soluciones de software más grandes y complejas. En resumen, la arquitectura API-Led ayuda a las empresas a originar soluciones de software más

flexibles, escalables y adaptables para satisfacer las necesidades de sus clientes y del mercado en constante cambio.

Una vez desarrollada la solución de integración propuesta, se espera que se automatice la actualización del estado de los pedidos E-commerce, mejorando así la experiencia del cliente y aumentando la productividad de la empresa.

1.1 Motivación

La motivación detrás de este proyecto radica en la necesidad cada vez más crítica de integración de aplicaciones en el mundo empresarial actual. En este caso, los comercios están teniendo que adaptarse a las nuevas tecnologías, invirtiendo cada vez más en E-commerce, que lleva años en alza, pero sobre todo debido a la pandemia mundial que aceleró mucho su crecimiento, ya que las personas tuvieron que adaptarse a las restricciones de movimiento y distanciamiento social.

La integración de aplicaciones fue introducida en mi etapa estudiantil gracias a la asignatura IAP que cursé en el último año. Desde el primer momento despertó gran interés en mí y es por este motivo que decidí realizar las prácticas de empresa en DISID, ya que son especialistas en Cloud, MuleSoft, IBM Cloud Pak® for Data y low-code con Appian.

En DISID he tenido la oportunidad de trabajar en casos de uso reales empleando las distintas herramientas que proporciona MuleSoft, como Anypoint Studio o Anypoint Platform. Gracias a los conocimientos adquiridos, tanto en la asignatura IAP como en la formación de las prácticas, he podido colaborar en el proyecto sobre el que llevar a cabo este trabajo. Durante estos meses de trabajo me he dado cuenta de lo mucho que he aprendido y de la infinidad de posibilidades que ofrece la integración de aplicaciones en el ámbito laboral.

1.2 Objetivos

El objetivo principal de este trabajo es implementar una solución basada en el diseño técnico que ha realizado DISID para el proyecto que les ha sido propuesto.

Esta solución permitirá que los almacenes notifiquen, a través de una empresa de logística externa, los cambios de estado en los pedidos e-commerce. La solución se integrará dentro de una arquitectura ya existente, permitiendo escalabilidad y reusabilidad en un futuro.

Para lograr el objetivo final, se van a dividir las tareas en tres subobjetivos:

1. Diseñar los recursos que van a tener las APIs.
2. Implementar los flujos de integración.
3. Evaluar el funcionamiento y rendimiento de la solución.

1.3 Estructura de la memoria

La memoria del trabajo da comienzo en el Capítulo 2, en este se van a comparar las distintas tecnologías que se pueden utilizar en el ámbito de la integración de aplicaciones. Además, se justificará la elección de las API REST por encima de otras alternativas que se usan en la actualidad.

En el Capítulo 3, se explicará el contexto en el que se desarrolla la solución. Así como las posibles soluciones que se podrían implementar y finalmente la solución elegida.

A continuación, en el Capítulo 4, se expone la propuesta de un diseño para dar soporte a la funcionalidad descrita en el apartado anterior. Además de explicar la arquitectura y tecnología empleadas para el desarrollo.

En el Capítulo 5, se explicará como se ha llevado a cabo el proceso desde el diseño expuesto anteriormente hasta el desarrollo de la solución final. Este desarrollo realizará en un entorno de preproducción, ya que el proyecto utiliza servicios reales que pertenecen a la empresa cliente.

En el Capítulo 6 se explican los pasos que hay que seguir para ejecutar la puesta en marcha del sistema en producción y obtener los resultados que ofrece la solución final.

Posteriormente, en el Capítulo 7 se mostrarán las diferentes pruebas realizadas para verificar que el funcionamiento de la solución es la esperada. Además de analizar el consumo de recursos y la eficiencia que ofrece el sistema desarrollado.

Para finalizar, en el Capítulo 8 se expondrán las conclusiones obtenidas una vez finalizado el trabajo. En este último apartado se reflexionará sobre los conocimientos aprendidos en el trabajo y si se han cumplido los objetivos esperados que se marcaron en la Sección 1.2.

CAPÍTULO 2

Estado del arte

La integración de aplicaciones se define como «la capacidad de enlazar aplicaciones de una organización con las de otra para simplificar y automatizar procesos de la manera más amplia posible, evitando realizar cambios a las aplicaciones o estructuras de datos existentes»[1]. Una parte fundamental vendría a ser el middleware, que se encarga de proporcionar las capacidades necesarias para permitir la comunicación, la interoperabilidad y el intercambio de datos entre sistemas y aplicaciones heterogéneos.

Hoy en día, en el entorno empresarial, las organizaciones suelen utilizar una variedad de aplicaciones y sistemas para realizar diferentes funciones, como la gestión de recursos humanos, la contabilidad, la gestión de clientes, el inventario, el comercio electrónico, entre otros. Estas aplicaciones a menudo operan en silos, lo que significa que no están conectadas entre sí y no comparten información de manera eficiente. Esto puede resultar en redundancia de datos, procesos manuales, falta de visibilidad y dificultad para tomar decisiones basadas en información actualizada y precisa [1] [2].

La integración de aplicaciones aborda estos desafíos al mejorar la eficiencia operativa al automatizar procesos y reducir errores, facilita la toma de decisiones basada en datos actualizados y precisos, proporciona una experiencia del cliente fluida y personalizada, fomenta la agilidad empresarial al adaptarse rápidamente a los cambios del mercado, y promueve la innovación y la competitividad al permitir la creación de nuevas soluciones y servicios. En última instancia, la integración de aplicaciones optimiza los recursos, impulsa la productividad y ayuda

a las organizaciones a mantenerse al día en un entorno empresarial en constante evolución [1] [2] [3].

2.1 Tipos de Integración

Como se ha explicado anteriormente, la integración de aplicaciones permite la interoperabilidad entre varios sistemas con el propósito de obtener una funcionalidad unificada. En cambio, como el contexto de cada sistema puede ser completamente diferente hay que hacer un análisis previo para plantearnos que tipo de integración es más conveniente en cada caso [2] [4].

Antes de nada, habría que plantearse porque realizamos la integración e incluso si es necesaria, podría ser que se pueda desarrollar una aplicación independiente sin necesidad de que exista el middleware. Aunque esta posibilidad es bastante reducida, ya que actualmente por muy pequeña que sea la empresa, lo común es tener múltiples aplicaciones que necesitan interactuar entre ellas.

Una serie de consideraciones a tener en cuenta a la hora de elegir el tipo de integración serían las siguientes [4]:

- Acoplamiento de aplicaciones: En la integración de aplicaciones, es fundamental minimizar las dependencias entre ellas para permitir su evolución independiente sin afectar a las demás.
- Intrusión: Es importante reducir tanto los cambios en las aplicaciones como la cantidad de código de integración requerido, evitando interferir en la funcionalidad existente.
- Selección de tecnología: Diferentes técnicas de integración requieren diferentes recursos especializados, por lo que es crucial elegir la tecnología adecuada en función de las necesidades específicas de integración.
- Formato de datos: Las aplicaciones integradas deben acordar un modelo canónico para el intercambio de datos. Cambiar el formato de datos en aplicaciones existentes puede ser difícil, por lo que se puede emplear un middleware que unifique los datos de aplicaciones con formatos distintos.

- Oportunidad de los datos: La integración debe minimizar el tiempo entre el intercambio de datos, lo cual puede lograrse aumentando la frecuencia de intercambio y fragmentando los datos. Sin embargo, es importante considerar la latencia en el diseño de la integración.

- Datos o funcionalidad: Las soluciones de integración permiten el intercambio no solo de datos, sino también de funcionalidades, brindando mayor abstracción a las aplicaciones. Sin embargo, invocar funcionalidades de aplicaciones remotas puede tener consecuencias en el funcionamiento de la integración.

- Fiabilidad: Al utilizar conexiones remotas, es esencial considerar la fiabilidad de la red para evitar la pérdida de datos durante la comunicación y abordar posibles problemas de congestión.

- Comunicación remota: La llamada a un subprocedimiento remoto es más lenta que una local, por lo que es preferible invocarlos de forma asincrónica para evitar esperas innecesarias. Sin embargo, la asincronicidad añade complejidad al diseño, desarrollo y depuración de la solución.

2.1.1. Modelos de comunicación

Una vez analizadas las necesidades del entorno en el que se va a llevar a cabo la integración, estas son algunas de las posibles soluciones en lo que a modelos de comunicación se refiere:

- Integración punto a punto (*Point-to-Point Integration*): En este enfoque, las aplicaciones se conectan directamente una a una para intercambiar datos o comunicarse. Es una solución rápida y simple, pero puede volverse complicada y difícil de mantener a medida que aumenta el número de aplicaciones y conexiones [2] [5] [6].

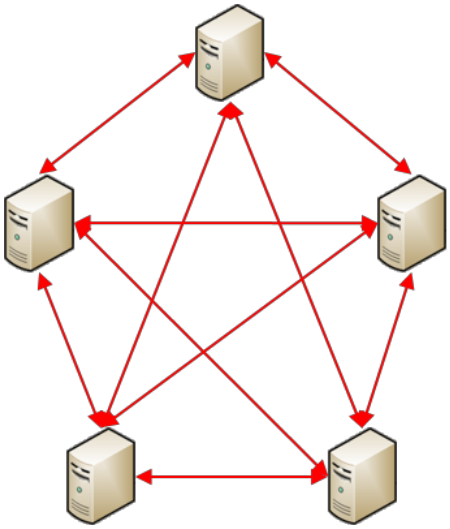

Figura 2.1: Modelo punto a punto

- Integración hub-and-spoke: Es un modelo de comunicación en el cual se utiliza una aplicación central que actúa de núcleo para facilitar la comunicación entre varios puntos de conexión. Todos los puntos de conexión se conectan directamente al núcleo y envían la información a través de él. Este coordina y dirige el flujo de información entre los puntos de conexión. Este modelo proporciona simplicidad y control centralizado en la comunicación, pero puede generar dependencia del concentrador central.

  Respecto al modelo punto a punto, ofrece muchas más escalabilidad gracias al controlador central. Por otra parte, si el núcleo sufre una caída, todo el sistema fallará [2] [5] [7].

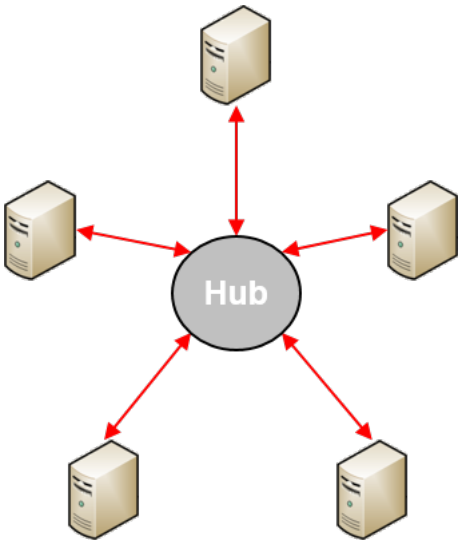

Figura 2.2: Modelo hub-and-spoke

- Integración mediante bus de servicios (*Enterprise Service Bus*): Se utiliza un bus de servicios como intermediario para permitir la comunicación entre las aplicaciones y sistemas. Cada aplicación se conecta al bus y puede enviar y recibir mensajes a través de él. Esto proporciona un enfoque más flexible y escalable, ya que las aplicaciones pueden comunicarse de forma centralizada a través del bus. También facilita la reutilización de servicios y la implementación de políticas de seguridad y monitoreo centralizado.

  El modelo ESB mejora el modelo hub-and-spoke al ofrecer flexibilidad, interoperabilidad y enrutamiento inteligente. Permite una arquitectura más escalable y adaptable, al permitir que las aplicaciones se comuniquen directamente y facilitar la integración entre sistemas heterogéneos. Además, promueve la reutilización de servicios y reduce la duplicación de esfuerzos, lo que resulta en una integración más eficiente y modular en la arquitectura empresarial [2] [5] [8].

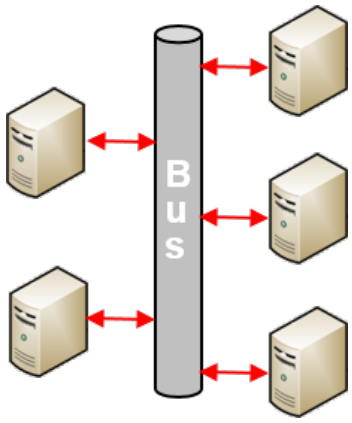

Figura 2.3: Modelo ESB

2.1.2. Tipos de comunicación

Antes de finalizar esta sección, me gustaría comentar brevemente los distintos métodos de comunicarse entre dos aplicaciones en una integración. Algunas de las más frecuentes son las siguientes [9]:

- Transferencia de archivos: permite la comunicación mediante transferencia de archivos usando el protocolo FTP (*File Transfer Protocol*) o SFTP (*Secure File Transfer Protocol*).

Figura 2.4: Comunicación por transferencia de archivos

- Base de datos compartida: mediante una base de datos compartida por todos los sistemas asegurando la transmisión de datos consistentes. Pueden existir conflictos si varias aplicaciones intentan acceder simultáneamente.

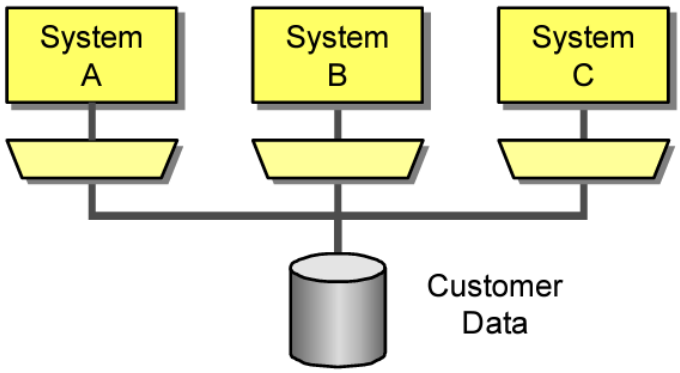

Figura 2.5: Comunicación mediante base de datos compartida

- Intercambio de datos sin procesar: intercambio directo de datos a través de protocolos de transferencia de datos en red, como los sockets TCP/IP. Estos mecanismos permiten el intercambio directo de datos entre dos sistemas en tiempo real o casi en tiempo real.

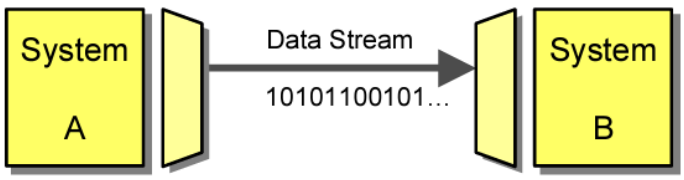

Figura 2.6: Comunicación mediante intercambio de datos sin procesar

- Llamadas de procedimiento remoto (RPC): aíslan la aplicación de los mecanismos de intercambio de datos en bruto a través de una capa adicional. Esta capa se encarga de la conversión de tipos de datos complejos en secuencias de bytes según lo requerido por la capa de transporte. Como resultado,

los mecanismos de RPC permiten que una aplicación invoque de manera transparente una función implementada en otra aplicación.

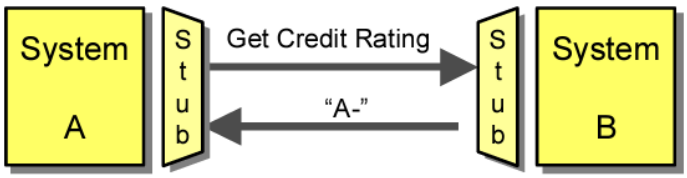

Figura 2.7: Comunicación por llamadas de procedimiento remoto (RPC)

- Mensajería: permite enviar y recibir mensajes de forma asíncrona. La gestión de los mensajes se produce mediante colas, que se encargan de almacenar los mensajes hasta que son entregados al destinatario.

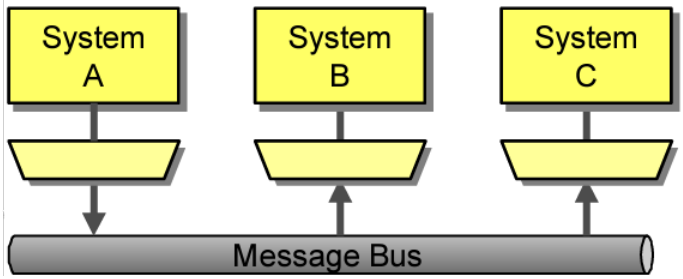

Figura 2.8: Comunicación mediante mensajería

2.2 API REST y arquitectura API-Led

En cuanto a este trabajo en particular, se va a desarrollar siguiendo la arquitectura API-Led de MuleSoft que engloba tres categorías o capas de APIs (*Application Programming Interfaces*) [10]:

- API de experiencias: reciben las peticiones de los clientes consumidores de las aplicaciones. Se trata de unificar puntos de entrada para las distintas tecnologías.
- API de procesos: encargadas de realizar la lógica de negocio y la orquestación de servicios.
- API de sistemas: encargadas de conectar con los sistemas externos dónde se almacena y obtiene la información.

En la Figura 2.9 podemos observar un ejemplo práctico de esta arquitectura con las diferentes APIs distribuidas por las tres capas [10].

Esta arquitectura nos ofrece una estructura modular, escalable y flexible para la integración de aplicaciones. Permite la exposición y el descubrimiento de servicios a través de APIs, facilita la adopción de mejores prácticas y proporciona mayor visibilidad y control sobre las integraciones. Esto ayuda a las organizaciones a desarrollar soluciones de software más ágiles, eficientes y adaptativas para satisfacer las necesidades del negocio y del mercado en constante cambio. Por tanto, se adapta perfectamente a las características que necesitamos en nuestro contexto como empresa que necesita tener el menor "Time to Market" posible y la mayor escalabilidad y flexibilidad para futuros desarrollos.

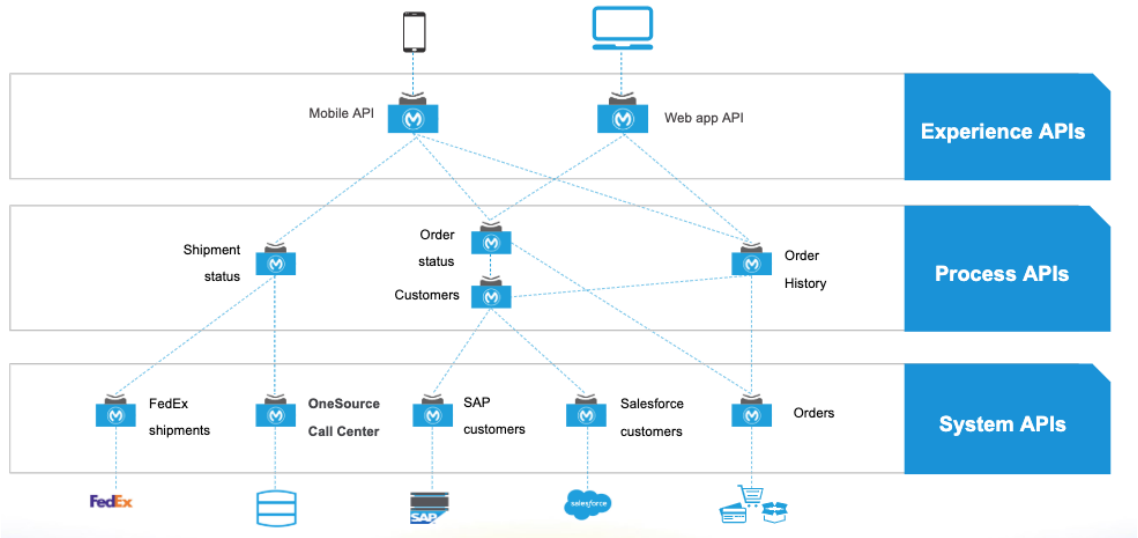

Figura 2.9: API-Led connectivity

En relación con el estilo de arquitectura elegido para estas APIs, se va a usar la arquitectura REST (*Representational State Transfer*). Es un estilo arquitectónico basado en los principios y estándares web existentes. REST utiliza el protocolo HTTP (*Hypertext Transfer Protocol*) para la comunicación y se enfoca en recursos y sus representaciones, que se acceden mediante URIs (*Uniform Resource Identifiers*). Una de las principales ventajas de REST es su simplicidad y ligereza, lo que facilita su implementación y comprensión. Además, REST utiliza formatos de datos como JSON y XML, lo que permite una fácil interoperabilidad entre diferentes sistemas. Finalmente, comparándolo con SOAP (*Simple Object Access Protocol*), REST

es más flexible, ofrece un menor acoplamiento entre el cliente y el servidor, tiene un mejor rendimiento y escalabilidad, y ha sido ampliamente adoptado en la comunidad de desarrollo. Estas ventajas hacen que REST sea una opción preferida para la implementación de servicios web en muchos escenarios [11] [12].

Para que una API sea RESTful tiene que cumplir con estos requisitos [11]:

- Arquitectura cliente-servidor: comunicación mediante el protocolo HTTP.
- Sin estado: el estado de cada sesión se almacena en el cliente, nunca en el servidor.
- Capacidad de caché: elimina el consumo de memoria al hacer comunicaciones cliente-servidor.
- Sistema en capas: permite ofrecer funcionalidades como equilibrio de carga, cachés compartidas o seguridad distribuida.
- Código de demanda: los servidores pueden extender las funciones de los clientes.
- Interfaz uniforme: interfaz genérica con el fin de simplificar y separar la arquitectura.

CAPÍTULO 3

Análisis del problema

Para lograr una integración exitosa, es crucial que la aplicación esté completamente desacoplada de los sistemas existentes del cliente. Esto implica que la aplicación no debe depender directamente de los sistemas ni estar vinculada a su lógica interna. Una opción altamente recomendable es desarrollar un middleware que actúe como intermediario entre la aplicación y los sistemas, facilitando la comunicación y la gestión de la integración.

Antes de ver las posibles soluciones tenemos que conocer el entorno de la integración. La solución a desarrollar interactúa con cinco sistemas diferentes:

- Ellis: es el almacén encargado de enviar la solicitud de actualización del estado a nuestro middleware.
- Cegid: servidor FTP al que hay que enviar un fichero con el estado actualizado.
- Saleforce Service Cloud: "back-end" al que enviamos una petición HTTP con el estado actualizado.
- Saleforce Commerce Cloud: servidor FTP al que hay que enviar un fichero con el estado actualizado.
- Salesmanago: "back-end" al que enviamos una petición HTTP con el estado actualizado.

El propósito de la solución es integrar entre sí todos los sistemas mencionados anteriormente. Tiene que poder simplificar y agilizar la comunicación entre

los sistemas involucrados, evitando cualquier acoplamiento directo entre la aplicación y estos, lo que permite una mayor flexibilidad y escalabilidad.

3.1 Especificación de requisitos

En este apartado se van a detallar los requisitos que debe de cumplir la aplicación. Son declaraciones de las funcionalidades y características que debe tener para satisfacer las necesidades y expectativas del cliente. Van a servir como guía para el diseño, desarrollo y evaluación del sistema, y nos ayudarán a garantizar que se cumplen los objetivos establecidos. Estos se dividen en funcionales y no funcionales.

3.1.1. Requisitos funcionales

Los requisitos funcionales definen "qué" debe realizar la aplicación. En cada requisito podemos observar un identificador, un título y su descripción.

RF1	Recepción de datos
Descripción	La API debe ser capaz de recibir datos entrantes desde cualquiera de los sistemas involucrados.

Tabla 3.1: RF1 - Recepción de datos

RF2	Transformación de datos
Descripción	La API debe poder realizar las transformaciones necesarias en los datos recibidos para adaptarlos al formato requerido por cada uno de los sistemas de destino.

Tabla 3.2: RF2 - Transformación de datos

RF5	Seguridad y Autenticación
Descripción	La API debe garantizar la seguridad de los datos durante su transmisión y almacenamiento. Debe implementar medidas de autenticación y autorización para asegurar que solo los sistemas y usuarios autorizados puedan acceder a los datos y realizar las operaciones correspondientes.

Tabla 3.5: RF5 - Seguridad y Autenticación

RF3	Propagación de datos
Descripción	La API debe ser capaz de propagar los datos transformados hacia los sistemas de destino correspondientes, ya sea mediante el protocolo HTTP o FTP.

Tabla 3.3: RF3 - Propagación de datos

RF4	Gestión de errores
Descripción	La API debe ser capaz de manejar errores durante el proceso de integración. Debe detectar y registrar los errores ocurridos y proporcionar respuestas adecuadas para informar sobre los errores a los sistemas de origen y destino, así como a los usuarios o aplicaciones que consuman la API.

Tabla 3.4: RF4 - Gestión de errores

3.1.2. Requisitos no funcionales

Por otra parte, en cuanto a los requisitos no funcionales, estos se centran en "cómo" debe funcionar el sistema, estableciendo criterios de calidad y rendimiento.

RNF1	Rendimiento
Descripción	La API debe ser eficiente y capaz de manejar un alto volumen de solicitudes de integración sin afectar negativamente su rendimiento. Debe tener una baja latencia y tiempo de respuesta para garantizar una experiencia fluida.

Tabla 3.6: RNF1 - Rendimiento

RNF2	Escalabilidad
Descripción	La API debe ser escalable y capaz de gestionar un crecimiento en el número de sistemas conectados y en el volumen de datos procesados. Debe poder adaptarse y escalar fácilmente para mantener un rendimiento óptimo incluso en condiciones de alta carga.

Tabla 3.7: RNF2 - Escalabilidad

RNF3	Disponibilidad
Descripción	La API debe estar disponible en todo momento, minimizando el tiempo de inactividad y asegurando que los sistemas de origen y destino puedan comunicarse de manera continua. Debe implementar mecanismos de tolerancia a fallos y de recuperación en caso de errores o interrupciones.

Tabla 3.8: RNF3 - Disponibilidad

RNF4	Monitorización y registro
Descripción	La API debe contar con capacidades de monitorización y registro para realizar un seguimiento de las solicitudes, detectar posibles problemas o anomalías, y facilitar el análisis y la solución de problemas. Debe proporcionar registros detallados de las operaciones realizadas, errores ocurridos y métricas de rendimiento.

Tabla 3.9: RNF4 - Monitorización y registro

RNF5	Documentación
Descripción	La API debe contar con una documentación clara y completa que describa: su funcionalidad, los endpoints disponibles, los formatos de datos aceptados y devueltos, y los requisitos de autenticación y seguridad. Esta documentación debe estar actualizada y ser accesible para los desarrolladores y usuarios que interactúen con la API.

Tabla 3.10: RNF5 - Documentación

3.2 Identificación y análisis de soluciones posibles

En la búsqueda de la solución más adecuada para abordar el desafío de integración planteado, se explorarán diferentes enfoques y tecnologías disponibles. Entre las posibles soluciones a considerar se encuentran los servicios web SOAP y los middleware de integración empresarial (EAI), además de las API REST que ya se han mencionado en la Sección 2.2.

Estas opciones ofrecen diferentes enfoques y funcionalidades para la integración de sistemas y cada una tiene sus propias ventajas y consideraciones a tener en cuenta. En el siguiente apartado, se analizarán en detalle las características de cada una de estas posibles soluciones, con el objetivo de tomar una decisión informada y adecuada para la integración propuesta.

3.2.1. Servicios Web SOAP

Una de las opciones posibles es utilizar una arquitectura de servicios web basada en el estándar SOAP (*Simple Object Access Protocol*). Puede ser considerado como una solución para la integración debido a sus características y ventajas. SOAP es un protocolo basado en XML que define una estructura de mensajes estructurados y formales para el intercambio de información entre sistemas.

Una de las ventajas de SOAP es su capacidad para garantizar la seguridad y la integridad de los datos mediante el uso de protocolos de seguridad como SSL *(Secure Socket Layer)* y WS-Security. Además, SOAP admite la definición de inter-

faces de servicio a través del lenguaje WSDL *(Web Services Description Language)*, lo que facilita la interoperabilidad entre diferentes plataformas y lenguajes de programación.

Sin embargo, también tiene sus inconvenientes. Puede tener una mayor complejidad en comparación con otras alternativas, lo que implica una mayor carga de desarrollo, configuración y mantenimiento. El procesamiento y análisis de mensajes SOAP basados en XML puede ser más pesado en términos de recursos y tiempo de ejecución.

Además, SOAP no se adapta tan bien a los entornos web y móviles como REST, ya que su enfoque más formal y estructurado puede resultar menos flexible en situaciones en las que se requiere una comunicación más ligera y orientada a recursos [12].

3.2.2. Middleware de integración empresarial (EAI)

Otra alternativa es utilizar un middleware de integración empresarial (EAI). Emplea un motor de integración centralizado, conocido como middleware, que se encuentra en el centro de la red y se encarga de realizar las funciones de transformación, enrutamiento y otras funcionalidades de interoperabilidad entre aplicaciones. Todas las comunicaciones entre las aplicaciones deben pasar a través de este intermediario, lo que garantiza la consistencia de los datos en toda la red [2].

Por otro lado, este enfoque presenta algunas desventajas. Al depender de un motor central, el middleware se convierte en un posible punto único de fallo en la red. Si el middleware experimenta una carga de trabajo intensa, puede convertirse en un cuello de botella para el procesamiento de los mensajes. Además, la necesidad de pasar todos los mensajes a través del middleware dificulta su uso efectivo en entornos con grandes distancias geográficas.

Otra desventaja es que las implementaciones del modelo de middleware suelen ser productos propietarios y pesados, diseñados para respaldar las tecnologías específicas de un proveedor. Esto puede ser problemático si la situación de integración abarca productos de diferentes proveedores, sistemas desarrollados

internamente y productos heredados que ya no son compatibles con el proveedor específico.

Para concluir, aunque el modelo de middleware en la EAI ofrece flexibilidad y centralización de la configuración, también presenta desafíos como la posibilidad de convertirse en un punto único de fallo y la dependencia de productos propietarios. Estos factores han llevado a la adopción creciente de arquitecturas basadas en API REST distribuidas en capas, como la arquitectura API-Led, que proporcionan una mayor flexibilidad, escalabilidad y facilidad de mantenimiento en el contexto de la integración de aplicaciones empresariales [10].

3.2.3. API REST como solución

En resumen, como ya se ha comentado en la Sección 2.2, elegir un middleware compuesto por varias APIs REST distribuidas en capas, como lo define la arquitectura API-Led, proporciona una solución flexible, escalable y modular para la integración de sistemas. Permite una total independencia entre la aplicación y los sistemas existentes del cliente, facilitando la adaptación a medida que evolucionan los sistemas. Además, las APIs REST ofrecen simplicidad y compatibilidad con los estándares web, lo que mejora la interoperabilidad y la colaboración en el proceso de integración.

Para terminar, en la Figura 3.1 se muestra un esquema de la solución, donde se puede apreciar la interacción entre el middleware y los diferentes sistemas. El middleware actúa como un intermediario, recibiendo las peticiones del almacén, posteriormente realizando transformaciones y enrutamientos necesarios, y finalmente transmitiendo la información a los sistemas correspondientes. Estas conexiones permiten una comunicación eficiente y segura entre los sistemas, facilitando la integración y la transferencia de datos de manera fiable.

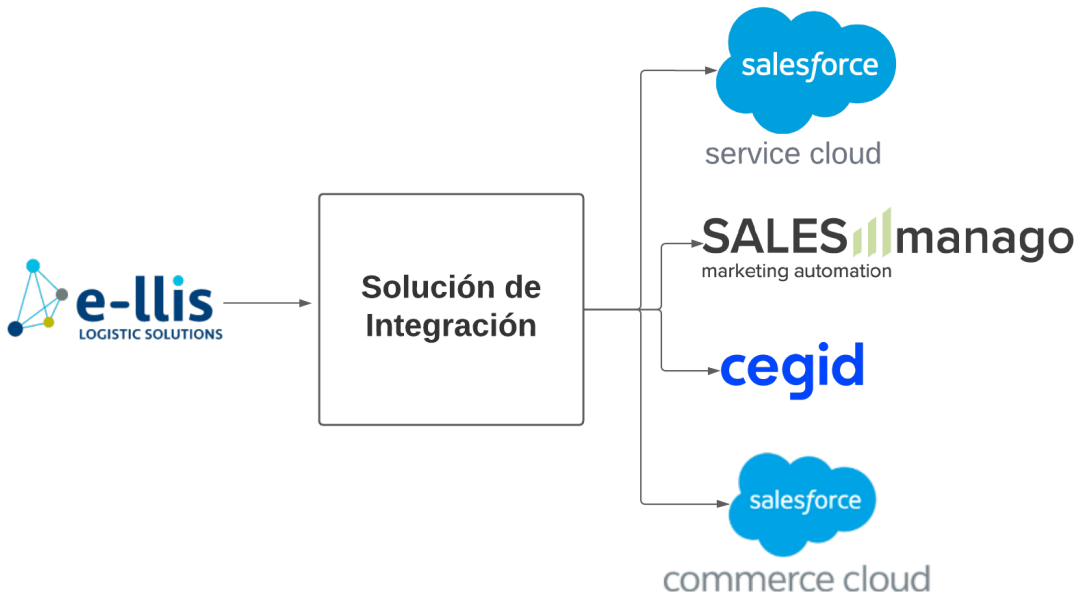

Figura 3.1: Esquema solución de la integración

3.3 Metodología y Plan de Trabajo

El proyecto ha seguido una metodología ágil, específicamente la metodología Scrum. Esta se basa en la idea de trabajar en iteraciones cortas y enfocadas, llamadas sprints, que en este caso han tenido una duración de dos semanas. Al final de cada sprint, se realiza una revisión de las tareas completadas y se definen los nuevos objetivos para el siguiente sprint.

Durante las revisiones de cada sprint, se lleva a cabo una evaluación del progreso del proyecto y se actualiza el tiempo restante para su finalización. Esto se debe a que algunas tareas pueden haber llevado más tiempo del estimado inicialmente, mientras que otras pueden haberse completado más rápidamente de lo esperado. Estas actualizaciones son importantes para tener una visión precisa del avance del proyecto y ajustar la planificación en consecuencia.

En el transcurso del proyecto, se ha utilizado una herramienta ágil como Jira para gestionar las tareas. Se ha empleado un tablero ágil para tener un orden claro de las tareas pendientes, en progreso, bloqueadas y completadas. Se puede ver un ejemplo en la Figura 3.2. Esto ha permitido mantener un seguimiento cons-

tante del progreso del proyecto y facilitar la comunicación y colaboración entre el equipo.

Las tareas del proyecto se han organizado en distintas fases o etapas. En primer lugar, se han realizado las tareas dedicadas a la especificación RAML, para definir los recursos de las APIs o el formato de los datos. Posteriormente, se han llevado a cabo las tareas de implementación, priorizando algunas sencillas como descargar el RAML desde el Design Center para que se creen los flujos de integración. Una vez tengamos la estructura de cada API clara, con sus flujos ya definidos en Anypoint Studio, se ejecutarán tareas más concretas como hacer algún mapeo. Por último, una vez hayamos terminado con las tareas de implementación, procederemos a efectuar las tareas de pruebas. Estas últimas tienen la finalidad de evaluar que cada proceso hace lo que se espera de él, con el formato definido y el tipo de respuesta esperada.

Además de estas fases principales, se han asignado y realizado tareas de soporte, ya sean para solucionar errores, mejorar el propio sistema de gestión de errores o monitorización de logs. Estas tareas han permitido abordar los problemas encontrados durante el desarrollo y garantizar un funcionamiento óptimo de la aplicación.

La metodología Scrum, junto con el uso de Jira y la división de tareas en estas fases que se han explicado, han facilitado una gestión eficiente del proyecto, permitiendo una mayor flexibilidad y adaptación a los cambios que han podido surgir. Esto ha permitido un avance constante y controlado a lo largo de todo el proceso de desarrollo.

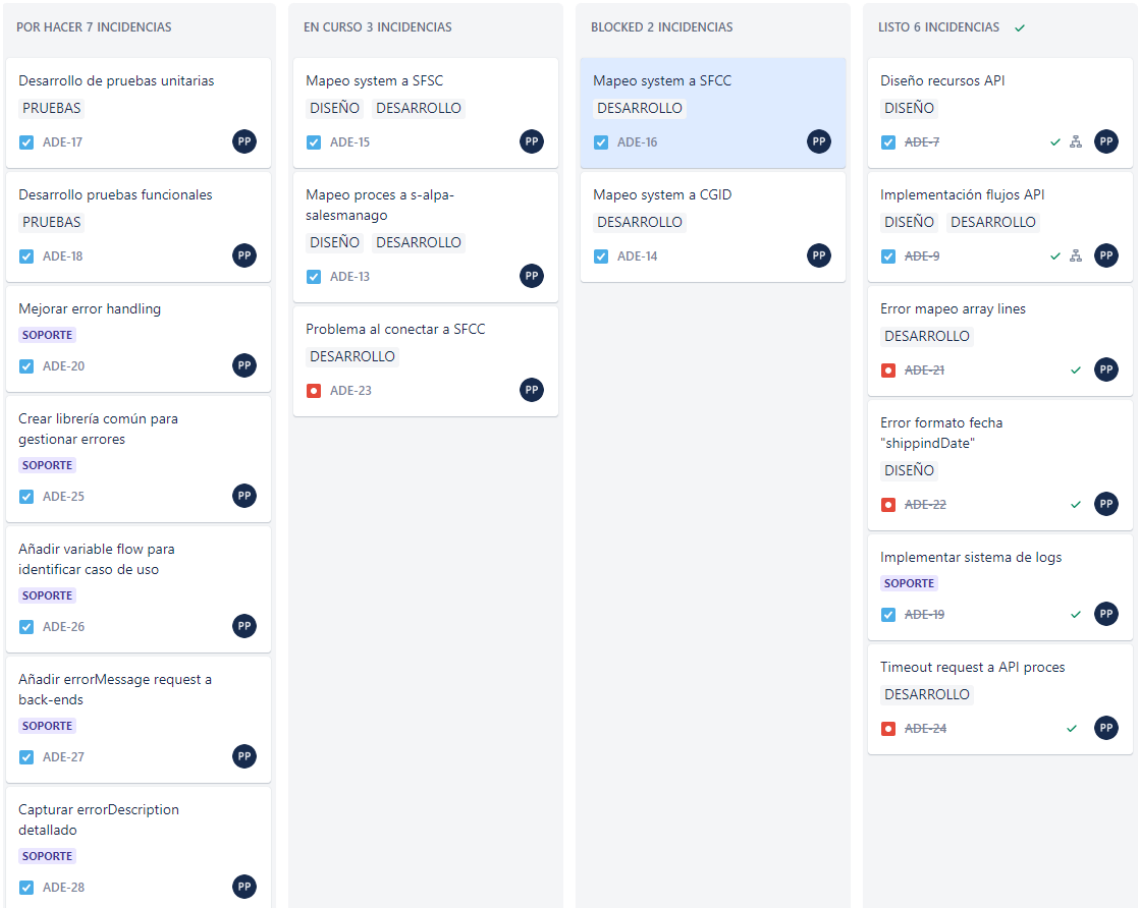

Figura 3.2: Tablero ágil

CAPÍTULO 4

Diseño de la solución

En este capítulo se va a explicar con detalle el diseño de la solución propuesta para el middleware. Como se ha explicado en la Sección 2.2, el middleware va a estar compuesto de diferentes APIs REST separadas por capas siguiendo la arquitectura API-Led.

En este caso, se van a integrar un total de seis APIs con la finalidad de desacoplar y proporcionar la máxima escalabilidad posible para futuros proyectos. Estas APIs desempeñarán roles específicos en el proceso de actualización del estado y la propagación de la información a los sistemas correspondientes.

Cada API del middleware contará con uno o más endpoints expuestos, que actuarán como puntos de acceso para recibir y responder a las solicitudes. Un endpoint es una URL específica en una API que define una funcionalidad o acción particular. Al enviar una solicitud a un endpoint, se pueden realizar operaciones específicas, como enviar datos, recuperar información o ejecutar acciones en el sistema subyacente. Los endpoints permiten la comunicación y la interacción entre diferentes sistemas o servicios a través de una interfaz estructurada y controlada.

En cuanto a la distribución por capas, en la capa de experiencia se encuentra la primera API, la cual actuará como punto de entrada para recibir la petición de actualización de estado proveniente del almacén. Esta API establecerá la comunicación con el almacén a través de HTTP, permitiendo la recepción y procesamiento de la solicitud.

En la capa de proceso, se ubicará una segunda API que será responsable de ejecutar la lógica de negocio y realizar la orquestación de servicios. Esta API gestionará las transformaciones de datos necesarias, adaptando la información recibida del almacén al formato requerido antes de ejecutar la petición a la capa de sistema.

Por último, en la capa de sistema, se implementarán cuatro APIs, una por cada sistema de destino al que se propagará la actualización del estado. Cada una de estas APIs se encargará de establecer la comunicación con el sistema correspondiente, ya sea mediante HTTP o FTP, y enviar la información actualizada de acuerdo con los requisitos específicos de cada sistema, ajustando el modelo de datos al requerido por cada "back-end".

En conjunto, estas seis APIs formarán un conjunto de servicios interconectados que permitirán la gestión eficiente de la actualización de estado y la propagación de los cambios a los sistemas relevantes. En la Figura 4.1 podemos observar un esquema de como quedaría la estructura de estas seis APIs conectadas entre sí.

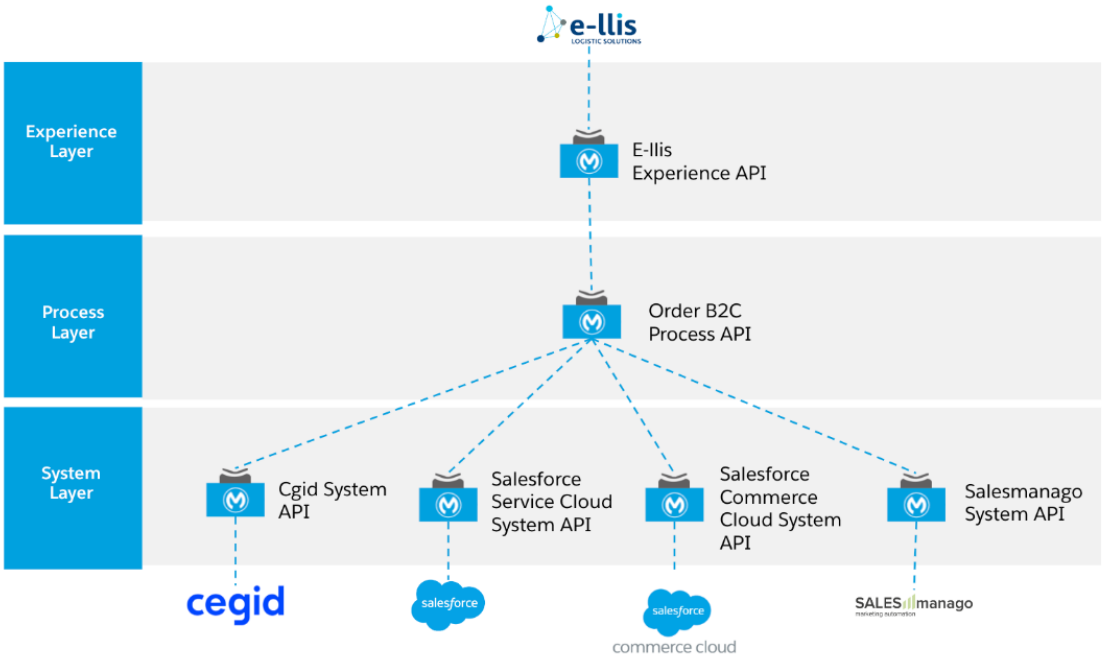

Figura 4.1: Solución diagrama API-Led

4.1 Diseño Detallado

En el siguiente apartado, se detallará la funcionalidad de los endpoints que se expondrán en cada una de las APIs del middleware propuesto. Estos endpoints serán la interfaz de comunicación con los sistemas y servicios involucrados en el proceso de actualización del estado.

Se explicará a nivel de lógica de negocio la funcionalidad de cada endpoint, mostrando un ejemplo del objeto JSON que se recibirá desde el almacén y que posteriormente serán sometidos a transformaciones de datos. También se presentarán tablas con los mapeos correspondientes que permitirán adaptar los datos según los requisitos de los sistemas de destino. De esta manera, se proporcionará una visión detallada de cómo se gestionará la información y cómo se llevarán a cabo las transformaciones para garantizar una integración efectiva y precisa entre los diferentes sistemas involucrados.

4.1.1. Capa de experiencia

Respecto a la capa de experiencia, como se ha comentado previamente, solo dispondrá de una API, debido a que simplemente necesitamos un punto de entrada para los datos que vamos a recibir del almacén.

Esta API se llamará “e-alpa-ellis” y tendrá expuesto el endpoint “/b2c/orders/status”. En este endpoint se recibirá una petición HTTP del método POST, para que el almacén nos envíe como cuerpo de la petición el pedido al que hay que actualizarle el estado. El cuerpo de la petición que recibiremos va a ser un JSON, podemos observar un ejemplo con detalle en la Figura 4.2. Si observamos el segundo atributo en la imagen proporcionada, podemos identificar el atributo “orderStatus”, el cual hace referencia al estado del pedido. Este atributo es crucial para determinar y actualizar el estado actual del pedido dentro del sistema. Este campo es de tipo enumeración y va a llegarnos con uno de estos ocho valores posibles:

- Order Accepted.
- In Progress.

- Ready To Ship.
- Shipped Order.
- Delivered To Customer.
- Return Not Delivery To Customer.
- Order Cancelled.
- Shortage.

```
#%RAML 1.0 NamedExample
{
    "orderNumber": "TEMDTR99932285",
    "orderStatus": "Order Accepted",
    "souche": "905",
    "docNumber": "14068",
    "customerId": "905000000001",
    "trackingId": "1234",
    "cc": false,
    "shippingDate": "2020-12-22",
    "city": "Madrid",
    "zipCode": "28005",
    "addressPrincipal": "PASEO VIRGEN DEL PUERTO 0",
    "addressSecondary": "Paseo de la Castellana 79",
    "currency": "EUR",
    "lines": [
        {
            "ean": "1234567898765",
            "orderQuantity": 10,
            "lineAmount": 30
        },
        {
            "ean": "1234567898765",
            "orderQuantity": 5,
            "lineAmount": 25.5
        }
    ]
}
```

Figura 4.2: Pedido que envia Ellis en formato JSON

Esta API se encargará principalmente de reenviar el cuerpo de la petición sin modificaciones a la API de proceso. Aunque pueda parecer prescindible porque no realiza transformaciones, hay que recordar porque estamos siguiendo la arquitectura de tres capas, la implementación de esta API puede servirnos para futuras integraciones con el mismo sistema, en este caso Ellis.

4.1.2. Capa de proceso

En cuanto a la capa de proceso, también dispondrá de una única API llamada "p-alpa-orderb2c". Tendrá expuesto el endpoint "/b2c/order/status?source=source" en el cual llegará una petición POST de HTTP desde la API de experiencia.

La API de proceso desempeña un papel fundamental en la orquestación del pedido hacia los diferentes sistemas de destino. Su principal objetivo es coordinar y gestionar el flujo de la información, asegurando que el pedido se comunique de manera adecuada y eficiente con los sistemas correspondientes.

En particular, antes de comunicarse con la API de sistema de Salesmanago, se ejecutarán transformaciones específicas en el cuerpo del mensaje para adaptarlo al formato requerido por dicho sistema. Estas transformaciones se basarán en el mapeo indicado en la Tabla 4.1, que establece la correspondencia entre los campos del mensaje original y los campos esperados por la API de Salesmanago.

Además, la API de proceso también establecerá una conexión con la API de sistema de CGID para obtener las credenciales necesarias asociadas a un correo electrónico específico. Estas credenciales serán utilizadas posteriormente en la petición a la API de Salesmanago, asegurando la correcta autenticación y autorización en la comunicación con dicha API.

En cuanto a la propagación de la actualización del estado hacia los otros sistemas de destino, no se realizarán modificaciones en el cuerpo del mensaje. La información será enviada tal como se recibe desde el almacén, sin llevar a cabo transformaciones adicionales.

Para terminar, quisiera comentar la utilidad de la *query param*[1] "source" que se puede observar en el endpoint de la API de proceso, cuya utilidad es diferenciar el origen de la petición a esta API. Esta funcionalidad resulta útil para futuras integraciones que puedan aprovechar esta API para realizar propagaciones a los sistemas.

[1]Una query param es un parámetro de URL que se usa para enviar datos adicionales en una solicitud HTTP. Se encuentra después del signo de interrogación "?" en la URL y se compone de un par clave-valor.

ORDER STATUS		SALESMANAGO API
orderNumber		location: www.havaianas-store.com/ ++ *orderNumber* detail18: *orderNumber*
orderStatus		detail16
souche		shopDomain
docNumber		detail5
customerId		externalId
trackingId		detail3
cc		detail4
shippingDate		date
city		detail4
zipCode		detail6
addressPrincipal		detail7
addressSecondary		detail8
currency		detail9
lines:		
	ean	products
	orderQuantity	detail11
	lineAmount	value

Tabla 4.1: Datos y mapeo para la API de sistema: s-alpa-salesmanago

Para evitar envíos innecesarios o repetidos a la API de sistema correspondiente, se hará una comparación antes de realizar la petición. Se comprobará el valor del parámetro "source", es decir, de donde proviene la petición a la API de proceso. Si se detecta que la actualización es redundante o ya ha sido efectuada previamente, se evitará enviar la petición nuevamente, lo que contribuirá a reducir la carga de trabajo y mejorar la eficiencia del proceso de integración.

4.1.3. Capa de sistema

Por último, respecto a la capa de sistema, está compuesta por un total de cuatro APIs. Todas tienen un solo endpoint expuesto menos la que conecta con CGID. Como se ha comentado en el apartado anterior la API de proceso enviará una petición a esta para obtener un email, este proceso va a estar separado mediante otro endpoint expuesto sobre el cual se darán más detalles a continuación.

Antes de entrar en detalle de cada API de sistema, en la Figura 4.3 podemos apreciar ya como quedarían todos los endpoints repartidos entre las seis APIs del middleware.

API	Endpoints
e-alpa-ellis	/b2c/order/status
p-alpa-orderb2c	/b2c/order/status?source={source}
s-alpa-sfsc	/b2c/orders/{orderNumber}/status
s-alpa-sfcc	/b2c/orders/status
s-alpa-cgid	/b2c/orders/status /b2c/orders/{order-id}/email
s-alpa-salesmanago	/orders/status

Figura 4.3: Endpoints por API

API de CGID

Como acabamos de comentar, esta API dispone de dos endpoints distintos. Por un lado, "/b2c/orders/status", para enviar el estado actualizado al "back-end" de CGID que se trata de un servidor FTP. Por otro lado, "/b2c/orders/{order-id}/email", para realizar la consulta del email que nos hace falta antes de que la API de proceso realice la petición a la API de sistema que conecta con Salesmanago.

En cuanto a este segundo endpoint, simplemente se ocupará de ejecutar una petición HTTP del método GET para obtener el email por parte del "back-end" de CGID. Para realizar la petición tendremos que indicar como *URI param*[2], cuyo nombre es "order-id", el valor del campo "orderNumber" del pedido.

Respecto al otro endpoint, cuya función es propagar la actualización del estado, se ocupará de enviar una petición de tipo POST al servidor FTP. Para poder llevar a cabo la petición correctamente deben realizarse unas transformaciones

[2]Un URI param es una parte de una URL que se utiliza para transmitir datos específicos dentro de la ruta de la dirección web. A diferencia de los query params, los URI params forman parte de la propia estructura de la URL y están precedidos por una barra diagonal "/".

de datos para adaptar el cuerpo del pedido, cuyo formato inicialmente era JSON, al formato CSV para que el servidor FTP pueda procesarlo y generar este nuevo fichero. En la Figura 4.4 se puede ver un ejemplo de como tiene que quedar el resultado de las transformaciones para generar el fichero CSV.

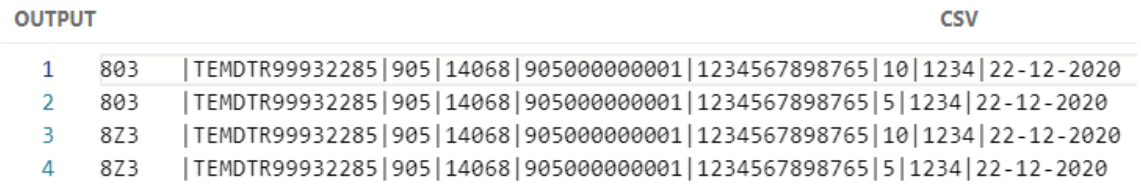

Figura 4.4: Ejemplo de fichero CSV para enviar a CGID

Como podemos observar en la Tabla 4.2 y Tabla 4.3, para obtener el resultado en formato CSV, se realizarán una serie de transformaciones en los datos.

ORDER STATUS		CGID FTP SERVER
orderNumber		OrderNumber
orderStatus = Shipped Order		[803, 8Z3]
souche		Souche
docNumber		Numero
customerId		Customer ID
trackingId		Tracking
cc		
shippingDate		Date
city		
zipCode		
addressPrincipal		
addressSecondary		
currency		
lines:		
	ean	EAN
	orderQuantity	Quantity
	lineAmount	

Tabla 4.2: Mapeo “orderStatus = Shipped Order” a CGID

ORDER STATUS		CGID FTP SERVER
orderNumber		OrderNumber
orderStatus = Order Accepted		[800, 8Z0]
souche		Souche
docNumber		Numero
customerId		
trackingId		
cc		
shippingDate		Date
city		
zipCode		
addressPrincipal		
addressSecondary		
currency		
lines:		
	ean	
	orderQuantity	
	lineAmount	

Tabla 4.3: Mapeo "orderStatus = Order Accepted" a CGID

Estas transformaciones pueden incluir el mapeo de los datos a las columnas correspondientes en el formato CSV, así como la aplicación de funciones o procesos específicos para lograr el formato deseado del archivo. Estas transformaciones garantizan que los datos se ajusten al esquema y la estructura requerida por un fichero CSV, lo que permite su procesamiento y análisis posterior. Dependiendo del estado del pedido se mapearán unos datos u otros, además si nos fijamos bien en el campo "orderStatus" podemos comprobar los valores que se le asignan dependiendo del estado para poder acoplarse al formato de datos que el servidor FTP espera recibir.

API de Saleforce Commerce Cloud

En cuanto a esta API, como se puede observar en la Figura 4.3 dispone de un endpoint llamado "/b2c/orders/status" cuya función es prácticamente idéntica a la de su homónimo en "s-alpa-cgid". Recibirá la petición POST por parte de la API de proceso y después de realizar las transformaciones necesarias para adaptar el cuerpo al formato CSV, procederá a enviarlo al servidor FTP de SFCC.

Como se puede observar en la Tabla 4.4, aunque el tipo de "back-end" al que envíamos los datos también es un servidor FTP, el mapeo difiere completamente comparado con el realizado en CGID. Cada sistema tiene su propio modelo de datos y nosotros somos los responsables de adaptarnos y moldear los datos para cada entorno.

ORDER STATUS		SFCC FTP SERVER
orderNumber		ORDERNUMBER INVOICENUMBER
orderStatus = Shipped Order		SENT FROM HEADQUARTERS
souche		
docNumber		
customerId		
trackingId		TRACKINGNUMBER
cc		
shippingDate		
city		
zipCode		
addressPrincipal		
addressSecondary		
currency		
lines:		
	ean	
	orderQuantity	
	lineAmount	

Tabla 4.4: Mapeo "orderStatus = Shipped Order" a SFCC

API de Saleforce Service Cloud

Continuando con la API de SFSC, esta dispone de un endpoint llamado "/b2c/orders/{orderNumber}/status", el cual recibirá una petición de tipo PATCH desde la capa de proceso. El *URI param* "orderNumber" hace referencia al atributo homónimo que recibimos en el cuerpo de la petición que nos llega desde Ellis, en la capa de experiencia. Antes de realizar la petición, también PATCH, para actualizar el estado al sistema, deberá transformar el cuerpo que ha recibido por parte de "p-alpa-orderb2c". En este caso la tranformación no incluye cambio de formato puesto que también hay que enviarlos en formato JSON. El mensaje que espera SFSC es bastante más sencillo que en casos anteriores, podemos ver la tabla de mapeo para un estado concreto en la Tabla 4.5.

ORDER STATUS		SFSC "back-end"
orderNumber		
orderStatus = Delivered To Customer		ACCEPTED BY THE CUSTOMER
souche		
docNumber		
customerId		
trackingId		Tracking_Number__c
cc		
shippingDate		EndDate
city		
zipCode		
addressPrincipal		
addressSecondary		
currency		
lines:		
	ean	
	orderQuantity	
	lineAmount	

Tabla 4.5: Mapeo "orderStatus = Delivered To Customer" a SFSC

API de Salesmanago

Finalmente, la última API de sistema tiene la finalidad de conectar con el "back-end" de Salesmanago. A través del endpoint "/orders/status", esta API recibirá una petición PUT proveniente de la capa de proceso. Una vez que la petición es recibida, la API simplemente reenviará el cuerpo de la petición hacia Salesmanago utilizando el método POST.

En este caso, no se realizará ninguna transformación en los datos, ya que según lo indicado en la Tabla 4.1, los datos ya han sido adaptados previamente para cumplir con el modelo requerido por la API de sistema de Salesmanago, que es el mismo que requiere el propio "back-end" de Salesmanago.

Por lo tanto, la función principal de esta API es transmitir los datos recibidos desde la capa de proceso directamente a Salesmanago, manteniendo el mismo formato de datos sin alteraciones adicionales.

En resumen, el diseño detallado del middleware propuesto ha definido las diferentes APIs separadas por capas, con sus respectivos endpoints y mapeos. Estas APIs permitirán la comunicación fluida y eficiente entre los sistemas involucrados en el proceso de actualización del estado. El diagrama de flujo de la Figura 4.5 proporciona una visión general de las peticiones y transformaciones que serán realizadas por el middleware. Este enfoque modular y escalable facilitará la adaptación del middleware a las necesidades cambiantes de integración en el entorno empresarial.

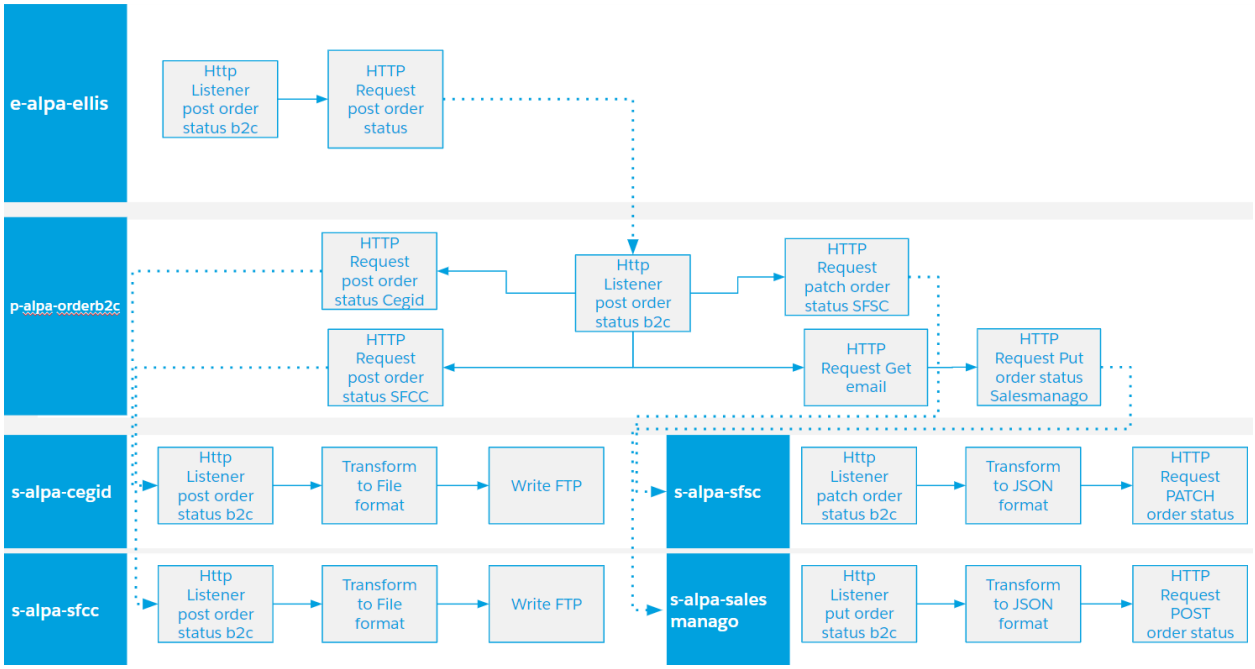

Figura 4.5: Diagrama flujo solución middleware

4.2 Anypoint Platform

En este trabajo se ha utilizado Anypoint Platform como principal herramienta para el desarrollo de la solución. Es una plataforma completa de integración y gestión de APIs proporcionada por MuleSoft. Esta plataforma ofrece una amplia gama de herramientas y servicios que facilitan el diseño, desarrollo, implementación y gestión de APIs de manera eficiente y efectiva.

Una de las herramientas clave dentro de Anypoint Platform es Design Center. Es una herramienta visual y colaborativa que se emplea para diseñar y modelar recursos de API. Utilizando el lenguaje de especificación RAML (*RESTful API Modeling Language*), los diseñadores pueden definir los recursos, métodos, parámetros y tipos de datos de una API de forma gráfica e intuitiva. Esto permite una comunicación y colaboración más fluida entre los equipos de diseño, desarrollo y negocio, asegurando una comprensión clara de los requisitos de la API desde el inicio del proceso. En la Figura 4.6 podemos ver un fragmento de un fichero RAML.

Por otro lado, también nos ofrece el "framework" Anypoint Studio. Es la herramienta de desarrollo de MuleSoft que se integra con Anypoint Platform, que

sigue la metodología iPaaS (*Integration Platform as a Service*)[13]. Una vez que los recursos de la API han sido diseñados y especificados en Design Center, Anypoint Studio permite a los desarrolladores llevar a cabo la implementación y desarrollo de la API. Con esta herramienta, los desarrolladores pueden: crear flujos de integración, realizar transformaciones de datos, definir puntos de conexión y configurar la lógica de negocio necesaria para construir una API funcional y de alta calidad.

Además de Design Center y Anypoint Studio, Anypoint Platform ofrece otras herramientas clave como Runtime Manager o API Manager, entre otras. Runtime Manager permite gestionar y supervisar los entornos de ejecución de las APIs, brindando visibilidad y control sobre su rendimiento y disponibilidad. API Manager, por otro lado, ofrece funcionalidades para la gestión completa del ciclo de vida de las APIs, incluyendo la seguridad, el control de acceso, la documentación y la medición del uso de las APIs.

En conclusión, Anypoint Platform proporciona una plataforma integral para el diseño, desarrollo, implementación y gestión de APIs. Proporciona una solución completa y robusta para las necesidades de integración y gestión de APIs, permitiendo a las organizaciones impulsar la transformación digital, mejorar la colaboración entre equipos y brindar una experiencia de API de alta calidad a sus usuarios.

```
/b2c:
  /orders:
    /status:
      post:
        is:
          - errorHandling
        body:
          application/json:
            type: !include schemas/orderb2c-status-dataType.raml
            example: !include examples/orderb2c-status-example.raml
        description: "Send information about order status"
        responses:
          200:
            body:
              application/json:
                type: !include schemas/orderb2c-status-post-200-response-dataType.raml
                example: !include examples/orderb2c-status-post-200-response-example.raml
```

Figura 4.6: Fragmento de un fichero RAML

CAPÍTULO 5

Desarrollo de la solución propuesta

En el siguiente capítulo se va a mostrar paso a paso como se ha llevado a cabo la implementación del diseño expuesto anteriormente. Para realizar el desarrollo se ha utilizado el "framework" Anypoint Studio, que como se ha comentado ya en la Sección 4.2, está completamente integrado con las herramientas que proporciona Anypoint Platform.

En Anypoint Studio, la programación se realiza mediante una interfaz visual de "Point and Click", donde los componentes y módulos se arrastran y se conectan entre sí para crear flujos de procesamiento. Estos flujos permiten que un payload, es decir, un conjunto de datos, circule a través del flujo y se realicen diversas transformaciones y operaciones en él. De esta manera, podemos diseñar de manera intuitiva y visual los pasos y lógica necesarios para procesar y transformar los datos según los requisitos del proyecto.

5.1 Implementación por capas

En esta sección se va a explicar el funcionamiento de los flujos de integración desarrollados para cada API.

5.1.1. API de experiencia

En la Figura 5.1 podemos observar el flujo principal de Ellis, todas las APIs tienen este mismo flujo principal. El primer componente es el encargado de re-

cibir las peticiones que lleguen a la interfaz que se haya determinado. El segundo trata de guardar el payload entrante en una variable para asegurarnos de no perderlo para posteriormente realizar transformaciones sobre este. Por último, el componente "APIkit Router", es el responsable de enrutar la petición al endpoint correspondiente.

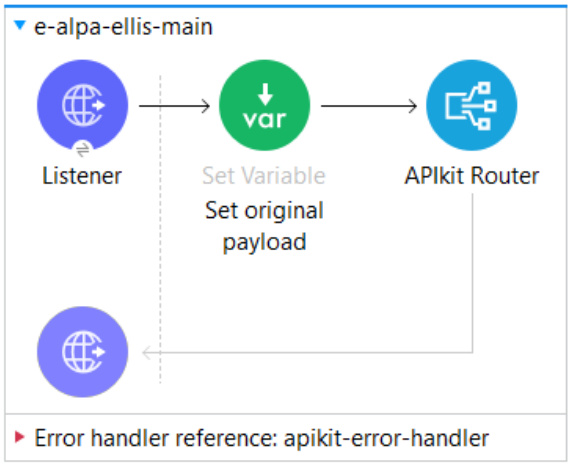

Figura 5.1: Flujo principal Ellis

Una vez enrutada la petición al respectivo endpoint, en la Figura 5.2 podemos ver el aspecto que suelen tener los flujos de cada endpoint. Como primer y último elemento siempre vamos a añadir un par de loggers para que nos indiquen el principio y final de cada enpoint. Justo después del primer logger encontraremos variables que sean importantes para el endpoint en cuestión, en este guardamos las variables flow, para identificar el caso de uso, y source, para la *query param* de la petición. Seguidamente, vemos el componente "Flow Reference" que hace que siga el flujo en un subflujo para poder reutilizar y ordenar el código. Finalmente podemos ver un "Transform Message" que se usa para realizar modificaciones sobre el payload, en este caso, formalizar la respuesta que dará al terminar la ejecución.

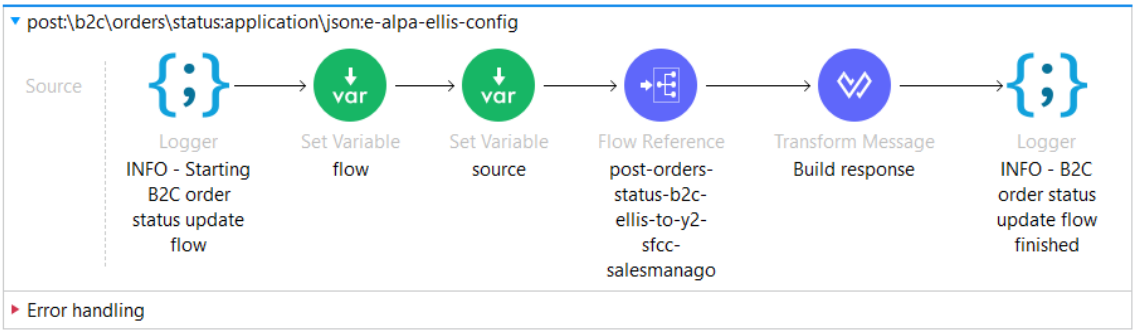

Figura 5.2: Flujo del endpoint

Para terminar con esta API, en la Figura 5.3 podemos ver el subflujo al que nos lleva el componente "Flow Reference" que hemos comentado antes. El primer elemento guarda una serie de atributos en una variable para mostrarlos en los logs, más adelante se profundizará en la implementación de los logs. Justo después tenemos un "Transform Message", que se encarga de ejecutar una función para atribuirle a cada pedido el país del cual proviene según el campo "souche". Para finalizar, vemos el componente que hace la petición a la capa de proceso, transmitiéndole el payload, que corresponde al pedido en formato JSON.

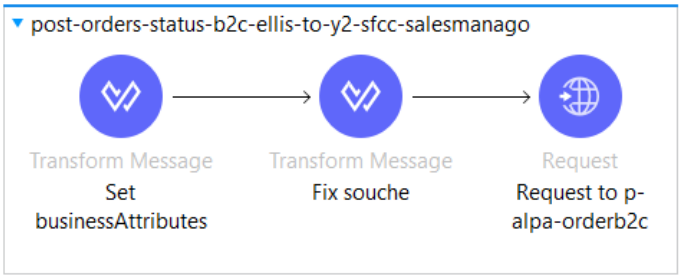

Figura 5.3: Subflujo donde se realiza la implementación

5.1.2. API de proceso

En cuanto a la API de proceso, "p-alpa-orderb2c", una vez se haya enrutado hacia el endpoint de la misma forma que hemos visto con la API de experiencia, entrará en el subflujo que se puede observar en la Figura 5.4. Para empezar, definiremos otra vez los atributos para almacenar en la variable. Seguidamente, podemos ver el componente "Scatter-Gather" dentro de un "Try", básicamente el "Scatter-Gather" permite múltiples ejecuciones en paralelo, todas ellas con el mismo payload (el objeto JSON del pedido). Al combinar estos dos elementos nos

va a permitir enviar de forma paralela el pedido a las cuatro APIs de sistema y capturar un posible error gracias al elemento "On Error Propagate" que podemos ver al final de la imagen.

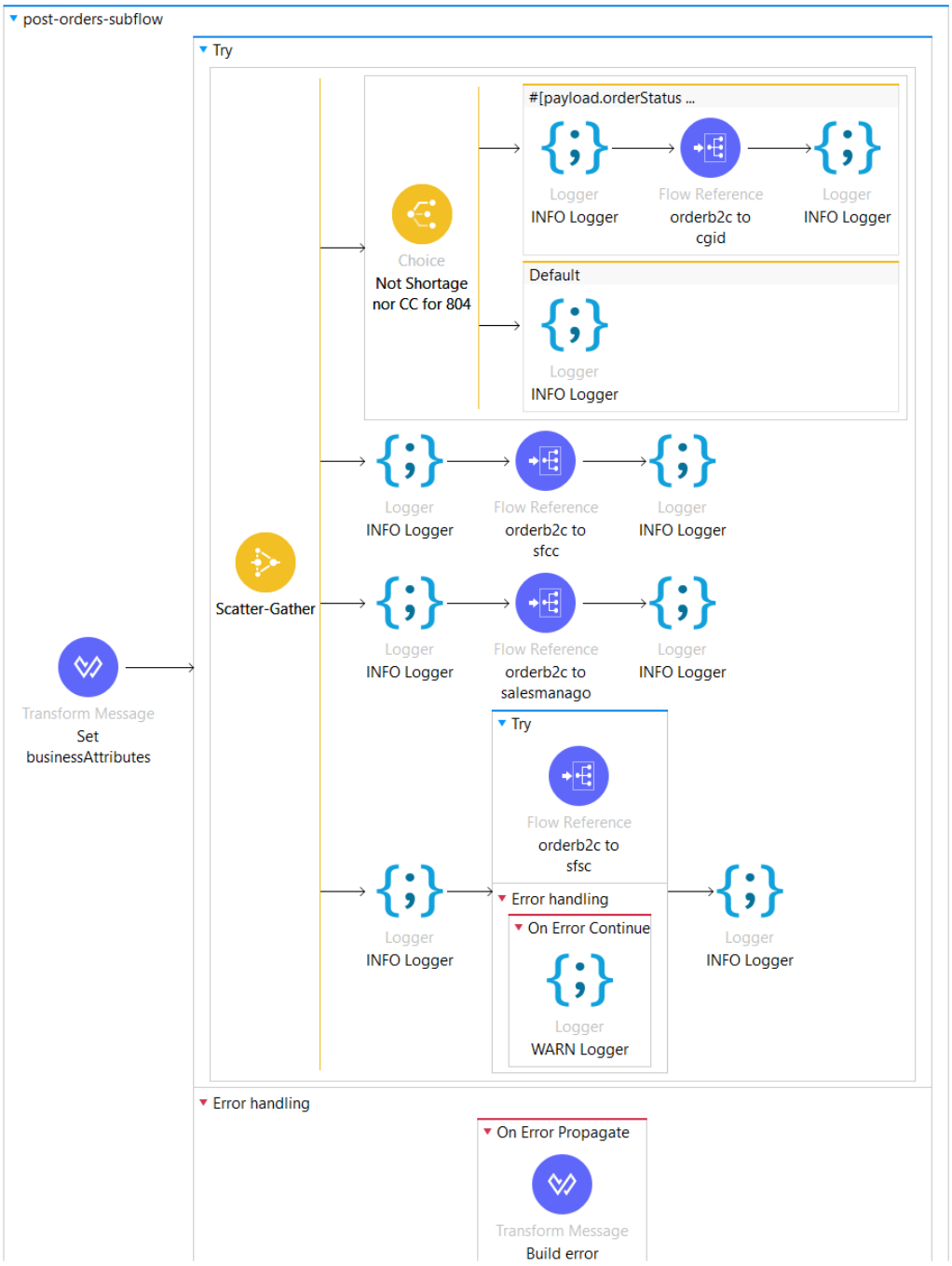

Figura 5.4: Orquestación de servicios hacia los diferentes sistemas

Antes de realizar la petición a la API de CGID, vamos a realizar una serie de comprobaciones dentro del componente "Choice", como por ejemplo asegurarnos de que la *query param* "source" no se corresponde con CGID. Esta ha sido una modificación que no se tenía prevista desde un principio pero se ha tenido que

incluir, ya que en otras implementaciones también se invoca a este flujo desde CGID, por tanto, nos aseguramos de no transmitir datos erróneos o redundantes la API de CGID. En la Figura 5.5 se puede ver el subflujo donde se realiza la petición a CGID. Los loggers que podemos ver en esta imagen solo están habilitados para DEBUG como indica su nombre, los que sí nos proporcionaran información en todo momento son los de la Figura 5.4, antes de empezar y terminar cada subflujo.

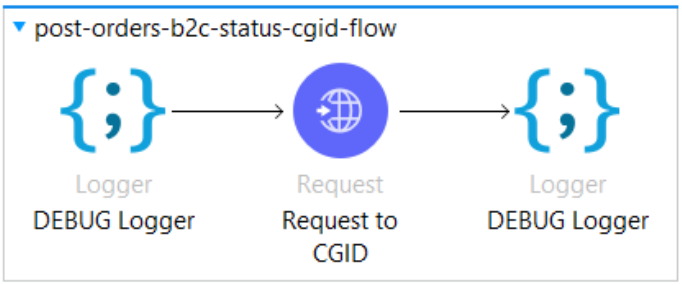

Figura 5.5: Subflujo: petición a la API de CGID

Posteriormente, en la segunda ejecución en paralelo, se efectuará la petición la API de SFCC. En la Figura 5.6 se puede ver el subflujo donde se realiza, el cual es muy sencillo, ya que tiene dos loggers a nivel de DEBUG y en medio de estos la petición hacia la API de SFCC.

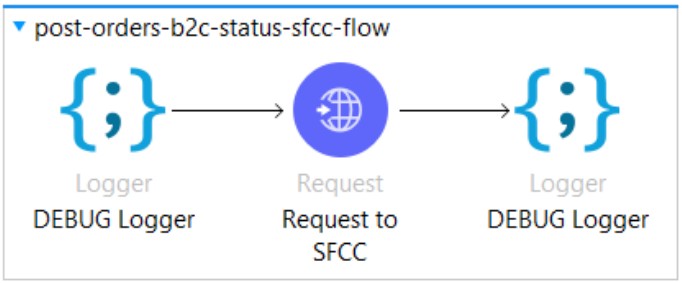

Figura 5.6: Subflujo: petición a la API de SFCC

Justo después, se encuentra la ejecución paralela para enviar el estado a la API de Salesmanago. Podemos ver el subflujo que realiza esta implementación en la Figura 5.7. Para poder ejecutarla, necesitamos obtener un email que nos proporciona CGID en el endpoint "/b2c/orders/{order-id}/email" mediante un GET. Justo antes de la petición guardaremos en una variable el payload actual para no perder el objeto JSON correspondiente al pedido. Una vez obtenido el

email procederemos a guardarlo en una variable y restablecer el payload que habíamos guardado como variable antes de la petición. Ahora se llevará a cabo la transformación de datos indicada en la Tabla 4.1 para ajustar el payload a la estructura que espera la API de Salesmanago, para finalmente realizar la petición.

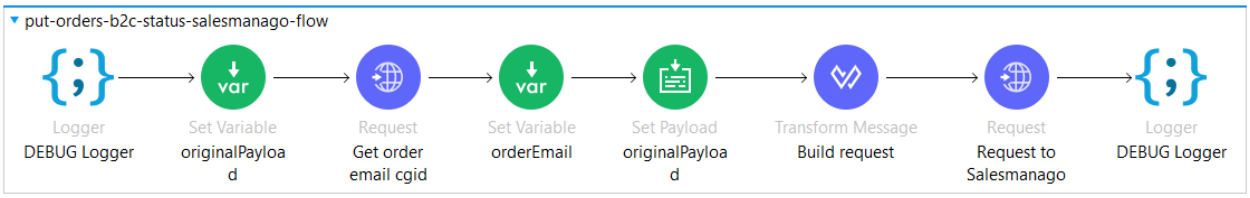

Figura 5.7: Subflujo: petición a la API de Salesmanago

Para concluir, se ejecutará el subflujo de la Figura 5.8 que se encargará de propagar el objeto hacia la API de SFSC. La llamada a este subflujo se ha encapsulado dentro de otro componente "Try", como se puede observar en la Figura 5.4, ya que estaba causando bastantes problemas la petición hacia la API de SFSC. Al introducir ahora el componente "On Error Continue", cuando esta petición genere un error ya no volverá a interrumpir las otras peticiones.

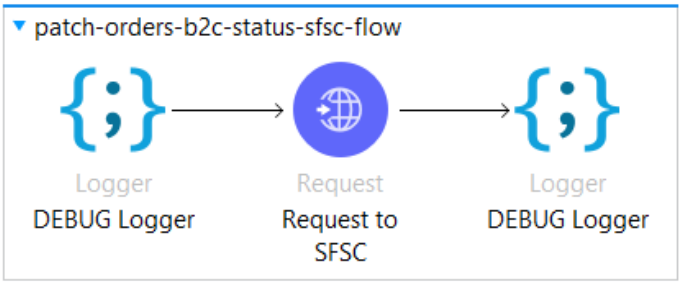

Figura 5.8: Subflujo: petición a la API de SFSC

5.1.3. APIs de sistema

Para terminar con la implementación de las APIs, divididas por capas, vamos a ver como se ha llevado a cabo el desarrollo de las cuatro APIs de sistemas que componen la solución.

s-alpa-cgid

Como se ha explicado en capítulos anteriores, esta API es la única que dispone de más de un endpoint. En la Figura 5.9 podemos ver el subflujo que realiza la implementación del endpoint "/b2c/orders/{order-id}/email". Ejecutará la petición hacia el "back-end" de CGID para obtener el email, el cual necesitamos en la capa de proceso para poder realizar el envio a "s-alpa-salesmanago". Posteriormente, mediante un "Choice", comprobaremos si hemos obtenido el email correctamente y dependiendo del resultado devolveremos una respuesta u otra.

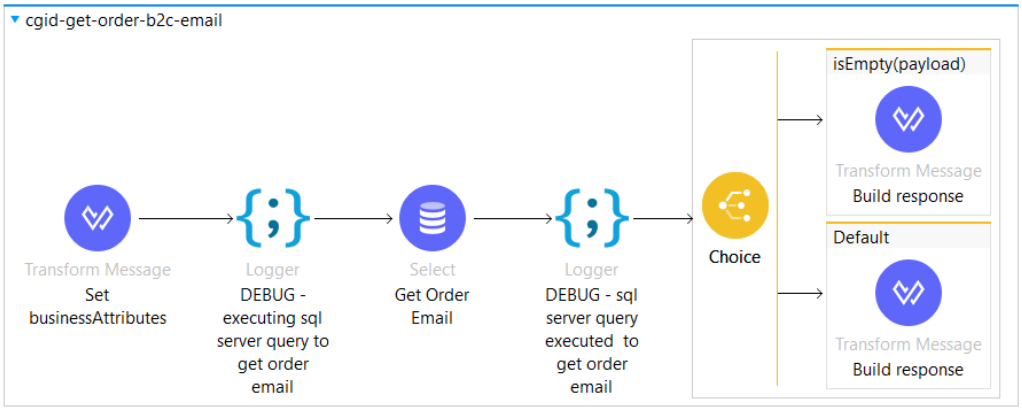

Figura 5.9: Subflujo: petición GET para obtener el email

Por otro lado, tenemos el endpoint "/b2c/orders/status", que se encarga de propagar el estado al servidor FTP. En la Figura 5.10 podemos ver el subflujo que se encarga de: indicar la ruta en la que se va a añadir el fichero y realizar una transformación como la mostrada en la Tabla 4.3, dependiendo del estado. El último componente nos llevará a otro subflujo donde se lleva a cabo la propia escritura del fichero en el servidor.

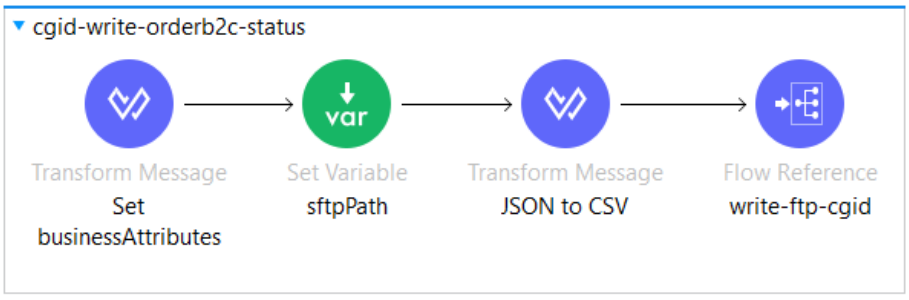

Figura 5.10: Subflujo: preparar envío del fichero

Finalmente, en este último subflujo mostrado en la Figura 5.11, se ejecutará el componente "Write SFTP" el cual recibirá como parámetros la ruta previamente establecida y el payload ya adaptado a formato CSV además de las transformaciones pertinentes.

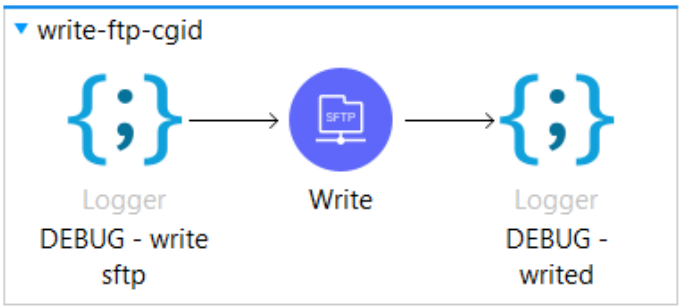

Figura 5.11: Subflujo: petición al servidor FTP de CGID

s-alpa-sfcc

En cuanto a la API de sistema que conecta con SFCC, esta dispone de un único subflujo que realiza toda la implementación, lo podemos observar en la Figura 5.12. Primero, guardamos en una variable la ruta que indica donde vamos a escribir el fichero. Seguidamente, vamos a inicializar una serie de condiciones que se van a usar justo seguido en el componente "Choice". Estas condiciones se han añadido para garantizar el correcto funcionamiento puesto que el subflujo se reutiliza en otro caso de uso dedicado a pedidos de tipo "Click and Collect". Básicamente, efectuaremos una comprobación antes de ejecutar la transformación, ya que tiene ligeras diferencias dependiendo del tipo de pedido. Una vez realizada la transformación y el mapeo correspondiente a la Tabla 4.4, dependiendo del estado, se procederá a escribir el archivo CSV en el servidor FTP de SFCC. En caso de que esta última acción falle, se capturarán los mensajes de error correspondientes.

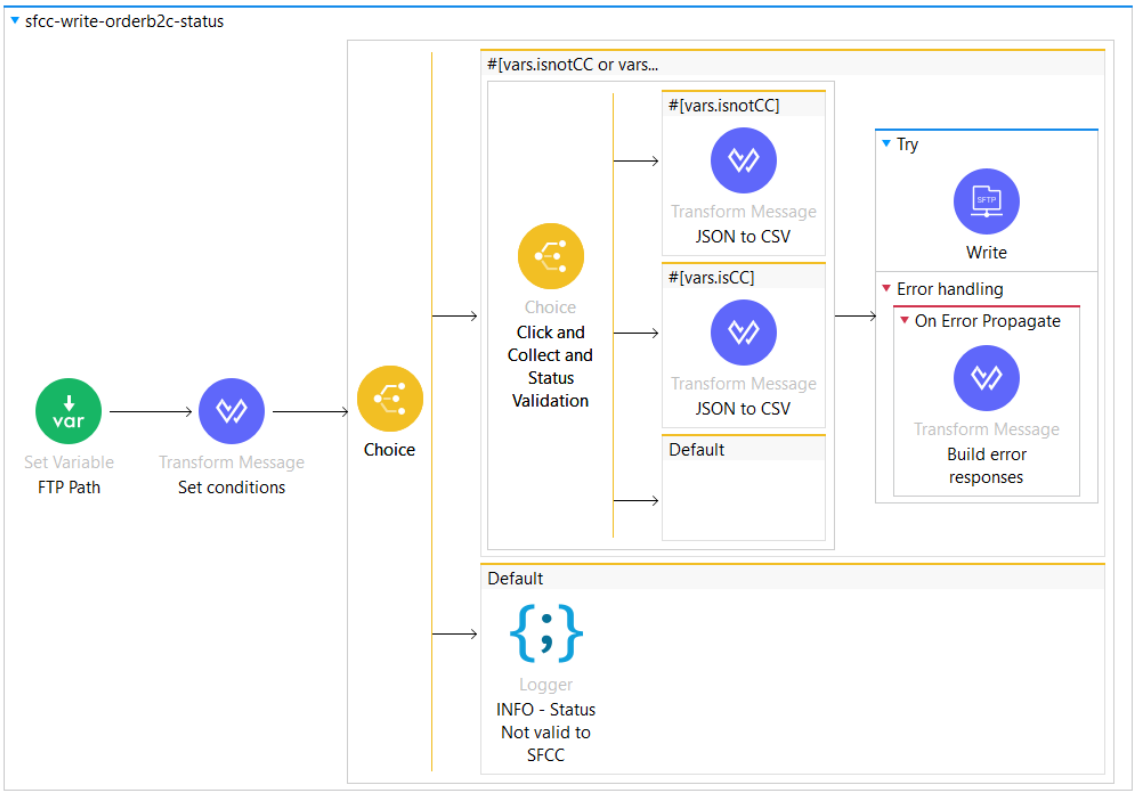

Figura 5.12: Subflujo: transformación y petición al servidor FTP de SFCC

s-alpa-sfsc

Por otro lado, la API "s-alpa-sfsc", dispone de un par de subflujos cuya función es obtener un "token" necesario para poder realizar posteriormente la petición a SFSC que actualice el estado. En la Figura 5.13 podemos observar el subflujo principal, donde el primer elemento nos conducirá a otro subflujo. El segundo llevará a cabo las transformaciones necesarias para ajustar el payload como se ha visto en la Tabla 4.5. Seguidamente, se enviará el pedido al "back-end" de SFSC, capturando los mensajes de error en caso de fallo. Y para terminar se ajustará el payload para proporcionar una respuesta.

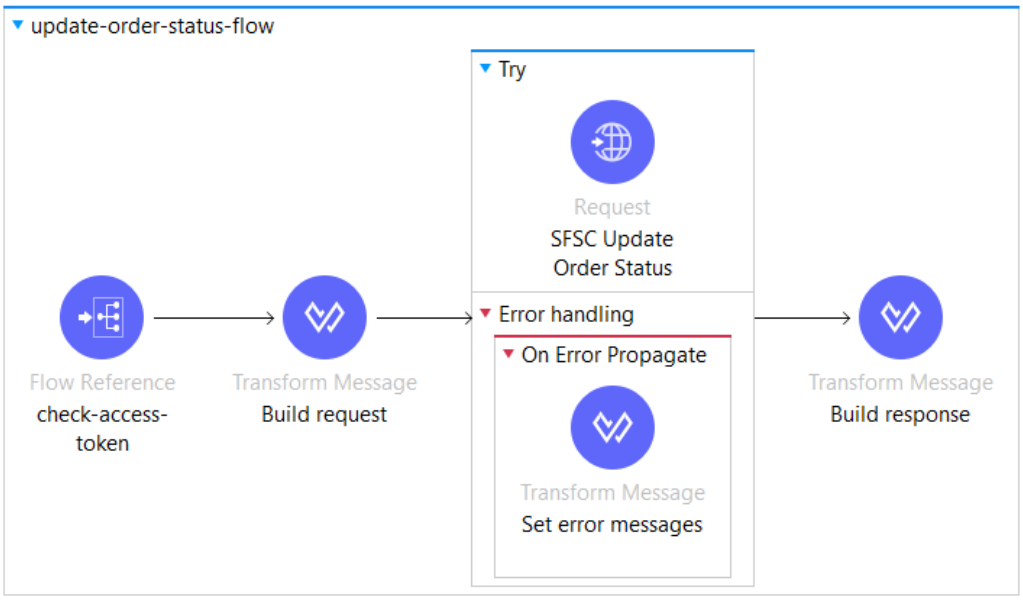

Figura 5.13: Subflujo: transformación y petición al "back-end" de SFSC

En la Figura 5.14 podemos ver el subflujo destino del componente "Flow Reference" que acabamos de ver en la Figura 5.13. En este, se intentará recuperar el "token" en caso de que ya hubiera sido generado previamente, gracias al componente "Retrieve". En caso contrario, el "Choice" entrará por la opción superior y se lanzará el subflujo que podemos observar en la Figura 5.15

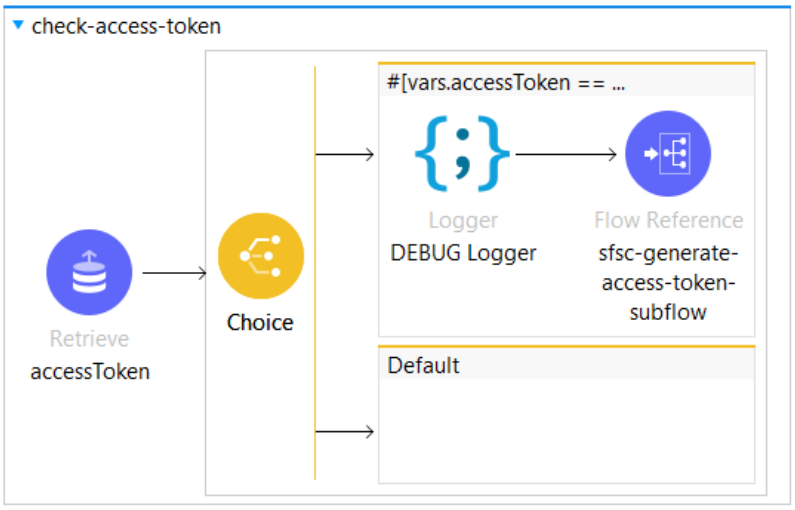

Figura 5.14: Subflujo: obtener token para autenticación

La función del subflujo de la Figura 5.15 es básicamente obtener un nuevo "token" accediendo al servidor de SFSC. Una vez realizada la petición guardaremos

el "token" en una variable y usaremos el componente "Store" para almacenar el valor de forma persistente.

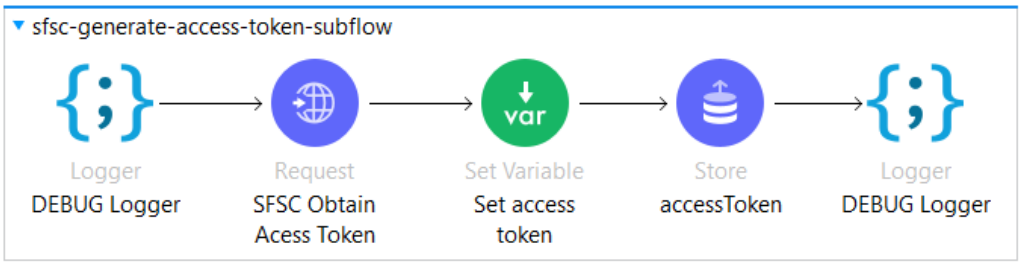

Figura 5.15: Subflujo: obtener y almacenar nuevo token

s-alpa-salesmanago

Para finalizar, la API de Salesmanago no dispone de una implementación extensa debido a que previamente, en la capa de proceso, se han realizado las transformaciones requeridas para enviar la petición al servidor de Salesmanago. Una vez enrutada la petición entrante al endpoint correspondiente, simplemente reenviará el pedido mediante una petición con el método POST hacia Salesmanago.

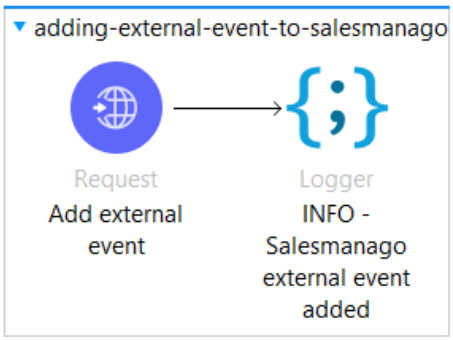

Figura 5.16: Subflujo: petición al "back-end" de Salesmanago

5.2 Implementación de errores y logs

En esta sección se va a profundizar y explicar como se ha llevado a cabo la implementación de los errores y logs que se han visto en apartados anteriores.

Para entender como funcionan los logs que se han podido observar en las imágenes comentadas previamente, podemos ver un ejemplo en la Figura 5.17 de un logger de tipo INFO. Este nos va a mostrar un determinado mensaje como pue-

de ser el inicio de un flujo o el intento de petición a otro sistema. Además, como se puede ver dentro del apartado "Content", se va a mostrar el contenido de la variable "businessAttributes" en formato JSON. Esta variable se inicializa, como hemos podido ver en imágenes anteriores, antes de realizar las implementaciones pertinentes de cada flujo, y su función principal es mostrar datos o campos relevantes para el flujo en cuestión.

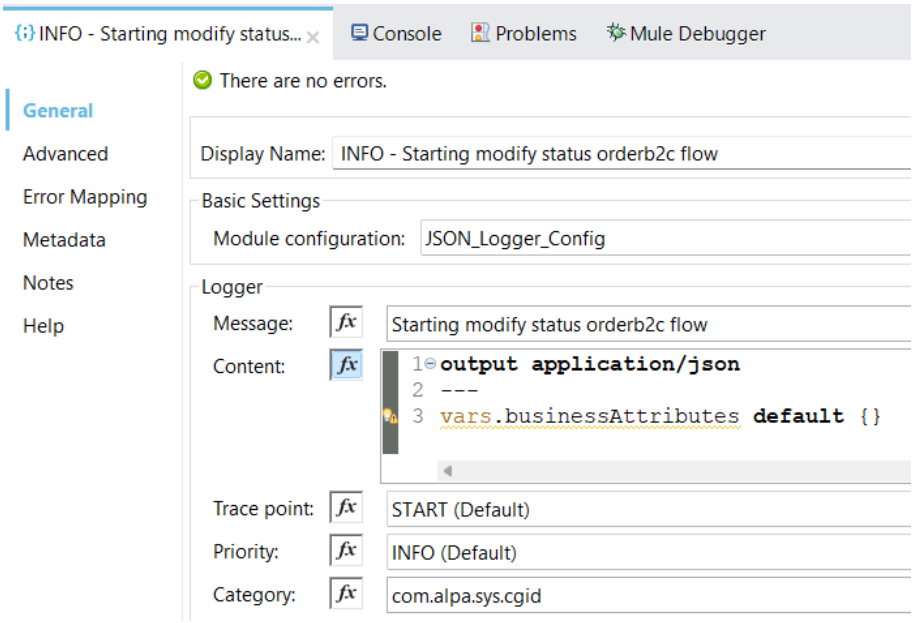

Figura 5.17: Ejemplo logger INFO

En la Figura 5.18 se puede ver un ejemplo del contenido de un componente "Transform Message", cuyo cometido es guardar un objeto JSON con diferentes campos como variable. En este caso en particular, podemos observar la variable "flow" que se ha comentado que sirve para identificar el caso de uso al cual hace referencia la integración. También, algunos campos del payload de la Figura 4.2, que se consideran importantes para poder filtrar información posteriormente en el sistema que recopila todos los logs.

Output Variable - businessAttributes

```
%dw 2.0
output application/json
---
vars.businessAttributes default {} ++ {
    flow: vars.flow,
    orderNumber: payload.orderNumber,
    orderStatus: payload.orderStatus,
    docNumber: payload.docNumber
}
```

Figura 5.18: Ejemplo businessAttributes

5.2.1. Librería común para la gestión de errores

En cuanto a la gestión de errores en el proyecto, se ha optado por realizar esta implementación en una librería común a todas las APIs. Gracias a esta libería se unifica la gestión de errores en un solo lugar, facilitando así futuras modificaciones en el código.

Para empezar, se ha añadido un controlador de errores para el flujo principal de cada API que se encarga de recibir las peticiones. Como se puede observar en la parte inferior de la Figura 5.1, estos flujos tienen un controlador de errores específico llamado “apikit-error-handler”. En la Figura 5.19 se puede ver un trozo de su implementación. Básicamente, se trata de diferentes controladores segun el tipo del error, asociado a un código de estado HTTP, el primer componente trata de guardar en una variable este mismo código. Mientras que el segundo componente hace que siga la ejecución del flujo en el subflujo de la Figura 5.21.

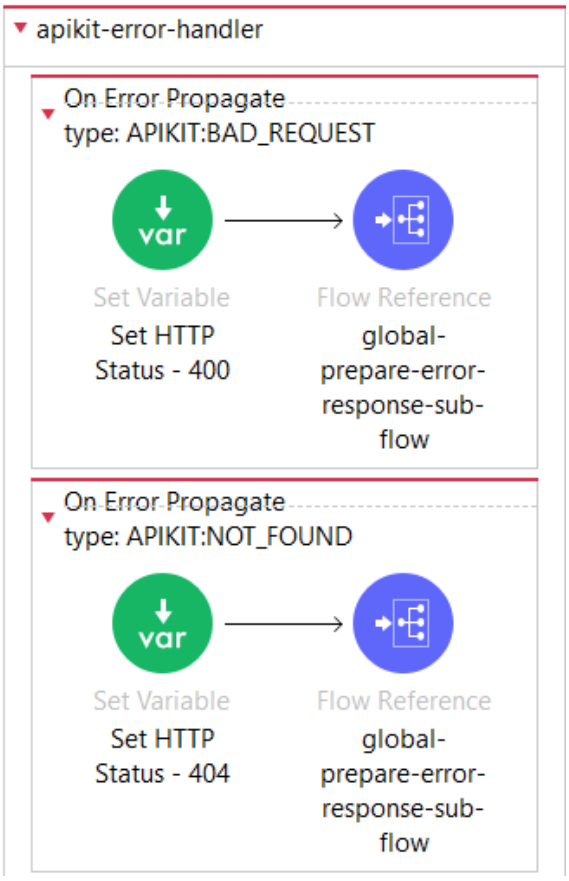

Figura 5.19: APIkit error handler

Por otro lado, el controlador de la Figura 5.20 sirve de gestor de errores para todos los otros flujos a parte del principal de cada API. Este va a enviar cualquier tipo de error hacia el subflujo de la Figura 5.21.

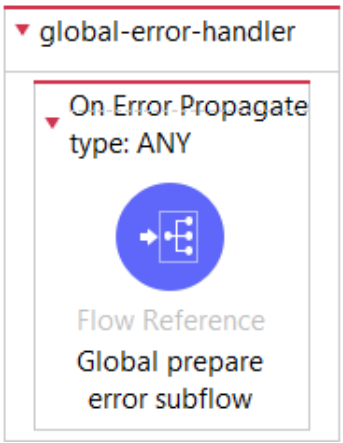

Figura 5.20: Global error handler

Una vez redirigido el flujo hacia el subflujo de la Figura 5.21, este va a guardar en una serie de variables los principales atributos que componen el mensaje de

error a transmitir. Estas son: el código de estado HTTP, un breve mensaje del error y finalmente un mensaje más descriptivo del error. Posteriormente, se llamará al subflujo de la Figura 5.22 que se encargará de construir el payload para el mensaje de error, y justo seguido generar el log de ERROR. El "Transform Message" que le sigue generará el payload a mostrar como respuesta a la petición fallida. Finalmente se publica un mensaje en un bus de mensajes asíncrono.

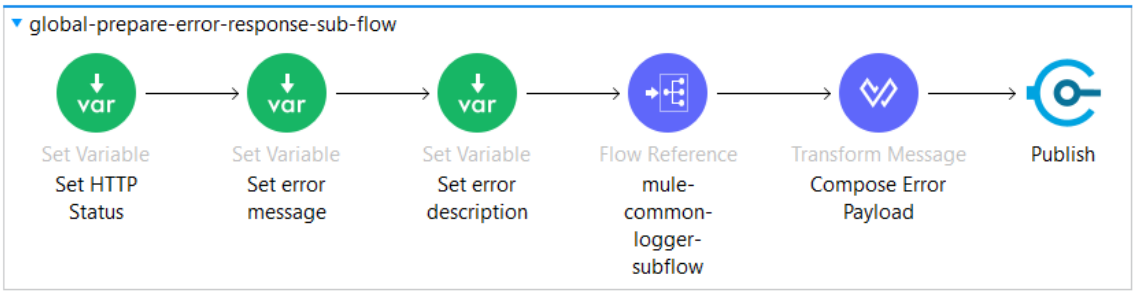

Figura 5.21: Subflow: establecer los diferentes componentes del error

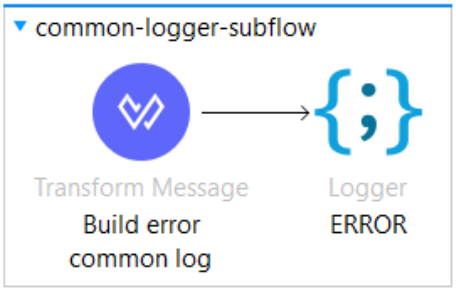

Figura 5.22: Subflow: construir mensaje de error

Este mensaje publicado se recibe en el flujo de la Figura 5.23. La función de este flujo asíncrono va a ser publicar el mensaje de error que hemos construido al sistema de logs de CloudHub, que es la plataforma donde van a estar desplegadas las APIs. Además de generarlo en CloudHub, también vamos a realizar una petición para enviar este mensaje de error a un sistema propio del cliente, donde se almacenan los logs. En caso de que alguno de estos procesos fallaran se ha hecho un controlador de errores similar al flujo mostrado en la Figura 5.21 para mostrar el error de la misma forma.

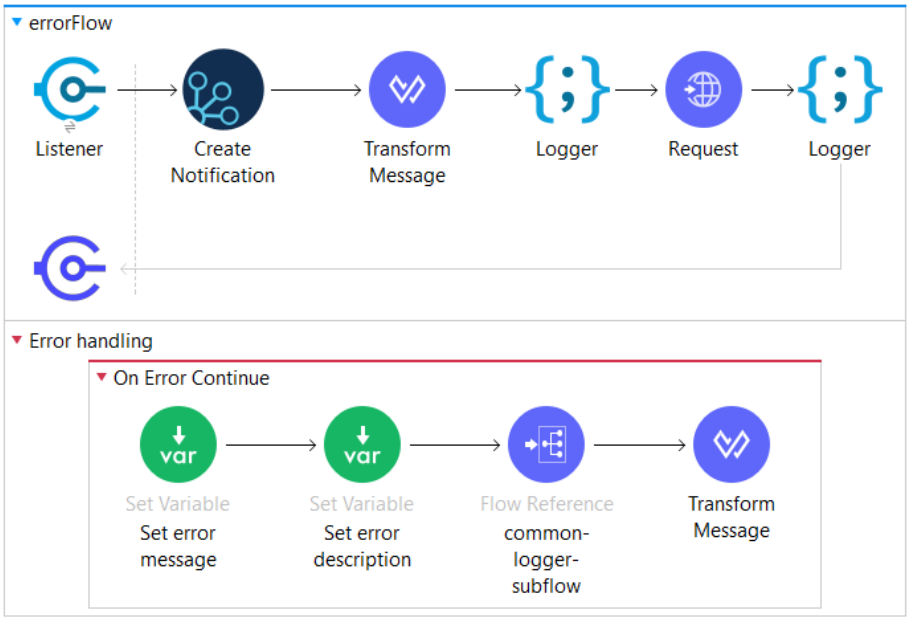

Figura 5.23: Crear notificaciones para los errores

CAPÍTULO 6

Implantación

En este capítulo se abordará la puesta en marcha de la aplicación desarrollada, donde se detallará el proceso para llevarla a producción. Para ello, se explicarán los diferentes entornos que forman parte del proyecto y su papel en el ciclo de desarrollo. Se describirán los entornos de desarrollo, test y producción, así como las configuraciones y requisitos específicos de cada uno. Además, se abordarán las mejores prácticas para desplegar la aplicación en cada uno de estos entornos, garantizando una transición sin problemas y asegurando la calidad y rendimiento de la solución final.

6.1 Funcionamiento Gitflow

Para la gestión de los diferentes entornos utilizados en el proyecto, se ha empleado la metodología Gitflow. Esta metodología proporciona una estructura clara y organizada para el flujo de trabajo con Git, permitiendo una gestión eficiente de las distintas ramas de desarrollo y facilitando la colaboración en el equipo [14].

En este proyecto en concreto existen dos ramas principales: “master” y “develop”. La rama “master” contiene el código estable y listo para ser desplegado en el entorno de producción. Por otro lado, la rama “develop” es donde se integran todas las características antes de ser probadas.

Las ramas “feature” se crean desde “develop” para desarrollar nuevas funcionalidades. Cuando se completan, se fusionan de vuelta a “develop”. Se crean

ramas “hotfix” desde “master” para corregir errores en producción. Una vez solucionados, se fusionan con ambas ramas, “master” y “develop”.

Este enfoque estructurado permite trabajar de manera ordenada, gestionar mejor las versiones y trabajar en paralelo sin interferir en el código estable. Podemos observar un ejemplo esquemático de como funciona GitFlow en la Figura 6.1.

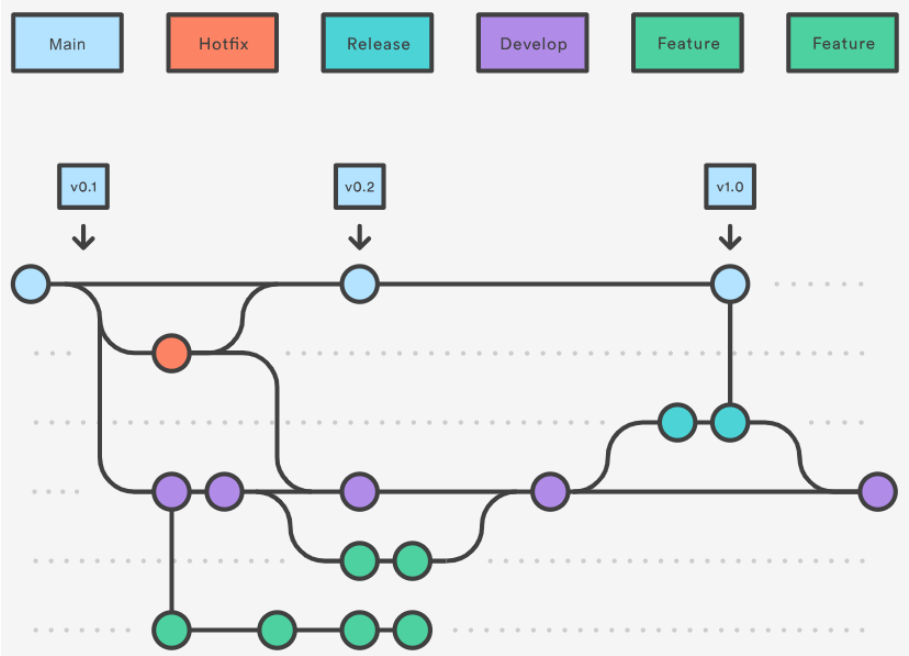

Figura 6.1: Funcionamiento Gitflow

6.2 Configuración por entorno

Para poder llevar a la práctica la metodología Gitflow, se han llevado a cabo una serie de configuraciones. Primero, para cada API, se ha creado un archivo de configuración por cada entorno, en estos archivos es donde se introducirán las direcciones de los sistemas a los que se accederán. Si nos fijamos en la Figura 6.2, también disponemos de otro archivo por entorno para propiedades securizadas. En este introduciremos valores como credenciales de autenticación para los diferentes sistemas.

Esto nos va a permitir diferenciar, según el entorno en el que nos encontremos, si queremos acceder al servidor de prueba o al que apunta a producción y seleccionar las credenciales adecuadas, según el entorno. Además de los en-

tornos “test” y “pro” que se corresponden con las ramas “develop” y “master”, disponemos de archivos para realizar configuraciones en el entorno “local”.

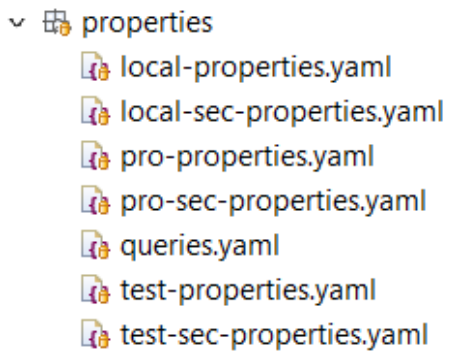

Figura 6.2: Ficheros de propiedades para cada entorno

Por otro lado, habrá que configurar cada API para que haga el uso correcto de los archivos que se han ido creando. Como esta configuración es común a todas las APIs, se ha decidido implementarla en la librería común donde implementamos la gestión de errores.

Esta dispondrá de todos aquellos elementos comunes a todas las APIs, como por ejemplo: la configuración de los loggers, la ruta de los ficheros de propiedades según el entorno, la variable donde indicamos el entorno, el algoritmo y la clave para proteger las propiedades, entre otros. En la Figura 6.3 podemos observar la lista completa de los elementos.

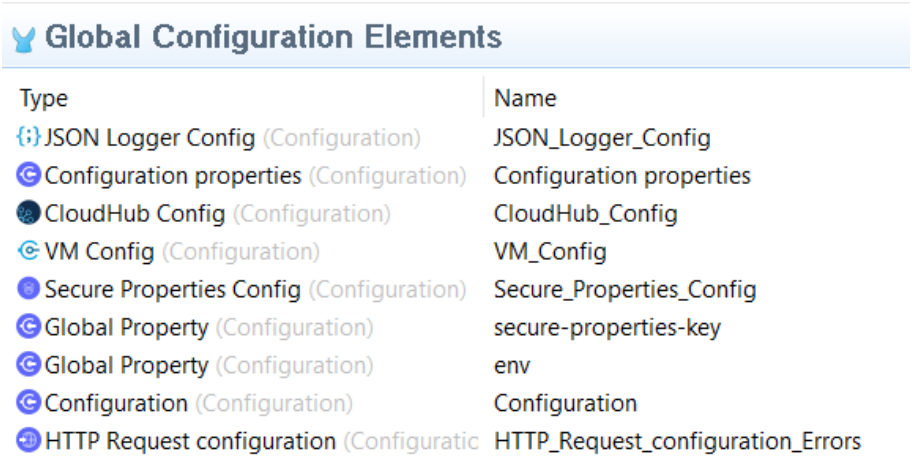

Figura 6.3: Configuración global de las APIs

6.3 Configuración de propiedades en CloudHub

Por otra parte, antes de desplegar las APIs en CloudHub también tenemos que realizar una serie de configuraciones previas. En la Figura 6.4 se pueden ver las propiedades configuradas para una API en la herramienta Runtime Manager de Anypoint Platform [15].

La primera y penúltima propiedad de la imagen son para permitir el envío de logs a la plataforma ELK, donde se recopilan y permite hacer un análisis y búsqueda de logs y eventos. La segunda y tercera son las credenciales de acceso, securizadas, para la API. Posteriormente, podemos ver una propiedad "subfix" que sirve para añadir automáticamente "-sandbox" al nombre de la API, en este caso para diferenciar que pertenece al entorno de test. También se puede ver inicializada la propiedad "env", a test, la cual servirá para que la API haga uso de las propiedades configuradas para el entorno en particular.

Figura 6.4: Configuración properties Runtime Manager

CAPÍTULO 7

Pruebas

En el proceso de desarrollo de software, existen diversos tipos de pruebas que se llevan a cabo para garantizar la calidad y el correcto funcionamiento de la aplicación.

7.1 Test de integración con Postman

Entre estos tipos de pruebas se encuentran las pruebas de integración con Postman, que se enfocan en verificar la comunicación y el funcionamiento de las APIs expuestas por los diferentes componentes del sistema. Utilizando Postman, se pueden realizar solicitudes HTTP a las APIs y comprobar las respuestas recibidas para asegurarse de que los datos se transmitan correctamente y que las respuestas sean las esperadas [16].

A continuación, se van a ir mostrando las diferentes pruebas que se han ejecutado para algunos de los endpoints expuestos. En conjunto, las peticiones son muy similares, puesto que todas hacen un POST y en el cuerpo de la petición envían el JSON de la Figura 4.2.

En primer lugar, en la Figura 7.1 podemos ver la primera prueba que se ha llevado a cabo. Esta se trata de la petición hacia la API de experiencia, simulando la petición que nos llegará desde el almacén. Como host indicaremos la URL expuesta para la API, seguidamente de la ruta base "/api/v1" y junto con el endpoint correspondiente. Finalmente, en la parte inferior de la imagen se puede apreciar el mensaje de respuesta que se configuró en la implementación de la

API, indicando que se ha recibido la actualización del estado. Además del código HTTP "200 OK", lo que nos indica que todo ha ido bien.

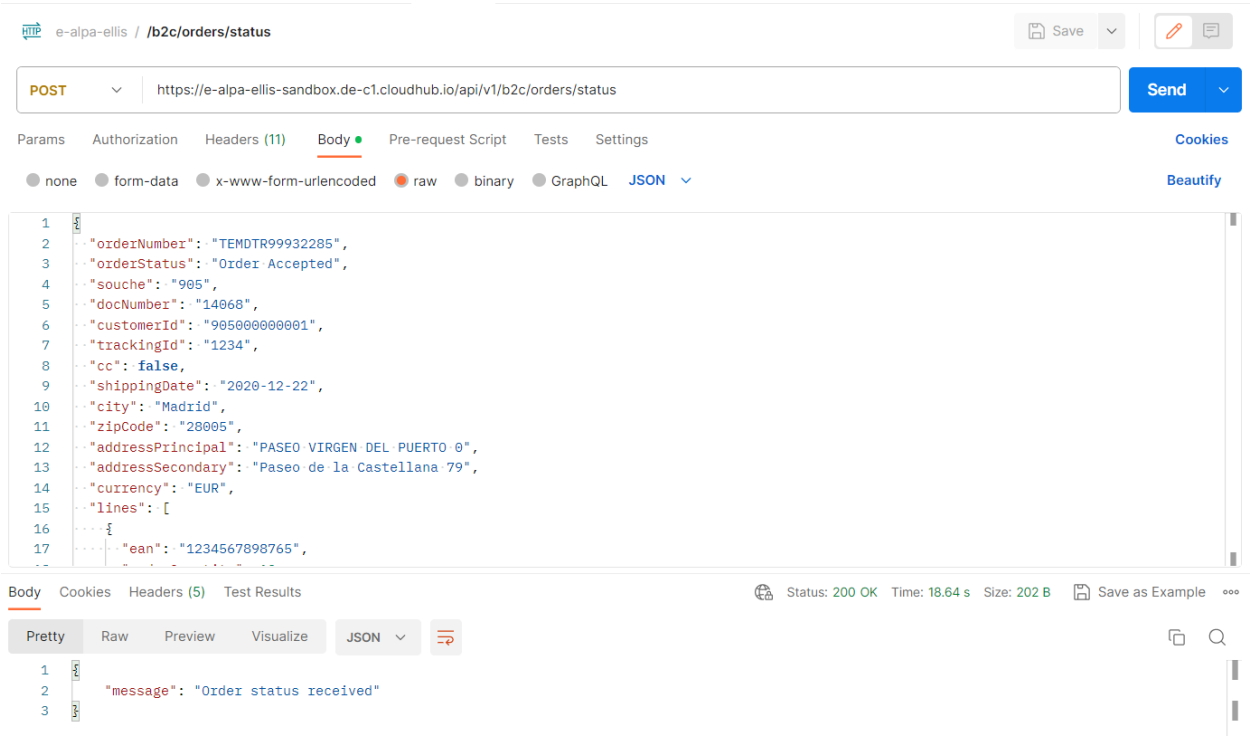

Figura 7.1: Request con Postman a la API de experiencia

En la segunda prueba de integración vamos a realizar la petición hacia la capa de proceso. Como podemos observar en la Figura 7.2, solo se requiere modificar el host, en este caso apuntando a la URL expuesta para la API "p-alpa-orderb2c". En la parte inferior podemos comprobar que la petición ha llegado a buen puerto, de la misma forma que comentábamos en la anterior prueba.

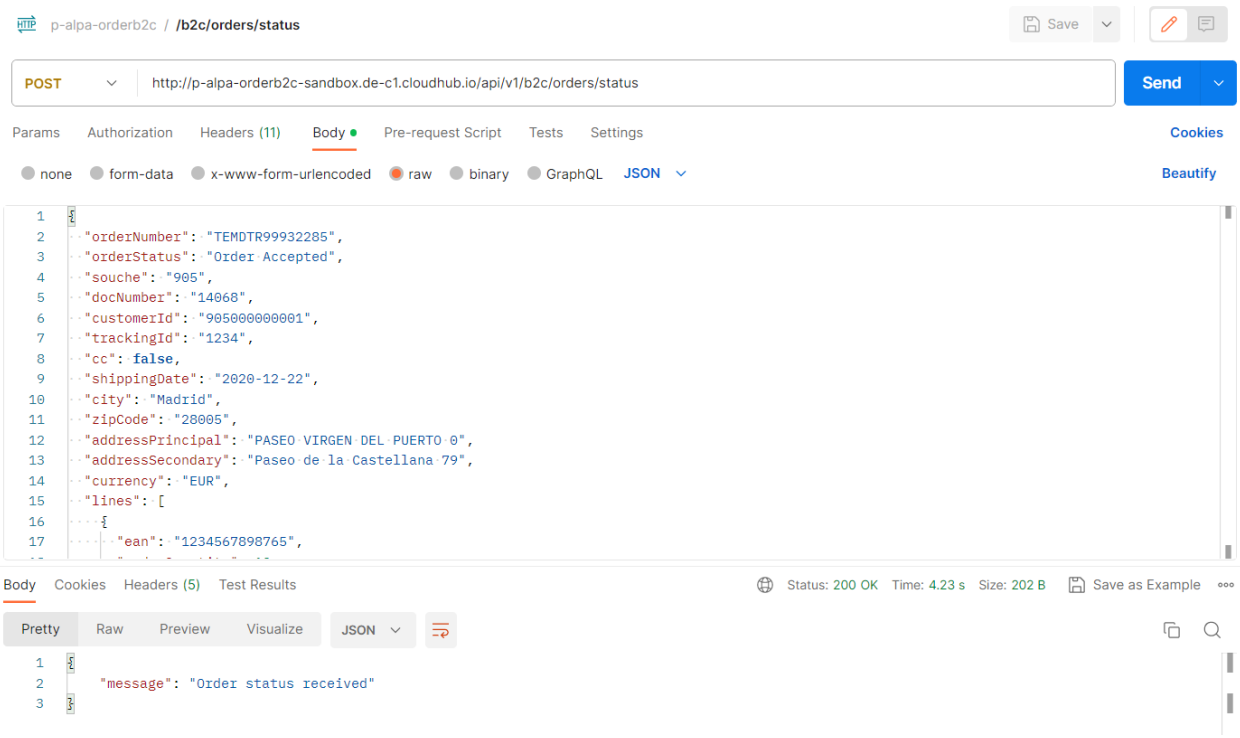

Figura 7.2: Request con Postman a la API de proceso

Finalmente, vamos a realizar una última petición directamente a una de las APIs de sistema, en este caso a "s-alpa-cgid". Como se puede observar en la Figura 7.3, esta va a seguir el mismo procedimiento que las anteriores. En la parte inferior también podemos observar que todo ha ido según lo esperado.

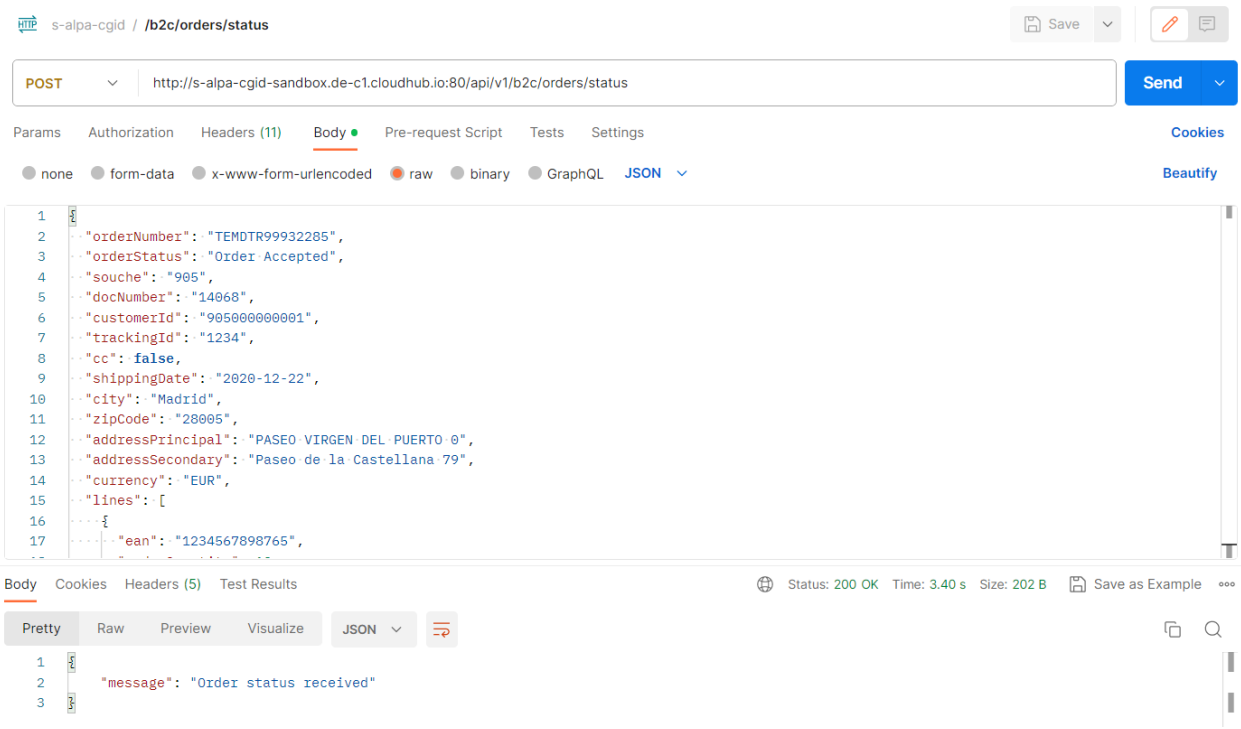

Figura 7.3: Request con Postman a la API de CGID

7.2 Test de estrés JMeter

Por último, las pruebas de estrés con JMeter son utilizadas para evaluar el rendimiento y la estabilidad del sistema bajo condiciones de carga y estrés extremas. Con JMeter, se pueden simular escenarios con alta concurrencia y evaluar cómo responde el sistema, identificando posibles cuellos de botella o problemas de rendimiento. Estas pruebas de estrés permiten determinar la capacidad y la resistencia del sistema frente a situaciones de carga intensa, asegurando su correcto funcionamiento incluso en condiciones de uso intensivo [17].

Con la finalidad de envaluar el rendimiento de las APIs que se han implementado, se han llevado a cabo una serie de pruebas para evaluar su funcionamiento con altas cargas. En la Tabla 7.1, se pueden ver algunos valores obtenidos de las diferentes pruebas que se han ido ejecutando. En conclusión, se estima que como máximo estas APIs que componen el middleware de la solución recibirían entre diez y quince peticiones por minuto, cada una. Como se observa en la tabla, las APIs cumplen de sobra con la capacidad que deben poder soportar, ya que

se realizan un total de 25 peticiones por API en una media de 1.4 peticiones por segundo, sin que se produzcan errores.

API	Requests	Average	Min	Max	Error %	Thoughput
e-alpa-ellis	25	2228	508	7257	0.00 %	1.1/sec
p-alpa-orderb2c	25	1375	435	3473	0.00 %	1.5/sec
s-alpa-cgid	25	682	271	2612	0.00 %	1.6/sec
Total	75	1428	271	7257	0.00 %	3.1/sec

Tabla 7.1: Test de estrés JMeter

CAPÍTULO 8

Conclusiones

En este trabajo, se ha desarrollado un middleware basado en una arquitectura API-Led para abordar el desafío de integrar cinco sistemas diferentes en el contexto de un proyecto de E-commerce. El objetivo principal era lograr una integración eficiente y desacoplada, permitiendo la comunicación fluida entre los sistemas y facilitando la propagación de la actualización del estado de los pedidos. A lo largo del desarrollo de la solución, se han alcanzado varios objetivos y se han superado diversos desafíos.

En primer lugar, se ha logrado implementar con éxito un middleware que cumple con los requisitos funcionales y no funcionales definidos. Se han diseñado seis APIs REST, separadas por capas, para gestionar la interacción con los sistemas implicados en la integración. Cada API ha sido implementada y probada para garantizar su correcto funcionamiento. Además, se ha seguido una metodología ágil basada en Scrum, lo que ha permitido una planificación y gestión eficiente de las tareas, asegurando una entrega oportuna y de alta calidad.

A lo largo del desarrollo, se han enfrentado algunos problemas y desafíos. Uno de los principales desafíos fue la configuración y gestión de las conexiones con los diferentes sistemas, teniendo en cuenta que utilizaban protocolos distintos como HTTP y FTP. Sin embargo, se logró superar estos obstáculos con una investigación y un enfoque cuidadoso en la implementación. También se encontraron desafíos relacionados con la transformación de los datos para adaptarlos a los requisitos de cada sistema. Mediante el uso de DataWeave y flujos de integración bien diseñados, se resolvieron estos problemas y se logró una transformación

exitosa de los datos. Además, hubo que adaptar los flujos al contexto en el que se encontraba el proyecto para que las nuevas implementaciones no generaran conflictos.

La relación con los estudios cursados ha sido fundamental en la realización de este trabajo. Los conocimientos adquiridos mediante asignaturas de desarrollo de software, arquitectura de sistemas, y tecnologías de la información han sido aplicados en el diseño e implementación del middleware. Cabe recalcar la importancia de la asignatura de cuarto curso, IAP. Esta me permitió conocer y me hizo interesarme por la integración de aplicaciones a nivel empresarial, lo que me llevó a realizar unas prácticas de empresa donde poder llevar a cabo mis propias integraciones.

En conclusión, este trabajo ha demostrado la importancia de una integración eficiente y desacoplada en proyectos de E-commerce. Mediante la implementación de un middleware basado en una arquitectura API-Led, se ha logrado una comunicación fluida entre diferentes sistemas, facilitando la propagación de la actualización del estado de los pedidos. De esta manera, se ha cumplido con éxito el desarrollo propuesto al cliente, solventando así el caso de uso que le fue planteado a Disid. En definitiva, este trabajo representa un paso importante hacia una integración más eficiente y sostenible en el ámbito del E-commerce.

Agradecimientos

En este útlimo apartado me gustaría agradecer a todas aquellas personas que considero que me han ayudado a poder finalizar mi etapa académica con éxito.

Primeramente, a todos los compañeros que he tenido durante los cinco años que he estado en la ETSINF, han conseguido que el camino sea más ameno y me llevo muchos momentos para el recuerdo.

Seguidamente, a las personas que me han ayudado a realizar este trabajo. Mis tutores: Pedro Valderas por parte de la universidad y Carlos Cabanes de las prácticas en DISID. También me gustaría mencionar a Lluís Marqués, por ayudarme cada día en las prácticas y elegir el tema para este trabajo.

Por último, agradecerle todo el esfuerzo a mi familia, sin la cual no hubiera llegado tan lejos. Y para terminar me gustaría dedicárselo especialmente a mi abuela.

Bibliografía

[1] Joan Fons i Cors y Pedro Valderas Aranda. (2022) *Tema 1: Introducción a la integración de aplicaciones.* Consultado en https://poliformat.upv.es/access/content/group/GRA_11614_2022/IAP%20-%20Tema%201.pdf

[2] MuleSoft. (n.d) *Entender la integración empresarial de aplicaciones: las ventajas del ESB para la EAI.* Consultado en https://www.mulesoft.com/es/resources/esb/enterprise-application-integration-eai-and-esb

[3] SAP Integration Suite. (n.d) *¿Qué es la integración de aplicaciones?* Consultado en https://www.sap.com/spain/products/technology-platform/integration-suite/application-integration.html

[4] Hohpe, Gregor, and Bobby Woolf. (2004) *Enterprise integration patterns: Designing, building, and deploying messaging solutions.* Addison-Wesley Professional.

[5] Priscila Renwick. (2022) *Ultimate Introduction to Salesforce Integration.* Consultado en https://www.salesforceben.com/salesforce-integration/

[6] SnapLogic. (2023) *Point-to-Point Integration: Advantages, Disadvantages, and Uses.* Consultado en https://www.snaplogic.com/blog/point-to-point-integration-advantages-disadvantages-and-uses

[7] Ellucian. (n.d) *Hubs, spokes, and point-to-point: a quick guide to common integration terminology.* Consultado en https://www.ellucian.com/blog/system-integration-techniques#:~:text=hub%2Dand%2Dspoke%20integration&text=Point%2Dto%2Dpoint%20integration%20uses,mechanics%20of%20transporting%20the%20data.

[8] IBM. (n.d) *ESB (Enterprise Service Bus).* Consultado en https://www.ibm.com/es-es/topics/esb

[9] Hohpe, Gregor, and Bobby Woolf. (2002) Enterprise integration patterns. *9th conference on pattern language of programs.* p 1-9

[10] MuleSoft. (n.d) *Tipos de API y cómo decidir cuál desarrollar.* Consultado en https://www.mulesoft.com/es/resources/api/types-of-apis

[11] Hernández, Luis Miguel Alamilla. (2021) Arquitectura REST para el desarrollo de aplicaciones web empresariales. *Revista Electrónica sobre Tecnología, Educación y Sociedad*, 8.15.

[12] Soni, Anshu, and Virender Ranga. (2019) API features individualizing of web services: REST and SOAP. *International Journal of Innovative Technology and Exploring Engineering*, 8.9: 664-671.

[13] MuleSoft. (n.d) *¿Qué es una plataforma de integración como servicio (iPaaS)?* Consultado en https://www.mulesoft.com/es/resources/cloudhub/what-is-ipaas-gartner-provides-reference-model.

[14] Atlassian. (n.d) *Flujo de trabajo de Gitflow.* Consultado en https://www.atlassian.com/es/git/tutorials/comparing-workflows/gitflow-workflow.

[15] Mulesoft. (n.d) *Manage Properties for Applications on CloudHub.* Consultado en https://docs.mulesoft.com/cloudhub-1/cloudhub-manage-props.

[16] Postman. (n.d) *Learning Postman.* Consultado en https://learning.postman.com/docs/introduction/overview/.

[17] Erline, B. (2017) *Performance Testing with JMeter 3.* Packt Publishing Ltd.

APÉNDICE A

Objetivos de desarrollo sostenible

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenible	**Alto**	**Medio**	**Bajo**	**No procede**
ODS 1. **Fin de la pobreza.**				X
ODS 2. **Hambre cero.**				X
ODS 3. **Salud y bienestar.**				X
ODS 4. **Educación de calidad.**		X		
ODS 5. **Igualdad de género.**			X	
ODS 6. **Agua limpia y saneamiento.**				X
ODS 7. **Energía asequible y no contaminante.**				X
ODS 8. **Trabajo decente y crecimiento económico.**				X
ODS 9. **Industria, innovación e infraestructuras.**	X			
ODS 10. **Reducción de las desigualdades.**			X	
ODS 11. **Ciudades y comunidades sostenibles.**				X
ODS 12. **Producción y consumo responsables.**				X
ODS 13. **Acción por el clima.**				X
ODS 14. **Vida submarina.**				X
ODS 15. **Vida de ecosistemas terrestres.**				X
ODS 16. **Paz, justicia e instituciones sólidas.**				X
ODS 17. **Alianzas para lograr objetivos.**	X			

La integración de aplicaciones y la creación de un middleware eficiente tienen un impacto significativo en el logro de varios Objetivos de Desarrollo Sostenible (ODS) establecidos por las Naciones Unidas. Al analizar la tabla con los distintos objetivos marcados y su grado de relación con este trabajo final de carrera, se pueden identificar diversas implicaciones y contribuciones que esta solución puede aportar.

En primer lugar, la creación de este middleware contribuye al objetivo de "Industria, innovación e infraestructuras". Al contar con una arquitectura API-Led, la integración de sistemas se vuelve más eficiente y ágil, lo que puede impulsar el progreso tecnológico y la innovación en la empresa. Las APIs bien diseñadas y un middleware eficiente facilitan las interacciones entre los diferentes sistemas, promoviendo la modernización de la infraestructura tecnológica y contribuyendo así al desarrollo sostenible de la industria.

Además, el enfoque colaborativo en el desarrollo de la solución también tiene una alta implicación en el objetivo de "Alianzas para lograr objetivos". La metodología Scrum y la comunicación cercana entre los equipos de diseño y desarrollo han sido fundamentales para alcanzar los objetivos del proyecto. Trabajar con herramientas como Design Center y Anypoint Studio ha permitido una mayor eficacia en el diseño y desarrollo de las APIs, fortaleciendo la colaboración entre diferentes áreas de la empresa.

Aunque el trabajo no se centra directamente en la educación, la creación de una solución de integración eficiente puede tener un impacto indirecto en el objetivo de "Educación de calidad". Al contar con una infraestructura tecnológica más fluida y una comunicación más clara entre sistemas, los empleados pueden beneficiarse de un entorno de trabajo más eficiente y productivo, lo que potencialmente mejora su formación y desempeño.

En última instancia, aunque el trabajo no aborda directamente los objetivos de "Igualdad de género" y "Reducción de las desigualdades", es importante reconocer su relevancia en cualquier proyecto de desarrollo sostenible. Estos objetivos promueven la equidad y la justicia social, y su consideración es fundamental para avanzar hacia un mundo más inclusivo y justo. Pese a que el trabajo se centra en la integración de diferentes sistemas mediante un middleware, es esencial apoyar

y concienciar sobre la importancia de estos objetivos en todas nuestras acciones para lograr un futuro más equitativo y sostenible.

En conclusión, este trabajo ha demostrado su capacidad para impactar positivamente en varios Objetivos de Desarrollo Sostenible. Desde impulsar la innovación tecnológica hasta fomentar alianzas colaborativas, la integración de sistemas se convierte en una herramienta clave para el desarrollo sostenible de la industria. Aunque el proyecto no aborda directamente la educación, la mejora en la eficiencia de los procesos puede tener un efecto indirecto en la calidad educativa de los empleados. Asimismo, se reconoce la importancia de los objetivos de igualdad de género y reducción de desigualdades, aunque no sean el enfoque central del trabajo, para avanzar hacia un mundo más inclusivo y equitativo. Con la integración de diferentes sistemas facilitada por este middleware, se sientan las bases para una infraestructura tecnológica más colaborativa y eficiente, respaldando así los esfuerzos hacia un futuro sostenible y equitativo.