UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Escuela de Ingeniería Informática

Grado en Ingeniería Informática

Desarrollo de un sistema de gestión de API´s en la plataforma Moonshot

Autor: Saulo Santana Batista

Tutores:

Jose Juan Hernández Cabrera (Tutor académico)

Iñigo Aramburu Martínez de Estíbaliz (Tutor de empresa)

Índice de ilustraciones

Ilustración 1. Descripción general de Moonshot Innovation 13
Ilustración 2. Logo Swagger 17
Ilustración 3.Overview (Web final) 25
Ilustración 4.Platform policy (Web final) 25
Ilustración 5. Documentation (Web final) 26
Ilustración 6. Federation Swagger (Web final) 26
Ilustración 7. Core Swagger (Web final) 27
Ilustración 8. Community Swagger (Web final) 27
Ilustración 9. Endpoint Swagger (Web final) 28
Ilustración 10. Todos los esquemas de eventos 29
Ilustración 11. Esquema de eventos filtrados 29
Ilustración 12. Esquema de eventos (Esquema) 30
Ilustración 13. Esquema de eventos (Ejemplo) 30
Ilustración 14. Esquema de eventos (Modelo) 31
Ilustración 15. Ejemplos de códigos (Javascript) 32
Ilustración 16. Ejemplos de códigos (Java) 32
Ilustración 17. Ejemplos de códigos (Python) 33
Ilustración 18. Logo Angular 38
Ilustración 19. Logo de TypeScript 39
Ilustración 20. Logo HTML5 40
Ilustración 21. Logo CCS3 41
Ilustración 22. Primera versión del prototipo 46
Ilustración 23. Descripción de la API (YAML) 47
Ilustración 24. Tipo, tags y path params (YAML) 48
Ilustración 25. Query params (YAML) 48
Ilustración 26. Request body (YAML) 49
Ilustración 27. Componentes (YAML) 49
Ilustración 28. Componentes con referencias (YAML) 50
Ilustración 29. Respuestas del endpoint (YAML) 50
Ilustración 30. Seguridad de tokens o contraseñas (YAML) 52
Ilustración 31. Seguridad de permisos (YAML) 52
Ilustración 32. Interfaz de Swagger generada por el YAML 53
Ilustración 33. Permisos en la interfaz de Swagger 53
Ilustración 34. Ejemplo de código Javascript en PlayCode 56
Ilustración 35. Ejemplo de código python en Programiz 57
Ilustración 36. Ejemplo de código Java en IntelliJ 58
Ilustración 37. Documentos externos (YAML) 59

Índice de tablas

Tabla 1. Diferencias entre SOAP y REST 24
Tabla 2. Planificación de las iteraciones 42

Índice

1. Resumen ejecutivo 7
2. Problema 9
2.1 Moonshot Innovation como plataforma 9
2.2 Necesidades 11
2.3 Marco teórico 13
2.4 Estado del arte 22
3. Solución 25
4. Proceso 34
4.1 Metodología 34
4.2 Herramientas y recursos utilizados 37
4.3 Iteraciones 41
4.4 Ejecución 44
5. Conclusiones 60
5.1 Resultados 60
5.2 Contribuciones 61
5.3 Experiencia personal 61
5.4 Trabajo futuro 62
6. Competencias 64
7. Bibliografía 66

1. Resumen ejecutivo

Tras el año de experiencia realizando prácticas extracurriculares en la empresa Moonshot Innovation hemos visto la oportunidad de realizar un trabajo de finde de carrera que cubra la necesidad de crear un *API Developer Portal*. Esta necesidad viene dada por los propios usuarios de la plataforma que solicitan la posibilidad de conocer cómo funcionan estas API´s y poder interactuar con ellas.

La realización de este trabajo facilitará la colaboración entre Moonshot Innovation y otras personas o empresas ajenas a esta para que desarrollen sus propios módulos dentro de la aplicación, favoreciendo así la personalización de cada ecosistema. Esta comunicación se basa en que Moonshot ofrece el uso de su API a desarrolladores externos a la empresa y por tanto esta requiere de ser documentada para el entendimiento de sus usuarios.

Es necesario que la documentación sea interactiva y sencilla para el usuario, por eso, para la realización de este trabajo se ha tenido en cuenta la opinión de usuarios reales que nos han permitido conocer su experiencia. Además, esta cuenta con un apartado de pruebas (*sandbox*) para realizar pruebas e interactuar con las API´s en tiempo real mejorando la experiencia del usuario.

También se han añadido otras dos funcionalidades que complementan a la propia documentación. En primer lugar, este proyecto cuenta con un apartado donde se describen en diferentes lenguajes de programación que código tendría que escribir un desarrollador para poder realizar una petición a las API´s. Estos fragmentos de códigos están divididos en función de los módulos que ofrece Moonshot Innovation. Los lenguajes de programación disponibles por el momento son Java, Python y Javascript. Por otro lado, también se ha desarrollado otra funcionalidad que consiste en la creación de unas plantillas que definen cual es la estructura de los objetos que tienen que enviarse en las peticiones que se mandan a las API´s, divididas por el tipo de entidad que se maneja en la plantilla. Además de la estructura de cada objeto, también se ha añadido un ejemplo para que sea más sencillo de entender e implementar.

La duración de este trabajo de fin de carrera han sido 300 horas, las cuales están divididas en las siguientes tareas: prototipo funcional, recolección de información de los endpoints, implementación de la página web, transcribir la información a Swagger, implementación de las dos funcionalidades descritas anteriormente y la realización de pruebas.

La motivación de este trabajo viene dada por la visión a futuro de la empresa, ya que documentar estas API´s será un incentivo para que desarrolladores de software ajenos a la empresa puedan desarrollar sus propios módulos de software utilizando las API´s en cuestión. Además, se incrementa el valor de negocio añadiendo un nuevo módulo disponible para todos aquellos que sean participes de un ecosistema, y aunque este módulo está principalmente enfocado para los desarrolladores de software, la idea inicial es que todas las personas que sean dueñas o formen parte de un ecosistema puedan tener estos recursos disponibles.

Por otro lado, esta motivación también viene dada por las ganas de expandir mis conocimientos y vivir el proceso de crear un proyecto real del que tendré opiniones y objeciones de usuarios reales. Desde el primer momento que entré en esta empresa mis conocimientos no han parado de aumentar y pensé que esta era una oportunidad perfecta para seguir ese camino de mejorar como desarrollador de software aumentando mis conocimientos en el área de API Management y en el desarrollo y documentación de una API´s REST.

La realización de este trabajo ha sido una gran experiencia, porque este proyecto me ha forzado a desarrollar un producto software creado y pensado para usuarios reales, donde no solo su experiencia es vital para la empresa, sino que también está sometido a la mejora continua por los propios comentarios de los usuarios que serán los encargados de validar los requisitos e identificar otros nuevos durante la vida del producto.

2. Problema

2.1 Moonshot Innovation como plataforma

Moonshot una plataforma de gestión flexible de la innovación. Gracias a esta plataforma puedes construir tu herramienta de gestión de la innovación y replicar el comportamiento de un ecosistema en un entorno digital. Esta herramienta cuenta con diferentes módulos que ayudan a la gestión del ecosistema del que tú eres dueño o formas parte. Esta es una plataforma que trabaja en tres áreas principales: descubrimiento, exploración y análisis.

Al momento de la realización de este trabajo, Moonshot cuenta con cuatro módulos principales:

Federation. Este módulo es el encargado de la gestión de usuarios y sus sesiones, las contraseñas y los permisos.

Core. Este módulo es el encargado de proveer todas las funcionalidades principales que son compartidas por el resto de los módulos, centralizando la plataforma. Asimismo, también es el módulo que se encarga de la gestión del ecosistema, es decir, los ajustes, las invitaciones etc.

Community. Dentro de este módulo se encuentra la gran parte de la lógica de negocio. Aquí se recoge todas las funcionalidades que permiten la relación entre los usuarios de la plataforma, es decir, la gestión de los muros donde se realizan las publicaciones, la gestión de la comunicación de los usuarios ya sea por mensajes de textos o videollamadas, la gestión de los miembros de la plataforma y los actores a los que pertenece etc.

Datahub. Este módulo se encarga de la gestión de los eventos generados por la arquitectura de *Event Sourcing* que utiliza Moonshot en sus API´s. Esta arquitectura será explicada en un próximo capítulo de este documento.

Para entender en su profundidad esta documentación es necesario explicar cuáles son las API´s que tiene Moonshot Innovation en el momento en el que se

desarrolló este trabajo. La plataforma cuenta con una API para cada módulo, y los endpoints que están definidos en cada una son los siguientes:

Federation. Esta API recoge todos los endpoints relacionados con los usuarios de Moonshot Innovation. Aquí se encuentran definidos los siguientes endpoints:

- Todos los endpoints de sesión de usuario.
- Todos los endpoints de modificación de permisos de usuarios.
- Todos los endpoints de creación y modificación de usuarios.
- Todos los endpoints de recuperación de contraseñas.

Core. Esta API recoge todos los endpoints relacionados con las principales funcionalidades de la plataforma y la gestión del propio ecosistema. Aquí se encuentran definidos los siguientes endpoints:

- Todos los endpoints de creación y obtención de taxonomías.
- Todos los endpoints de obtención, creación, modificación y ajustes del ecosistema.
- Todos los endpoints de obtención y creación de notificaciones e invitaciones.

Community. Esta API recoge todos los endpoints relacionados con desarrollo de una comunidad y la comunicación entre sus usuarios. Aquí se encuentran definidos los siguientes endpoints:

- Todos los endpoints de obtención, creación y modificación de eventos (eventos online y eventos presenciales).
- Todos los endpoints de obtención, creación y modificación de las salas (salas públicas, chats directos o salas creadas para grupos)
- Todas los endpoints de obtención, creación y modificación de las publicaciones creadas en el muro.
- Todos los endpoints de obtención, creación y modificación de los actores.
- Todos los endpoints de obtención, creación y modificación de los miembros.

- Todos los endpoints de obtención, creación y modificación de los proyectos

Datahub. Esta API recoge todos los endpoints relacionados con la gestión de eventos. Aquí se encuentran definidos los siguientes endpoints:

- Todos los endpoints de creación de mensajes/eventos (mensajes/eventos generados para Moonshot).

2.2 Necesidades

Moonshot Innovation es una plataforma de la gestión de la innovación. Definida de esta manera ya que su objetivo es prestar un servicio software que permita trabajar con diversas tipologías de datos para poder procesarlos, analizarlos y obtener información que aporte algún valor. A diferencia de una solución software, donde se desarrolla un producto software específico, Moonshot tiene creado un producto que no está centrado en la parte de las operaciones, sino en la interconexión entre los productos de la propia empresa, disponiendo cada uno de inteligencia y comunicación entre sí. También busca eliminar los desarrollos individuales facilitando a sus usuarios ciertas herramientas para que estos puedan realizar aportaciones a sus ecosistemas, pudiendo ser personalizables y compartibles a otros ecosistemas. Además, también se caracteriza por permitir que los usuarios puedan realizar carga y descarga de datos dentro de esta.

Actualmente, Moonshot cuenta con bastantes servicios que cumplen satisfactoriamente la definición de plataforma descrita anteriormente, como la carga de usuarios a través de un archivo .csv, o la integración de una funcionalidad creada por un tercero para la creación y gestión de retos. Es por ello por lo que se ha encontrado la necesidad de realizar un *API Developer Portal*, ya que aporta una gran valor a la empresa y permite su crecimiento dentro del mercado. La creación e integración de este proyecto supone una serie de ventajas a la empresa:

- Permite competir con otras empresas ofreciendo una funcionalidad que es indispensable en una plataforma que quiera captar clientes que desarrollen software.
- Facilita la integración de nuevos desarrolladores a la empresa. Cuando un programador comienza sus primeros días en una empresa, necesita conocer la lógica del negocio, ver cómo están estructurados los proyectos en los que va a trabajar y conocer el funcionamiento de los mismos. En Moonshot no existe ningún tipo de documentación por lo que alguien que quiera desarrollar código usando estas API´s tendrá disponible la documentación para conocer su uso en profundidad.
- Evita pérdidas de tiempo de analizar el código para saber cómo realizar una petición a una API, ya que es posible que un programador quiera hacer uso de endpoints que no han sido desarrollados por él o fueron desarrollados por él y ha pasado mucho tiempo. En ese caso, es una ventaja el tener una documentación disponible para resolver las dudas del trabajador.
- Abre la posibilidad de creación de futuras herramientas para integrar dentro de Moonshot Innovation, como por ejemplo, la creación de una tienda que permita la publicación de módulos creados por desarrolladores externos a la empresa y que estos módulos puedan ser integrados en todos los ecosistemas gracias a se rigen por las mismas API´s. Además, en un futuro se podría identificar cuáles de las peticiones que reciben las API´s son generadas por la aplicación web y cuáles son generadas por el API Portal, pudiendo utilizar esos datos para la creación de *dashboards*, que nos permitan la visualización de los datos en gráficas sobre el uso de los diferentes módulos para su posterior estudio y beneficio de la empresa.

En la siguiente imagen, se puede observar cuál es la estructura de Moonshot clasificando los módulos de la aplicación.

Ilustración 1. Descripción general de Moonshot Innovation

En esta imagen se puede observar que el *API Developer Portal* se encuentra en el apartado del desarrollo de módulos para la plataforma, ya que la finalidad de este proyecto es que otros desarrolladores puedan crear sus propios módulos. Dando posibilidad a la creación de la Moonshot store.

2.3 Marco teórico

Previo a la realización de este trabajo de fin de grado, Moonshot Innovación no contaba con ningún tipo de documentación para ninguna de las API´s. Es por ello por lo que se ha tenido que realizar un estudio de cómo llevar a cabo la gestión de una API aplicando buenas prácticas. Para esto ha sido necesario entender conceptos como "*API Management*", "*API REST*" y tecnologías como "*Swagger*" o "*Angular*" que serán explicadas a continuación.

2.3.1 API REST

Una API (interfaz del programa de aplicación) es un conjunto de reglas que permite que diferentes programas se comuniquen entre sí. Describe la manera apropiada para que un desarrollador de software componga un programa en un servidor que se comunica con varias aplicaciones cliente.

Por lo tanto, una API REST es la interfaz de programación de aplicaciones de transferencia de estado representacional. Es decir, cuando se llama a una API REST, el servidor devuelve al cliente una representación del estado de los recursos solicitados al sistema del cliente.

Esta tecnología utiliza el protocolo HTTP para obtener datos o realizar operaciones en varios formatos de datos (como XML y JSON), permitiendo la ejecución de los procesos de manera más rápida.

REST determina la estructura de un API, es decir, los desarrolladores se obligan a cumplir un conjunto específico de reglas a la hora de diseñar una API.

Para poder realizar una solicitud a una API REST, es necesario conocer la URI del recurso que queremos acceder, modificar o borrar. La diferencia que existe entre una URL y una URI es que esta primera es un localizador uniforme de recursos y consigue que cada página web tenga su dirección propia y concreta, en cambio una URI está formada por una URL y un URN, siendo este último un nombre estático del recurso al que queremos acceder.

Una API REST desglosa una transacción para generar una secuencia de pequeños componentes. Cada componente aborda un aspecto fundamental específico de una transacción. Esta modularidad lo convierte en un enfoque de desarrollo flexible.

Una API REST aprovecha los métodos HTTP descritos por Protocolo RFC 2616. Este utiliza las siguientes solicitudes HTTP:

- **GET**. Solicita buscar datos
- **PUT**. Solicita alterar el estado de los datos (como un objeto, archivo o bloque)
- **POST**. Solicita crear datos
- **DELETE**. Solicita la eliminación de datos

El servicio REST no puede retener nada entre ejecuciones, lo que lo hace beneficioso en aplicaciones en la nube, es por ello por lo que se define como un servicio sin estado.

A continuación, se presentan las ventajas más importantes de las API´s REST.

Escalabilidad. REST API ofrece una excelente escalabilidad. Como los clientes y los servidores están separados, un equipo de desarrolladores puede escalar un producto sin muchos problemas.

Además, es más fácil integrar REST con los sitios actuales sin refactorizar la infraestructura del sitio web. Esto permite a los desarrolladores trabajar más rápido en lugar de perder tiempo reelaborando un sitio web desde cero.

Flexibilidad y Portabilidad. Los usuarios pueden comunicarse fácilmente incluso si el servidor-cliente REST está alojado en diferentes servidores, ofreciendo un beneficio esencial desde la perspectiva de la gerencia.

Independencia. Gracias a la separación entre cliente y servidor, el protocolo REST facilita que los desarrollos en las diferentes áreas se produzcan de forma autónoma. Además, la API REST es ajustable a la plataforma y sintaxis operativa, lo que ofrece la posibilidad de probar numerosos entornos durante el desarrollo.

2.3.2 OPEN API y Swagger

La especificación OpenAPI es un formato de descripción de API´s para API´s REST. Un archivo OpenAPI permite describir una API incluyendo:

- Puntos finales disponibles y operaciones en cada punto final
- Parámetros de operación entrada y salida para cada operación
- Métodos de autenticación
- Información de contacto, licencia, condiciones de uso y otra información.

Las especificaciones de la API pueden escribirse en YAML o JSON. El formato es fácil de aprender y legible tanto para humanos como para máquinas.

Por otro lado, Swagger es un conjunto de herramientas de código abierto construidas en torno a la especificación OpenAPI que nos ayuda a diseñar, construir, documentar y consumir API´s REST. Las principales herramientas de Swagger son

- **Swagger Editor**. Es un editor basado en el navegador donde se pueden escribir las especificaciones OpenAPI.
- **Swagger UI**. Nos ayuda a mostrar las especificaciones OpenAPI como documentación interactiva de la API.
- **Swagger Codegen**. Genera trozos de código del servidor y bibliotecas de cliente a partir de una especificación OpenAPI

Sus ventajas predominan de tal manera que Swagger puede definirse como la aplicación estándar por excelencia para la descripción de interfaces en las API de RESTful. Como otras tantas aplicaciones de código abierto, Swagger disfruta de una gran divulgación y, con ello, de compatibilidad con muchas herramientas.

Por un lado, facilita el diseño de la API gracias a las herramientas que existen para ello, proporcionando una estructura bien definida a seguir que asegura el cumplimiento del estándar.

Por otro lado, permite centrarnos en las necesidades del consumidor de la API, abstrayéndose de los problemas que puedan surgir durante el desarrollo.

Además, existen múltiples herramientas que nos permiten generar el código fuente de clientes/servidores de API´s a partir de su diseño, la documentación de una API ya implementada, la posibilidad de validar la estructura del diseño de la API o de realizar pruebas durante el desarrollo, entre otras tantas opciones.

También reduce la dependencia entre los equipos que tienen que trabajar en la implementación de la API y permite comenzar con antelación los trabajos de estos al conocerse desde un principio la funcionalidad de la API.

Dada la necesidad de la empresa de documentar las API´s existentes pensamos que Swagger es la mejor tecnología para ello. Como bien describimos en las ventajas de esta tecnología, esta tiene una gran compatibilidad con herramientas que los desarrolladores usan en su día a día como el editor de texto IntelliJ, Github que es una plataforma que nos ayuda a controlar las versiones de nuestro software o con el framework de Java más usado, Spring Boot, que gracias a sus

anotaciones no es necesario escribir ninguna línea adicional para que se genere el documento YAML que documente nuestras API´s.

Además de esto, Swagger es un referente a la hora de documentar API´s REST, por lo que podemos tener un alto grado de confianza ya que es una dependencia que tiene un gran soporte y es mejorada constantemente por sus desarrolladores, siendo bastante improbable que se quede obsoleta o que deje de dar ofrecer sus servicios a los usuarios.

Ilustración 2. Logo Swagger

2.3.4 Event Sourcing

Las API´s definidas dentro de la plataforma de Moonshot Innovation siguen la arquitectura de "*event sourcing*", la cual define un enfoque para las operaciones de manipulación de datos garantizando que cada cambio de estado de una aplicación se almacene en un evento. Con esto lo que conseguimos es poder ir hacia atrás y ver los pasos que hemos seguido, así como saber cómo hemos llegado a ese punto. *Event sourcing* nos permite llevar un registro de los eventos y poder reconstruir estados pasados y ver que es lo causante de ese resultado, así como revertirlos si fuera necesario.

La idea principal de cómo funciona este patrón es la siguiente:

El objeto evento es aquel objeto que almacena el estado de la aplicación en un momento dado. Los objetos se almacenan en el orden en el que han sido capturados (como si hubiéramos hecho una foto en ese instante a la aplicación.)

Esto nos permite realizar consultas temporales sí como revertir cambios que hemos hecho.

Los eventos se almacenan en el denominado *events long* (almacén de eventos), a este se puede acceder en cualquier momento para acceder al estado de la aplicación en el momento que queramos.

En cierta manera, lo que conseguimos con esta arquitectura es tener un histórico de todos los eventos que se han realizado dentro de la aplicación. Para Moonshot este tipo de arquitectura proporciona una gran cantidad de ventajas entre las que se destacan las siguientes.

Análisis de datos. Al tener todas las acciones de los usuarios registrados, tenemos una gran cantidad de datos que podemos analizar. Esto nos sirve para crear otras herramientas dentro de la plataforma, como un tablero donde se muestren gráficas que monitoricen los datos recogidos. Un ejemplo de esto podría ser obtener una tabla que nos diga quienes son los usuarios que más mensajes envían dentro de la aplicación en cada mes.

Análisis del comportamiento. Obteniendo la información de estos eventos, vamos a ser capaces de ver dónde está la mayor afluencia de tráfico de nuestra aplicación y enfocar nuestra atención y nuestro tiempo en mejorar las zonas de la aplicación más transitadas por los usuarios.

Corrección de errores. En caso de que el usuario haya tenido cualquier problema dentro de la aplicación, vamos a poder consultar nuestro histórico de eventos para obtener información que nos ayude a solucionar el error. Por ejemplo, si el error ha sido desencadenado por la ejecución de otra acción anterior.

Consulta del estado de la aplicación. Los eventos se almacenan en orden en función de su fecha de ejecución, esto nos permite conocer cuál es el estado de la aplicación en cualquier momento, ya que vamos a ser capaces de saber cuáles han sido las acciones que se han realizado. Por ejemplo, analizando los eventos, vamos a ser capaces de ver cuáles son los usuarios que han iniciado sesión en

un día, o también seremos capaces de ver cuáles son los mensajes que tiene una conversación.

Reconstrucción del sistema. Como mencionamos anteriormente, este histórico guarda el estado actual de la aplicación, por lo que, en caso de ruptura del sistema o pérdida de los datos, vamos a ser capaces de recuperarlos gracias a esta arquitectura.

Este tipo de arquitectura nos ha dado la posibilidad de agregar en este proyecto la funcionalidad de documentar los eventos para saber cómo se construyen y exponer un ejemplo de cada uno. Esto lo hemos hecho porque en las peticiones que se realizan a la API para crear o actualizar un recurso es necesario mandar un evento como cuerpo de la petición, y aunque queda bien definido dentro de la documentación creada en Swagger, aporta a los usuarios la posibilidad de analizar en profundidad como están formados estos eventos y cuál es la información que se está guardando en el *events long*, manteniendo así una transparencia total con el usuario.

Dentro de esta funcionalidad, existe la posibilidad de ver todos los eventos o poder filtrar en función de la entidad a la que haga referencia. Además de esto, también se han añadido ejemplos de las clases de modelo usadas en los eventos para que el usuario no tenga ningún tipo de problema a la hora de realizar sus peticiones a la API.

2.3.5 Gestión de una API

La gestión de API es el proceso de crear y publicar interfaces de programación de aplicaciones (API) web, haciendo cumplir sus políticas de uso, controlando el acceso, nutriendo a la comunidad de suscriptores, recopilando y analizando estadísticas de uso e informando sobre el rendimiento. Los componentes de la gestión de una API proporcionan mecanismos y herramientas para apoyar a la comunidad de desarrolladores y suscriptores.

En nuestro caso, la gestión de nuestras API estará definida por la documentación de estas,

Los componentes que proporcionan la siguiente funcionalidad se encuentran generalmente en los productos de gestión de API:

- **Puerta de enlace**. Un servidor que actúa como un frontal de API, recibe peticiones de API, hace cumplir las directivas de limitación y seguridad, pasa las solicitudes al servicio de motor y luego transmite la respuesta de nuevo al solicitante. Una puerta de enlace a menudo incluye un motor de transformación para orquestar y modificar las peticiones y respuestas sobre la marcha. Una puerta de enlace también puede proporcionar funcionalidades como recopilar datos analíticos y proporcionar almacenamiento en la base de datos. La puerta de enlace puede proporcionar funcionalidad para admitir autenticación, autorización, seguridad, auditoría y cumplimiento normativo.

  Para Moonshot Innovation, las puertas de enlaces definidas al momento de la realización de este proyecto son las API´s de los cuatro módulos que conforman la aplicación, es decir, “Federation”, “Core”, “Community” y “Datahub”. Donde cada endpoint está configurado para permitir o no su acceso en función del tipo de usuario, de los permisos o si este ha iniciado sesión en la aplicación o no. Al llamar a estos endpoints se genera un evento que hace posible la recopilación de datos para su posterior análisis.

- **Herramientas de publicación**. Una colección de herramientas que los proveedores de API usan para definir API, por ejemplo, usando las especificaciones OpenAPI o RAML, generar documentación de API, gestionar políticas de acceso y uso para API, probar y depurar la ejecución de API, incluidas pruebas de seguridad y generación automatizada de pruebas y suites de pruebas, desplegar API en entornos de producción, de *staging* y de *quality assurance*, y coordinar el ciclo de vida en conjunto de API.
  En lo que respecta a este proyecto, la herramienta utilizada para la definición de estas API´s es Swagger, ayudándonos del *framework*

Angular para crear una interfaz donde el usuario se sienta más cómodo y podamos añadir más funcionalidades.

También existe la posibilidad de utilizar herramientas que permitan la integración y el despliegue continuos (CI/CD) que nos ayudan a la automatización de tareas como la ejecución de batería de pruebas o el despliegue de la aplicación en un servidor, incrementando así la productividad y acelerando los ciclos de lanzamiento, algo fundamental en caso de usar metodologías ágiles como Scrum.

En este caso, estas tareas de integración y despliegue continuo están realizadas con la herramienta Jenkins, pero existen otras muy famosas como Microsoft Azure o GitHub Actions.

- **Portal del desarrollador/tienda de API**. Es un sitio de la comunidad, generalmente con la marca de un proveedor de API, que puede encapsular para los usuarios de API, en una única fuente conveniente, información y funcionalidad incluyendo documentación, tutoriales, código de muestra, kits de desarrollo de software, una consola de API interactiva y *sandbox* para probar API, la capacidad de suscribirse a las API y gestionar las claves de suscripción, tales como el *Client ID* y *Client Secret* de OAuth2, y obtener asistencia del proveedor y usuario y comunidad de API.

  Gracias a Swagger tenemos la posibilidad de crear un entorno de desarrollo que nos permita realizar pruebas en tiempo real hacia cualquier API definida en la plataforma. Pudiendo gestionar desde esta herramienta las credenciales necesarias para el correcto funcionamiento de cada uno de los endpoints. Este portal ha sido creado para facilitar al usuario el uso de cada API incluyendo funcionalidades que permitan profundizar en el uso de estos endpoints y en la estructura de los objetos que debe conocer el usuario para realizar las peticiones de manera correcta.
- **Informes y analíticas**. Funcionalidad para monitorizar el uso y la carga de API (visitas en conjunto, transacciones completadas, número de objetos de datos devueltos, cantidad de tiempo de cómputo y otros

recursos internos consumidos, volumen de datos transferidos). Esto puede incluir el monitoreo en tiempo real de la API con alertas generadas directamente o mediante un sistema de gestión de red de nivel superior, por ejemplo, si la carga en una API se ha vuelto demasiado grande, así como funcionalidad para analizar datos históricos, como logs de transacciones, para detectar tendencias de uso. También puede ser proporcionada funcionalidad para crear transacciones sintéticas que se pueden usar para probar el rendimiento y comportamiento de endpoints de API. El proveedor de API puede utilizar la información recopilada por la funcionalidad de informes y analíticas para optimizar el ofrecimiento de API dentro del proceso de mejora continua general de una organización y para definir acuerdos de nivel de servicio de software para API.

En este proyecto, podemos monitorizar el uso de la API gracias a la arquitectura de *event sourcing* descrita anteriormente.

- **Monetización**. Funcionalidad para soportar cobrar por el acceso a API comerciales. Esta funcionalidad puede incluir soporte para configurar reglas de precios, basadas en el uso, la carga y la funcionalidad, emitir facturas y cobrar pagos, incluidos varios tipos de pagos con tarjeta de crédito.

  El producto que ofrece Moonshot Innovation es de pago y esta cantidad de dinero vendrá dada en función de los módulos que requiera el usuario. Inicialmente el producto se incluirá con la adquisición de la plataforma, pero esto puede estar sujeto a cambio en el futuro, ya que esto aportará un gran valor a la hora de poder crear otras funcionalidades como una tienda de módulos creada por desarrolladores externos.

2.4 Estado del arte

Este trabajo se basa principalmente en la construcción de la documentación de las API´s de Moonshot Innovation, siendo todas estas API´s de tipo REST. Es por ello por lo que se ha considerado necesario realizar una explicación sobre los tipos de API´s que existen y como ha avanzado la utilización de estas en el tiempo.

En el apartado anterior, se explicó con detalle que es una API REST y cuáles son las ventajas que ofrece, pero también existe otro tipo de API conocida como API SOAP.

SOAP (Protocolo simple de acceso a objetos) es un protocolo diseñado antes que el protocolo REST y la idea principal de este diseño consisten en garantizar que los programas creados en diferentes plataformas y lenguajes de programación pudieran intercambiar datos de una manera fácil. Siendo un protocolo, impone ciertas reglas integradas que aumenta la complejidad y la sobrecarga, lo cual puede retrasar el tiempo que tardan las páginas en cargarse. Los estándares de cumplimiento integrados incluyen la seguridad, la atomicidad, la uniformidad, el aislamiento y la durabilidad (ACID), que forman un conjunto de propiedades que garantizan operaciones confiables de las bases de datos. El envío de una solicitud de datos a una API de SOAP se puede administrar a través de cualquiera de los protocolos de la capa de la aplicación: HTTP (para los exploradores web), SMTP (para el correo electrónico), TCP, entre otros. Sin embargo, una vez que se recibe una solicitud, los mensajes SOAP de retorno deben ser documentos XML, que es un lenguaje de marcado que comprenden las personas y las máquinas.

Podemos ver las diferencias más notables en la siguiente tabla:

SOAP	REST
SOAP es un protocolo	REST es un estilo arquitectónico
SOAP utiliza interfaces de servicios para exponer la lógica de negocio	REST utiliza las URI para exponer la lógica de negocio
SOAP requiere más ancho de banda y por qué hace definición de todo en su formato	REST requiere menos ancho de banda ya que solo define su arquitectura
SOAP define las normas que deben seguirse estrictamente	REST define pocos estándares

SOAP define su propia seguridad	REST hereda las medidas de seguridad de transporte subyacente
SOAP permite XML como único formato de datos	REST permite diferentes formatos de datos tales como texto sin formato, HMTL, XML, JSON, etc.

Tabla 1. Diferencias entre SOAP y REST

3. Solución

Para cubrir la necesidad de tener un API Developer Portal se desarrolló una web que recogiera las tres funcionalidades que se pretendían implementar, teniendo comunicación entre las diferentes partes de la web. Para esto se utilizó el *framework* Angular y los lenguajes TypeScript, HTML y CSS. A continuación, se muestran las imágenes de la página web.

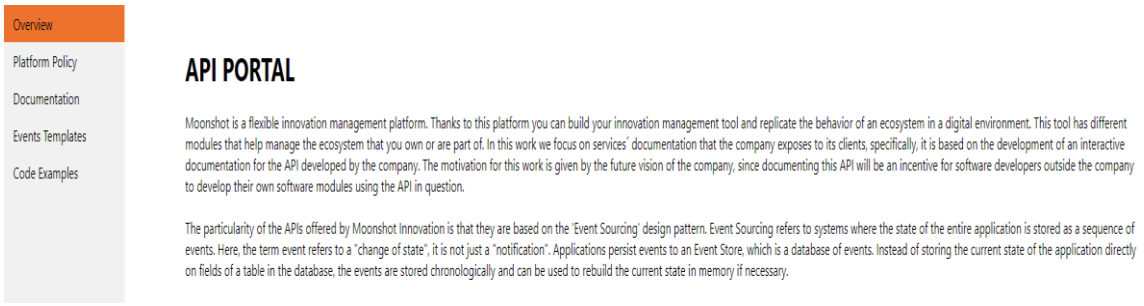


Ilustración 3.Overview (Web final)

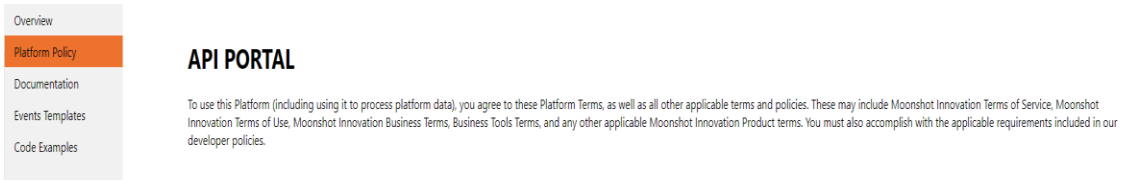


Ilustración 4.Platform policy (Web final)

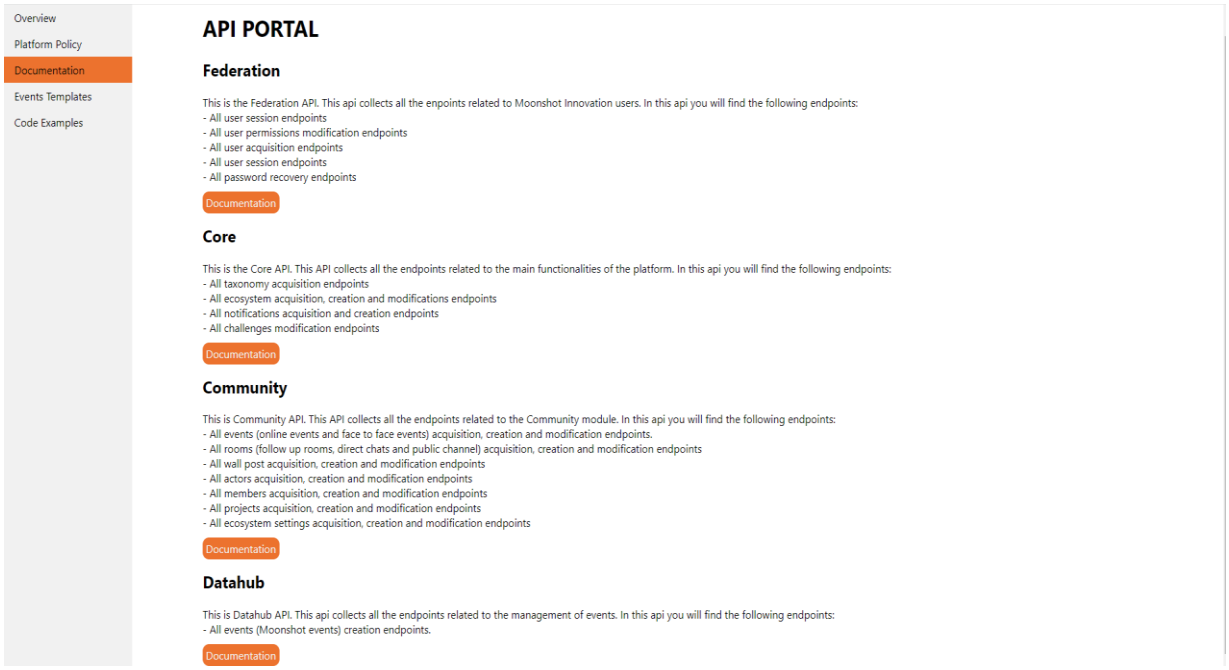


Ilustración 5. Documentation (Web final)

En primer lugar, la documentación de todas las API´s y sus respectivos endpoints fueron creadas a través de la definición del documento YAML que Swagger utiliza para presentar su interfaz. A continuación, se muestras las imágenes de la funcionalidad descrita.

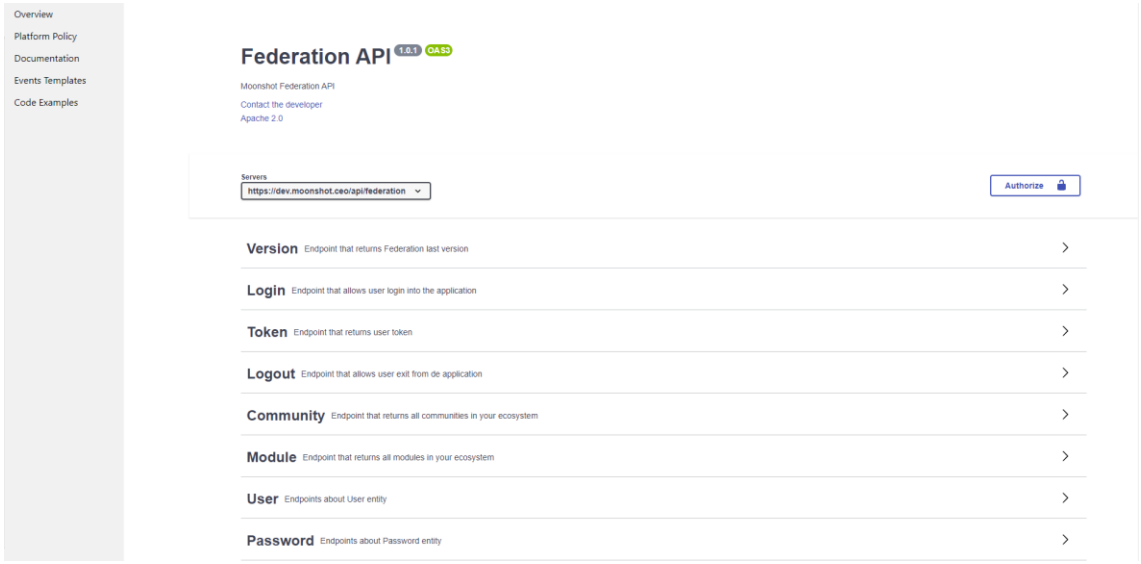


Ilustración 6. Federation Swagger (Web final)

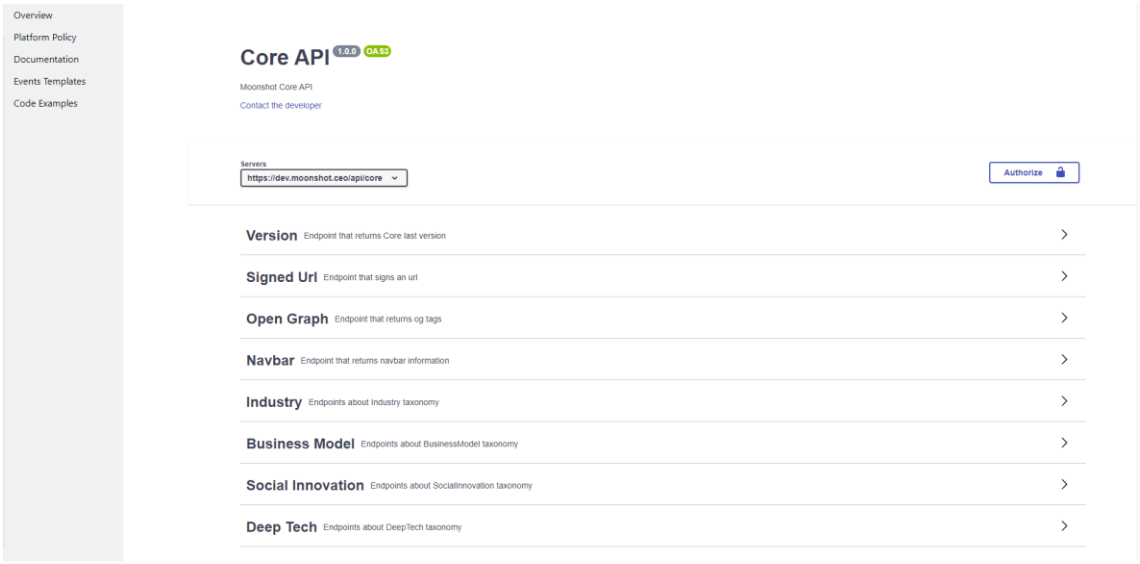


Ilustración 7. Core Swagger (Web final)

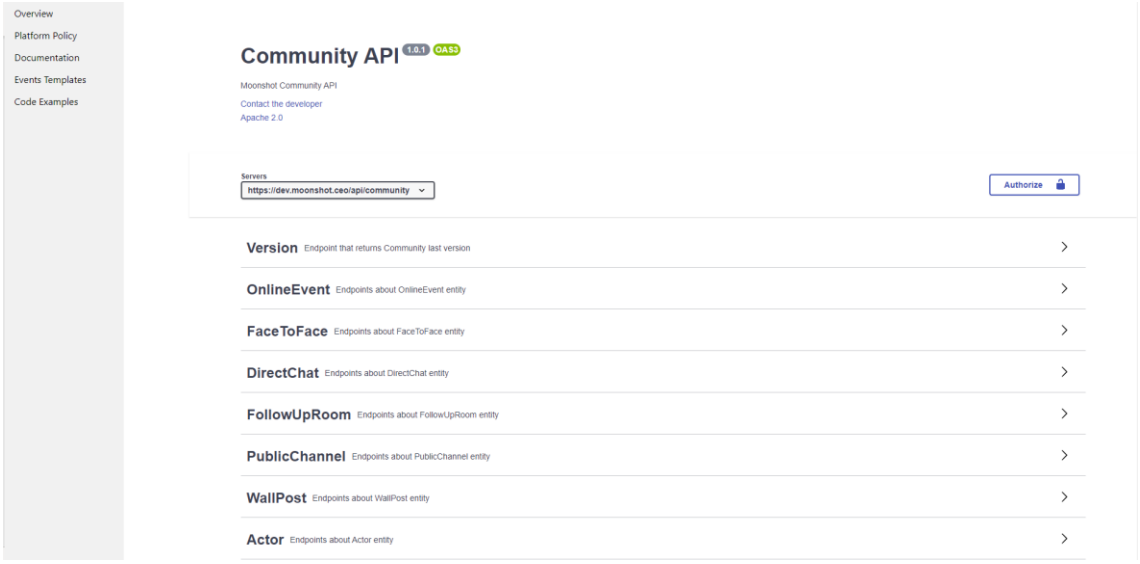


Ilustración 8. Community Swagger (Web final)

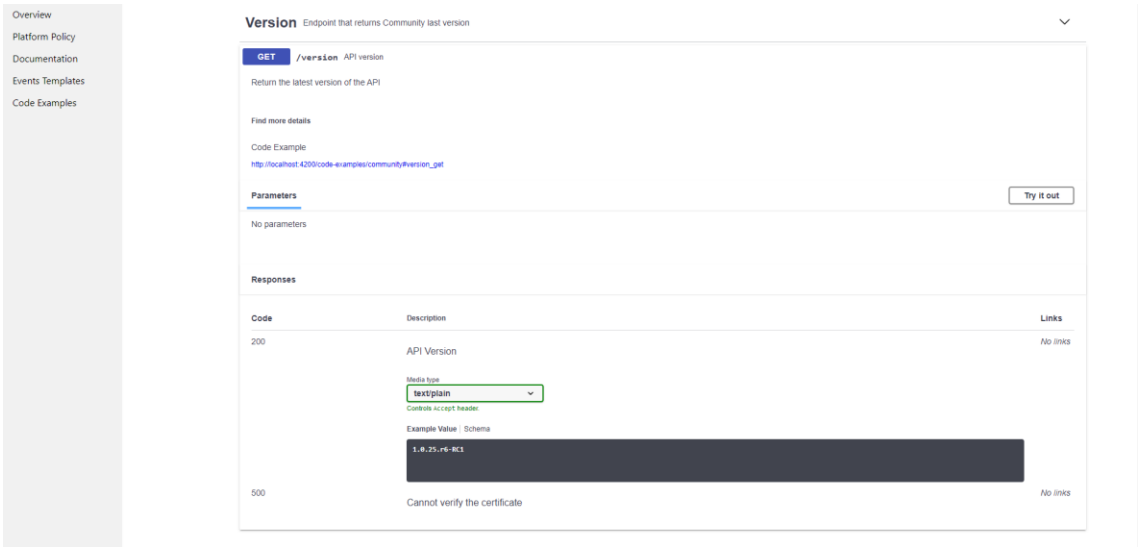


Ilustración 9. Endpoint Swagger (Web final)

Como se puede observar, cada endpoint tiene un enlace que apunta al ejemplo de código en los lenguajes de Javascript, Python y Java, además se puede observar el botón de "*Try it out*" que habilita la zona de pruebas en tiempo real.

Por último, se puede ver un apartado para los parámetros, donde se vería los parámetros necesarios para el correcto funcionamiento del endpoint, tanto si son parámetros incluidos en la ruta o si necesita un objeto dentro de la petición.

Por otro lado, los esquemas de los eventos de la aplicación fueron creados en Angular de manera interna, recogiendo toda la información de cada evento en archivos JSON que fueron leídos y transformados para un mejor visualización con ayuda del framework. En una primera instancia, cuando seleccionamos el botón para ver los esquemas se muestran todos los eventos y todos los modelos que utilizan estos. Si pulsamos en una de las entidades, realizaremos un filtrado para mostrar los eventos que estén relacionados con la entidad seleccionada. A continuación, se presentan las imágenes de la funcionalidad descrita.

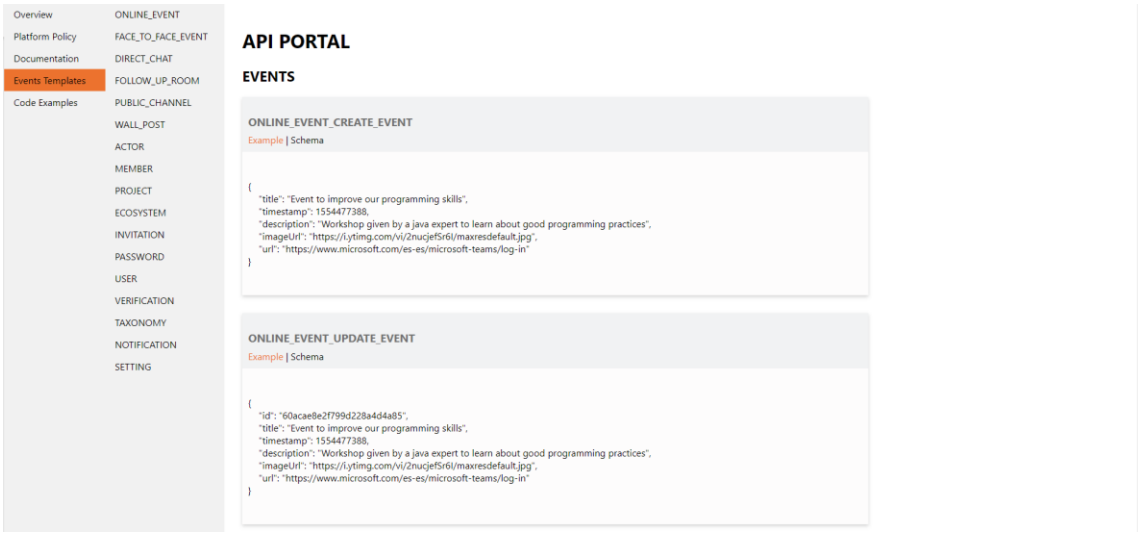


Ilustración 10. Todos los esquemas de eventos

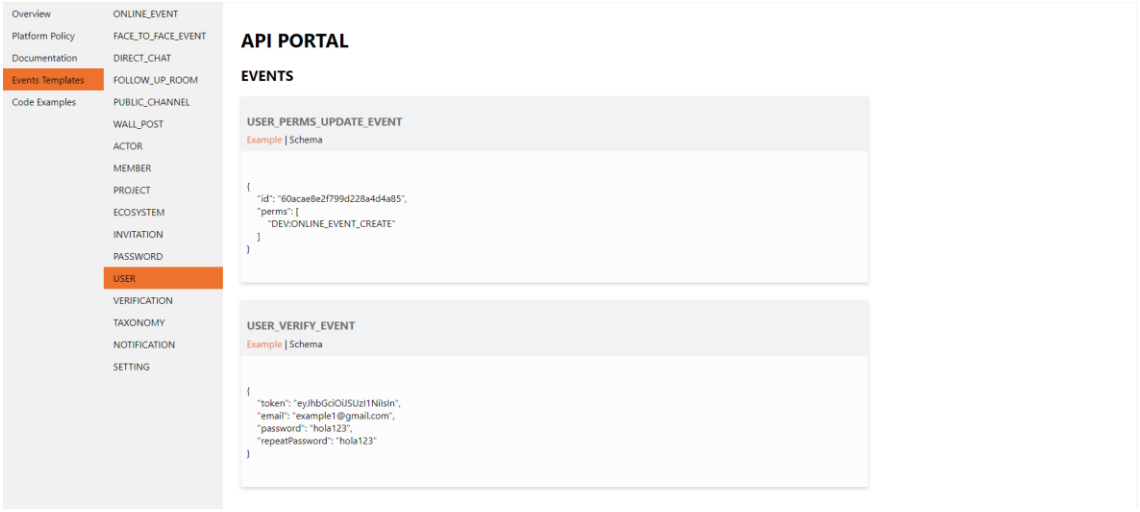


Ilustración 11. Esquema de eventos filtrados

En cada evento, inicialmente se ve la vista que refleja una muestra con valores de ejemplo. En cambio, si pinchamos en el botón "*Schema*" podemos ver el tipo de valor de cada propiedad que debería poner el desarrollador para crear ese tipo de evento.

Como existen propiedades cuyo tipo hace referencia a otro tipo de objeto, también se ha creado un apartado que representa las clases de modelo que utilizan estos eventos. Estos enlaces se encuentran al final de la vista de "*Schema*".

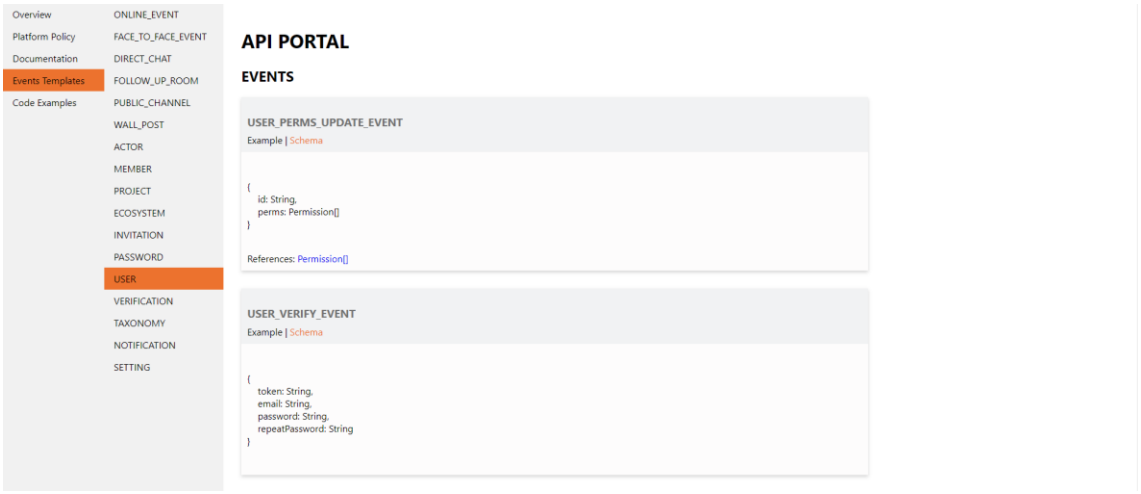


Ilustración 12. Esquema de eventos (Esquema)

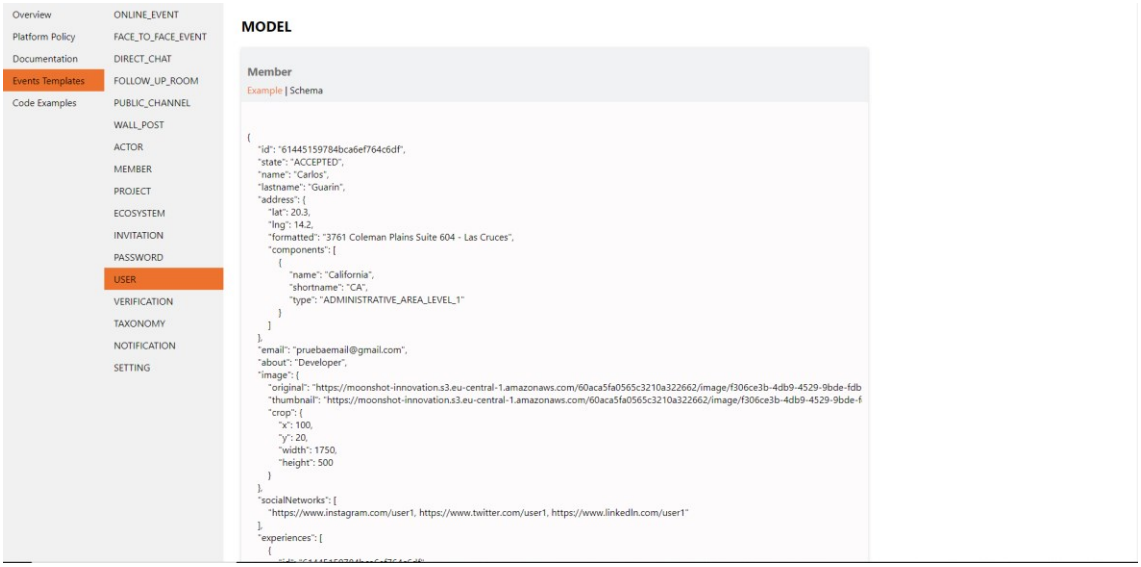


Ilustración 13. Esquema de eventos (Ejemplo)

Cada clase de modelo también esta divida en dos vistas al igual que los eventos, ya que existen algunas clases que también tienen propiedades cuyos tipos son referencias a otras objetos de modelo.

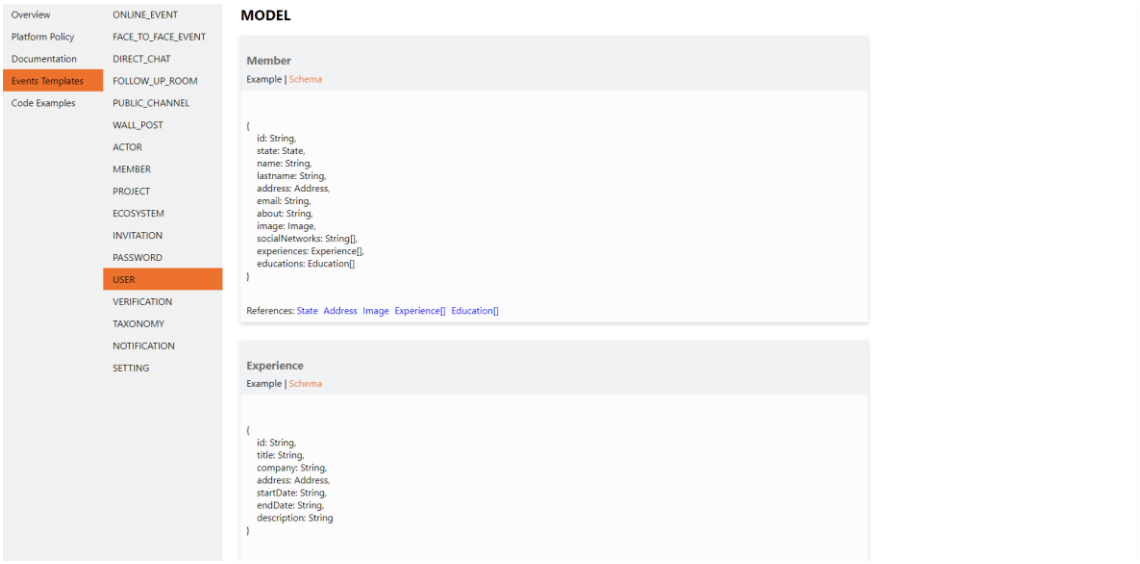


Ilustración 14. Esquema de eventos (Modelo)

Por último, se desarrolló una tercera funcionalidad que muestra los fragmentos de código que son necesarios agregar en una aplicación para realizar las peticiones a las API´s. Cuando nos movemos al apartado de "*Code examples*" inicialmente vemos los fragmentos de código del módulo de Core. En cada uno de ellos se muestra el nombre, el tipo de llamada HTTP, un botón para copiar el código y un selector para escoger el lenguaje de programación.

Cada fragmento de código se divide en dos partes, una donde se muestra el código necesario para realizar la llamada y otra donde se muestra un ejemplo de la respuesta que nos debería devolver si hacemos la llamada correctamente.

El botón ha sido implementado para aportar usabilidad a la aplicación haciendo más sencilla la tarea de copiar. Por otro lado, el selector permite escoger entre los tres lenguajes definidos antes de comenzar el proyecto Javascript, Python y Java.

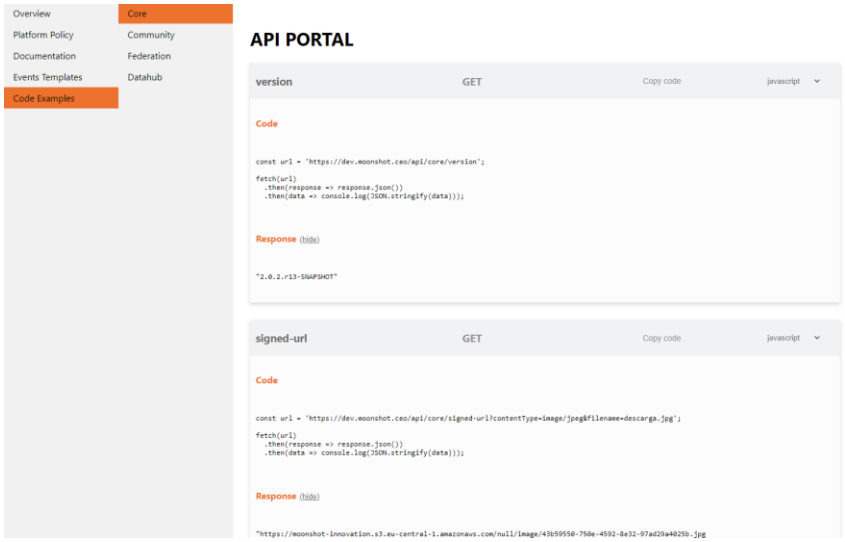


Ilustración 15. Ejemplos de códigos (Javascript)

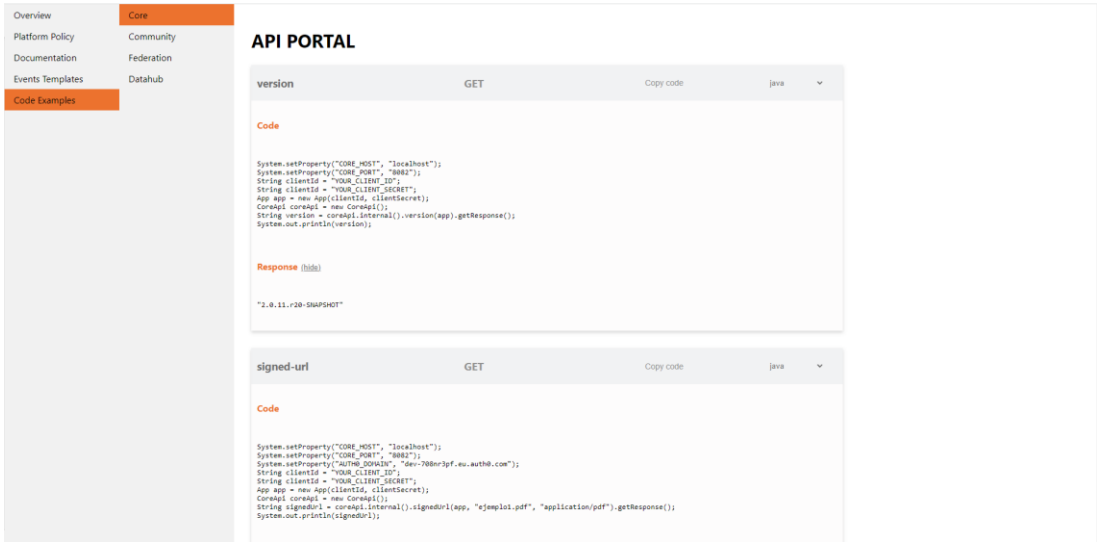


Ilustración 16. Ejemplos de códigos (Java)

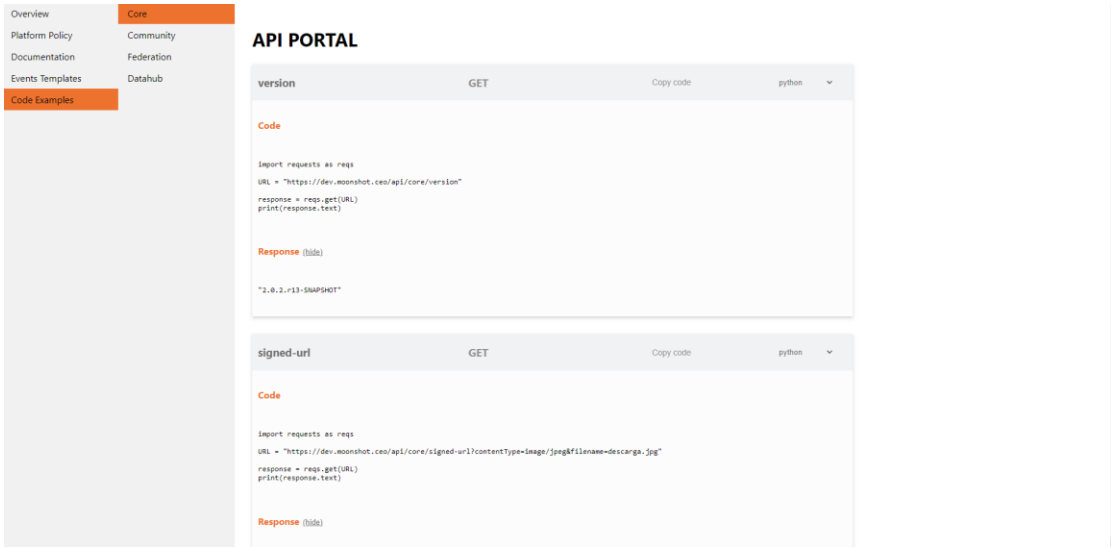


Ilustración 17. Ejemplos de códigos (Python)

4. Proceso

4.1 Metodología

4.1.1 Definición de la metodología

La metodología que se ha empleado para la realización de este trabajo es una metodología ágil que prima el desarrollo iterativo e incremental. Esta metodología está basada en la metodología ágil Scrum, donde se realizan entregas parciales y regulares del producto final, priorizadas por el beneficio que aportar al receptor del proyecto.

En Scrum, los requisitos definidos por el equipo se definen en las historias de usuario. Estas son una explicación en lenguaje natural de la funcionalidad que hay que desarrollar, donde se recoge diversa información como la descripción, su prioridad y los criterios que tiene que cumplir el producto software para dar por finalizada la funcionalidad.

En la metodología que hemos seguido nosotros, no hemos documentado estas historias de usuario, teniendo presente en todo momento que las funcionalidades principales eran la documentación de las API´s, los ejemplos de código y las plantillas de los objetos usados en las API´s.

Se ha decidido escoger esta metodología porque nos permite obtener resultados lo más rápido posible y donde los requisitos son cambiantes o poco definidos, donde la innovación, la competitividad, la flexibilidad y la productividad son fundamentales.

En esta metodología se han definido ciclos temporales y de duración fija. En concreto se ha acordado que cada ciclo debe tener una duración de tres semanas. Cada iteración tiene que proporcionar un resultado completo, un incremento de producto final que sea susceptible de ser entregado con el mínimo esfuerzo al cliente cuando lo solicite. El proceso parte de la lista de objetivos/requisitos priorizada del producto, que actúa como plan del proyecto.

4.1.2 Roles

Al igual que SCRUM, se han definido una serie de roles que serán definidas a continuación:

Product Owner. Su principal misión es encargarse de que exista una priorización clara de los objetivos a conseguir, con el propósito de maximizar el valor del trabajo que lleva a cabo el equipo. También es el encargado de representar a todas las personas interesadas (*stakeholders*) para conseguir una buena definición de los objetivos.

Este rol ha sido ejercido por David Suriol.

Scrum Master. Su principal misión es conseguir un equipo de alto rendimiento. El Scrum Master actúa como facilitador de reuniones donde pensar de manera conjunta y quita los obstáculos más allá del equipo que le impiden ser ágil.

Este rol ha sido ejercido por Jose Juan Hernández e Iñigo Aramburu.

Development Team. Cuando se habla específicamente de «equipo de desarrollo» se refiere al conjunto de personas más «técnicas» que de manera conjunta desarrollan el producto del proyecto. Tienen un objetivo común, comparten la responsabilidad del trabajo que realizan (así como de su calidad) en cada iteración y en el proyecto.

Este rol ha sido ejercido por Saulo Santana.

4.1.3 Planificación de una iteración

Para planificar una iteración hemos seguido los siguientes pasos:

- **Selección de requisitos (2 horas aproximadamente)**. Tras una reunión del *product owner* con los diferentes clientes o *stakeholders*, se presenta al equipo la lista de requisitos priorizada del producto o proyecto. El equipo pregunta al cliente las dudas que surgen y selecciona los requisitos más prioritarios que prevé que podrá completar en la iteración, de manera que puedan ser entregados si el cliente lo solicita.

- **Planificación de la iteración (2 horas aproximadamente)**. El equipo de desarrollo elabora la lista de tareas de la iteración necesarias para desarrollar los requisitos seleccionados. La estimación de esfuerzo se hace de manera conjunta y el equipo de desarrollo se encarga de desarrollar las tareas en el orden en que han sido priorizadas apoyado por los Scrum Masters en caso de que exista un bloqueo o dudas.

Una vez se ha planificado la iteración, cada tres semanas se realiza una reunión para actualizar al equipo con una duración de 15 minutos, normalmente realizado a través videollamadas dentro de la propia plataforma de Moonshot. El equipo inspecciona el trabajo que se ha realizado en esa iteración para poder hacer las adaptaciones necesarias que permitan cumplir con la previsión de objetivos a mostrar al final de la iteración. En cada reunión el equipo de desarrollo responde a tres preguntas:

- ¿Cuáles han sido los cambios o las mejoras que se han realizado en la iteración y cuáles son los beneficios que aportan esos cambios al proyecto?
- ¿Cuáles son los siguientes requisitos que se pretenden cubrir para incrementar el valor del producto?
- ¿Qué impedimentos tiene el equipo o va a tener que impidan conseguir el objetivo?

Durante la iteración los Scrum Masters se encargan de que el equipo pueda mantener el foco para cumplir con sus objetivos.

- Elimina los obstáculos que el equipo no puede resolver por sí mismo.
- Protege al equipo de interrupciones externas que puedan afectar el objetivo de la iteración o su productividad.
- Validan que las implementaciones y mejoras realizadas en la iteración cumplen los estándares de la empresa y que el incremento obtenido en la iteración satisfaga los requisitos definidos por los clientes.

4.1.3 Beneficios de la metodología

Entre los beneficios principales de aplicar la metodología podemos destacar.

Gestión regular de las expectativas del cliente. El cliente establece sus expectativas indicando el valor que le aporta cada requisito del proyecto y cuando espera que esté completado.

Resultados anticipados. El cliente puede empezar a utilizar los resultados más importantes del proyecto antes de que esté finalizado por completo.

Flexibilidad y adaptación. De manera regular el cliente redirige el proyecto en función de sus nuevas prioridades, de los cambios en el mercado, de los requisitos completados que le permiten entender mejor el producto, de la velocidad real de desarrollo, etc.

Retorno de la inversión. Cuando el beneficio pendiente de obtener es menor que el coste de desarrollo, el cliente puede finalizar el proyecto.

Mitigación de riesgos. Desde el primer momento, el equipo debe gestionar los problemas que puedan aparecer. Al hacer patentes estos riesgos, es posible iniciar su mitigación de manera anticipada.

Productividad y calidad. De manera regular el equipo va mejorando y simplificando su forma de trabajar.

Alineamiento entre cliente y equipo. Todos los participantes en el proyecto conocen cuál es el objetivo por conseguir.

Equipo motivado. Las personas están más motivadas cuando pueden usar su creatividad para resolver problemas y cuando pueden decidir organizar su trabajo.

4.2 Herramientas y recursos utilizados

A continuación, se presentan las herramientas y recursos utilizados para la creación de este proyecto.

4.2.1 Angular

Angular es un *framework* para aplicaciones web desarrollado en TypeScript, de código abierto, mantenido por Google, que se utiliza para crear y mantener aplicaciones web de una sola página. Su objetivo es aumentar las aplicaciones basadas en navegador con capacidad de Modelo-Vista-Controlador (MVC).

Angular se basa en clases tipo "Componentes", cuyas propiedades son las usadas para hacer el enlace de los datos. En dichas clases tenemos propiedades (variables) y métodos (funciones a llamar). La ventaja que nos ofrece a la hora de realizar estos componentes es la reutilización de los mismos, evitando que los desarrolladores dupliquen código y facilitando una visualización de los estilos uniforme a lo largo de toda la aplicación.

El **enlace de datos** (*data binding*) es un proceso que permite a los usuarios manipular elementos de la página web a través de un navegador web. Emplea HTML dinámico y no requiere secuencias de comandos ni programación complejas.

El motivo de escoger Angular como *framework* para realizar este proyecto es que la plataforma que ha creado Moonshot Innovation esta desarrollada con esta tecnología, por lo que realizar la integración del proyecto en el futuro será más sencillo usando el mismo *framework*. Además, teniendo en cuenta que esta aplicación no realiza llamadas a servidores, es ideal que sea una aplicación de página única (SPA) evitando recargas y tiempos de espera adicionales.

Ilustración 18. Logo Angular

4.2.2 TypeScript

TypeScript es un lenguaje de programación libre y de código abierto desarrollado y mantenido por Microsoft. Es un superconjunto de JavaScript, que esencialmente añade tipos estáticos y objetos basados en clases. Este es usado para desarrollar aplicaciones JavaScript que se ejecutarán en el lado del cliente o del servidor, o extensiones para programas

TypeScript extiende la sintaxis de JavaScript, por tanto, cualquier código JavaScript existente debería funcionar sin problemas. Está pensado para grandes proyectos, los cuales a través de un compilador de TypeScript se traducen a código JavaScript original.

TypeScript soporta ficheros de definición que contengan información sobre los tipos de librerías JavaScript existentes, permitiendo a otros programas usar los valores definidos en los ficheros como si fueran entidades TypeScript de tipado estático.

Ilustración 19. Logo de TypeScript

4.2.3 HTML 5

HTML ('lenguaje de marcado de hipertexto'), hace referencia al lenguaje de marcado para la elaboración de páginas web. Es un estándar que sirve de referencia del software que conecta con la elaboración de páginas web en sus diferentes versiones, define una estructura básica y un código (denominado código HTML) para la definición de contenido de una página web, como texto, imágenes, videos, juegos, entre otros. Es un estándar a cargo del *World Wide Web Consortium* (W3C)

El lenguaje HTML basa su filosofía de desarrollo en la diferenciación. Para añadir un elemento externo a la página, este no se incrusta directamente en el código de la página, sino que se hace una referencia a la ubicación de dicho elemento mediante texto. De este modo, la página web contiene solamente texto mientras que recae en el navegador web (interpretador del código) la tarea de unir todos los elementos y visualizar la página final.

Ilustración 20. Logo HTML5

4.2.4 CSS

CSS, es un lenguaje de diseño gráfico para definir y crear la presentación de un documento estructurado escrito en un lenguaje de marcado. Es muy usado para establecer el diseño visual de los documentos web, e interfaces de usuario escritas en HTML o XHTML; el lenguaje puede ser aplicado a cualquier documento XML, incluyendo XHTML, SVG, XUL, RSS, etcétera.

CSS está diseñado principalmente para marcar la separación del contenido del documento y la forma de presentación de este, características tales como las capas o *layouts*, los colores y las fuentes Esta separación busca mejorar la accesibilidad del documento, proveer más flexibilidad y control en la especificación de características presentacionales, permitir que varios documentos HTML compartan un mismo estilo usando una sola hoja de estilos separada en un archivo .css, y reducir la complejidad y la repetición de código en la estructura del documento.

La separación del formato y el contenido hace posible presentar el mismo documento marcado en diferentes estilos para diferentes métodos de renderizado, como en pantalla, en impresión, en voz (mediante un navegador de voz o un lector de pantalla), y dispositivos táctiles basados en el sistema Braille. También se puede mostrar una página web de manera diferente dependiendo del tamaño de la pantalla o tipo de dispositivo.

Ilustración 21. Logo CCS3

4.2.5 Moonshot Innovation

Para este proyecto fue indispensable la utilización del código fuente de la plataforma de Moonshot Innovation, ya que en él se encuentra la definición de las diferentes API´s.

Además de esto, fue necesario el uso de la infraestructura de la plataforma, ya que se tuvieron que realizar innumerables accesos a las bases de datos y a los eventos generados por la arquitectura para poder obtener información y ejemplo útiles para el desarrollo de este proyecto.

4.3 Iteraciones

La planificación de este proyecto viene descrita en la siguiente tabla.

Fases	***Duración Estimada*** ***(horas)***	***Tareas*** ***(nombre y descripción, obligatorio al menos una por fase)***

Prototipo funcional	65	Tarea 1.1: Identificación de los requisitos
		Tarea 1.2: Desarrollo del prototipo
		Tarea 1.3: Pruebas de funcionamiento con usuarios reales
Playground	65	Tarea 2.1: Recolección de información de los endpoints de la API
		Tarea 2.2: Implementación de la página web
		Tarea 2.3: Pruebas de funcionamiento con usuarios reales
Transcribir a Swagger las APIs existentes	65	Tarea 3.1: Recolección de todas las APIs existentes
		Tarea 3.2: Traducción de las APIs a Swagger
		Tarea 3.3: Pruebas de funcionamiento con usuarios reales
Creación de ejemplos de códigos para cada endpoint en diversos lenguajes	65	Tarea 4.1: Identificar los requisitos de los usuarios
		Tarea 4.2: Implementación de la nueva funcionalidad
		Tarea 4.3: Pruebas de funcionamiento con usuarios reales
Documentación/ Presentación	20	Tarea 5.1: Creación de la documentación
		Tarea 5.2: Creación del material de apoyo para la presentación

Tabla 2. Planificación de las iteraciones

En la primera fase, se desarrolló el prototipo funcional del proyecto. Para ello, se concertó una reunión entre el equipo incluyendo al "Product Owner" para identificar los requisitos que debía tener el producto inicialmente. En esta sección, se produjo el desarrollo de un MVP (Producto mínimo viable) que es una versión del producto que permite obtener una batería de opiniones iniciales por parte de los usuarios, y se propuso que este producto mínimo fuese la documentación de una de las API´s de Moonshot. En concreto, en la API del módulo de Federation.

Se escogió este módulo porque era el más pequeño de todos y al no tener ninguna experiencia con las tecnologías, decidimos que era lo mejor para ir familiarizándome con ellas.

En esta primera versión, ya pudimos observar el estilo general del proyecto, que viene dado por "Swagger", la distribución de los diferentes endpoints, los cuales están divididos en función de la entidad que manejan y por otro lado podemos ver la zona de "*sandbox*" en el cual se activa pulsando en el botón de "*Try it out*" que podemos ver en Ilustración 9 y nos permite realizar una llamada a la API en tiempo real.

En la segunda fase, tuvimos que recoger la información de todos los endpoints, ya que hay endpoints que tienen permisos especiales y había otros que hubo que refactorizar para seguir con las recomendaciones de la definición de una API REST. Por tanto, en esta fase se añadieron los permisos, las claves de autorización y una barra de navegación a la página para dar más visibilidad y accesibilidad a todos los elementos de la página.

En la tercera fase, una vez tuvimos toda la información necesaria, comenzamos con la transcripción de las API´s de los módulos de "Community" y "Core" a Swagger, consiguiendo así una alta cobertura en la documentación de todas las API´s de Moonshot. Cabe mencionar, que esta fase es la que mayor tiempo de desarrollo nos ocupó debido a la gran cantidad de endpoints que poseen las API´s.

Una vez, ya completada toda la transcripción y documentación de los endpoints a Swagger e integrados en la página web, decidimos crear otra funcionalidad para que los usuarios que hicieran uso de este proyecto tuvieran más facilidades a la hora de crear sus propios módulos y tuvieran en conocimiento cual era la forma correcta de hacer las llamadas a los diferentes endpoints.

Esta consiste en la visualización de fragmentos de códigos preparados con ejemplos para que los desarrolladores que quieran hacer una llamada a un endpoint únicamente tengan que copiar el código que se refleja en la web.

Al implementar esta funcionalidad, solo existía la opción de ver estos fragmentos de código en el lenguaje de “Javascript”, pero gracias a la retroalimentación de los usuarios y decidimos mejorar la funcionalidad para también tener estos fragmentos de código en los lenguajes de programación “Python” y “Java”.

Para poder mostrar esta funcionalidad en “Java” fue necesario crear unas librerías que permitieran el uso de las API´s.

Sin embargo, pensamos que esta funcionalidad se quedaba corta porque algunos de los objetos que se mandan en las peticiones de crear y actualizar son eventos de la arquitectura de las API´s. Es por ello, que decidimos mejorar esta implementación añadiendo otra funcionalidad que recogiera todas las plantillas de los eventos que se mandan en las peticiones de Moonshot. Estas plantillas están dividas por el tipo de entidad que maneja cada evento, y además permite al usuario filtrar en base al módulo que quiera ver.

También, se tuvo en cuenta que hay propiedades de estos eventos que son otros objetos. Es por ello por lo que cada plantilla esta divida en un ejemplo, donde se puede ver un evento con campos rellenados a modo de ejemplo y su esquema con referencias a las plantillas de los objetos.

4.4 Ejecución

Como se describe previamente en este documento, para la realización de este trabajo se ha llevado a cabo una metodología ágil, es decir, se han realizado un proceso iterativo e incremental de duración predefinida en los cuales al final de cada ciclo se ha obtenido un producto que ofrece valor al usuario final, en este caso la empresa.

Esta metodología consiste en reunir al equipo cada tres semanas aproximadamente para realizar una revisión del progreso obtenido durante ese ciclo, repriorizar en caso de que fuera necesario las funcionalidades implementadas y definir los requisitos de usuario que debían ser cubiertos en la siguiente entrega.

4.4.1 Primer sprint

En la primera reunión, ambos tutores del trabajo que actúan como Scrum Managers, definieron el requisito principal de la aplicación que es la documentación de las API´s. Para ello definimos que el proceso de documentación debía comenzar con la API de Federation, ya que era la que menos cantidad de endpoints tenía y además estos endpoints están relacionados con funcionalidades fundamentales de todas las aplicaciones como el inicio y cierre de sesión, la gestión de los tokens y contraseñas de los usuarios.

El sentido de empezar por la API más pequeña reside en el poco conocimiento que tenía al momento de comenzar este proyecto sobre Swagger y los estándares de Open API. Siendo preferible entregar un incremento del producto más pequeño para ver los fallos y poder corregir los errores y malas prácticas antes que desarrollar la documentación de un módulo muy grande donde se pueden cometer muchos más errores. Gastando así el doble de tiempo en documentar una sola API.

Fue por ello por lo que comencé viendo la documentación de Swagger para ir entendiendo su funcionamiento, su sintaxis, su uso, sus ventajas y también el valor que aporta al proyecto. Tras ver la documentación, cree un ejemplo sencillo de un servicio inventado que me sirvió a modo de pruebas para ir siguiendo la documentación e ir probando los límites de la herramienta.

Una vez había entendido como funcionaba Swagger y cuáles eran las posibilidades que ofrecía, me dispuse a utilizar el editor de Swagger para generar el fichero YML con la documentación de la API del módulo de Federation.

El resultado de este fichero se presenta en la siguiente imagen:

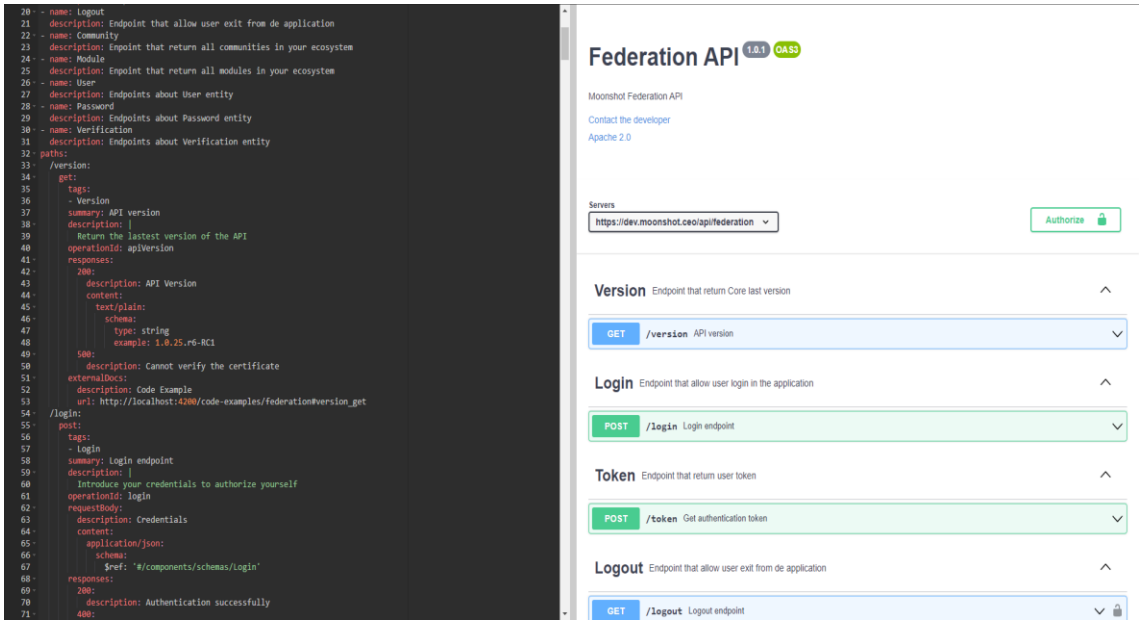
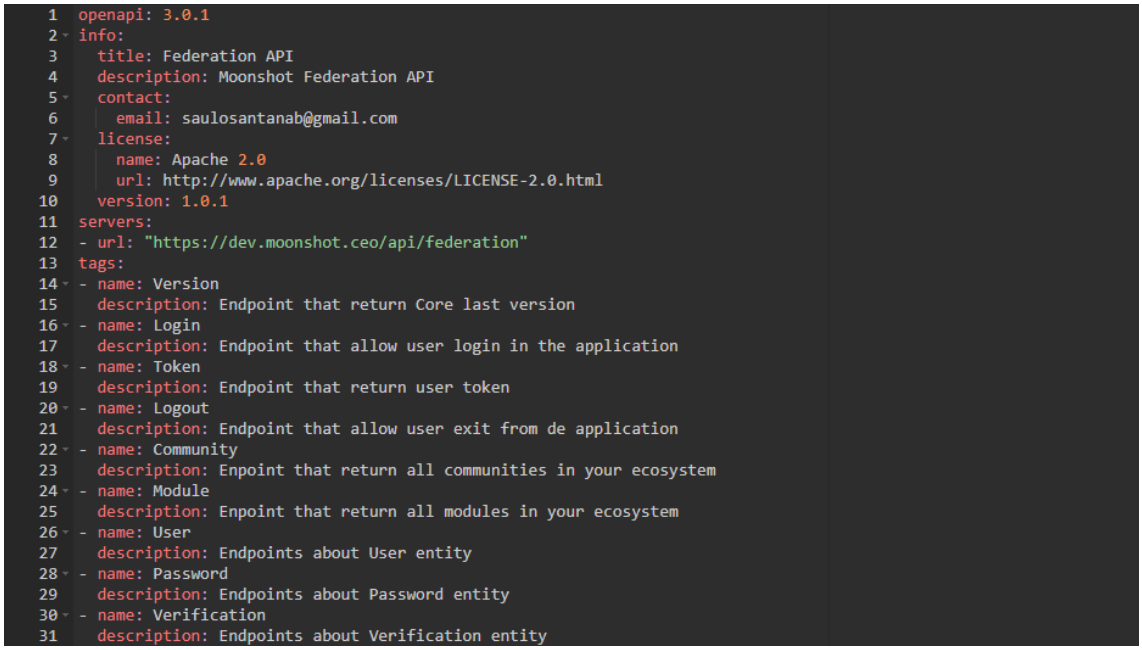


Ilustración 22. Primera versión del prototipo

La estructura del fichero YML es sencilla y amigable. Este utiliza los espacios y los saltos de línea diferenciando por bloques la información que tiene que leer. La estructura del archivo consiste en un mapa clave-valor donde cada clave debe ser única.

En primer lugar, se definen los parámetros generales como el nombre de la API, una descripción de la misma, el contacto del desarrollador, la licencia, la versión, el servidor al que debemos hacer referencia cuando probemos nuestros endpoints y una serie de tags que serán las que nos permitan agrupar los endpoints.



```yaml
openapi: 3.0.1
info:
  title: Federation API
  description: Moonshot Federation API
  contact:
    email: saulosantanab@gmail.com
  license:
    name: Apache 2.0
    url: http://www.apache.org/licenses/LICENSE-2.0.html
  version: 1.0.1
servers:
- url: "https://dev.moonshot.ceo/api/federation"
tags:
- name: Version
  description: Endpoint that return Core last version
- name: Login
  description: Endpoint that allow user login in the application
- name: Token
  description: Endpoint that return user token
- name: Logout
  description: Endpoint that allow user exit from de application
- name: Community
  description: Enpoint that return all communities in your ecosystem
- name: Module
  description: Enpoint that return all modules in your ecosystem
- name: User
  description: Endpoints about User entity
- name: Password
  description: Endpoints about Password entity
- name: Verification
  description: Endpoints about Verification entity
```

Ilustración 23. Descripción de la API (YAML)

A continuación, se definen los endpoints que contiene la API. Para ello, primero debemos definir la estructura de un endpoint empezando por definir la ruta a la que hace referencia, siendo esta única.

Una vez hemos definido la ruta, deberemos indicar cuál es el método HTTP (GET, POST, PUT, DELETE) al que hace referencia nuestro endpoint. Tras esto, añadiremos parámetros opcionales tales como las etiquetas, un sumario, una descripción que cuente brevemente para que se utilizará este endpoint y un id para nuestra operación que tiene que ser único. Esto nos sirve para completar más la documentación del endpoint.

En caso de que el endpoint que estemos documentando requiera el uso de parámetros en la ruta, ya sea de tipo cadena de consulta ("*query param*") o parámetros integrados en la propia ruta ("*path param*") deberemos indicarlo utilizando la etiqueta "*parameters*" y dentro de esta se definirá el nombre, el tipo, la descripción, si es obligatorio o no y un esquema en el que podemos indicar el formato y un ejemplo del parámetro.

Hay que tener en cuenta que, si el parámetro que se establece es un parámetro de la ruta, deberemos modificar la ruta de nuestro endpoint.

```yaml
/users/{id}:
  get:
    tags:
    - User
    summary: Get user by id
    description: |
      Find a user in the database by its identifier and get all the data associated
    operationId: userById
    parameters:
    - name: id
      in: path
      description: User id
      required: true
      schema:
        type: string
        format: mongo-id
        example: 61445159784bca6ef764c6df
```

Ilustración 24. Tipo, tags y path params (YAML)

```yaml
/users/by/email:
  get:
    tags:
      - User
    summary: Get user by email
    description: |
      Find a user in the database by email
    operationId: userByEmail
    parameters:
    - name: email
      in: query
      description: User email
      required: true
      schema:
        type: array
        items:
          type: string
          format: email
          example: example1@gmail.com
```

Ilustración 25. Query params (YAML)

Por otro lado, si la petición es de tipo POST O PUT, puede que requiera enviar un objeto en el cuerpo. Para ello hacemos uso de la etiqueta "*requestBody*", y en esta especificaremos una descripción del cuerpo y una esquema o referencia a un esquema del cuerpo que debemos enviar.

A continuación se presenta un ejemplo del uso de esta etiqueta:

```yaml
459       requestBody:
460         description: Object that contains all parameters to update the user permissions
461         content:
462           application/json:
463             schema:
464               $ref: '#/components/schemas/UserPermsUpdateEvent'
```

Ilustración 26. Request body (YAML)

Cabe mencionar, que la etiqueta "*schema*" puede ser definida in situ o en cambio podemos definir un modelo dentro de nuestro archivo y hacer una referencia a este. Es por ello por lo que se han realizado los modelos de todos los objetos que utilizan los endpoints para su funcionamiento, tanto los que tienen que ser enviados en las respuestas como los que son devueltos por el servidor, y también se han creado los modelos para todos los errores que devuelve la aplicación.

Para definir un modelo, se hace uso de la etiqueta "*components*", seguido por la etiqueta "*schemas*", seguido del nombre del objeto. A partir de aquí, podemos definir cuáles serán las propiedades de nuestro objetos y cuáles de ellas serán obligatorias. Además, dentro de estas definiciones también se pueden hacer referencia a otras definiciones en caso de que se trate de un objeto complejo.

```yaml
692 components:
693   schemas:
694     Login:
695       required:
696         - password
697         - username
698       properties:
699         username:
700           type: string
701           format: email
702           example: example1@gmail.com
703         password:
704           type: string
705           example: hola123
```

Ilustración 27. Componentes (YAML)

```yaml
    User:
      properties:
        id:
          type: string
          format: mongo-id
          example: 60acae8e2f799d228a4d4a85
        email:
          type: string
          format: email
          example: pedrojimenez@gmail.com
        memberId:
          type: string
          format: mongo-id
          example: 60acae8e2f799d228a4d4a85
        roles:
          type: array
          items:
            $ref: '#/components/schemas/Role'
        permissions:
          type: array
          items:
            $ref: '#/components/schemas/Permission'
```

Ilustración 28. Componentes con referencias (YAML)

Para finalizar la documentación obligatoria de un endpoint deberemos utilizar la etiqueta "*responses*" para mostrar los tipos de respuesta que nos devolverá nuestro servicio. Cada tipo de respuesta se define utilizando el código de respuesta del protocolo HTTP añadiendo una descripción y de manera opcional un esquema con la respuesta que nos devuelve nuestro endpoint.

```yaml
  /version:
    get:
      tags:
      - Version
      summary: API version
      description: |
        Return the lastest version of the API
      operationId: apiVersion
      responses:
        200:
          description: API Version
          content:
            text/plain:
              schema:
                type: string
                example: 1.0.25.r6-RC1
        500:
          description: Cannot verify the certificate
```

Ilustración 29. Respuestas del endpoint (YAML)

Una vez finalizado el documento, obtuvimos la documentación y la interfaz creada por Swagger del módulo de Federation.

4.4.2 Segundo sprint

El segundo sprint comenzó con la reunión para validar y verificar los cambios realizados en la iteración anterior. La conclusión a la que llegamos fue que el incremento obtenido era correcto, pero había que corregir una serie de errores.

En primer lugar, se pretende que el producto tenga una interfaz propia que permita añadir documentación relacionada con la empresa, declarar las políticas de uso del producto y añadir las nuevas funcionalidades, teniendo así una web centralizada donde se ofrezca tanto la documentación, como los fragmentos de código y los esquemas de los objetos que van incluidos en las peticiones a las API´s.

Además, inicialmente, los endpoints estaban divididos en función de los permisos y los roles de los usuarios siendo difícil de leer y dando poca accesibilidad y usabilidad a los usuarios. Es por ello por lo que se decidió cambiar las etiquetas del documento para dividir los endpoints en función del tipo de entidad que manejan y no en función de los permisos y los roles de los usuarios.

Finalmente, nos percatamos de que existen algunos endpoints que requieren que el cliente mande al servidor claves o que el usuario que realiza la petición tenga ciertos permisos para poder ejecutarse y es algo que debíamos documentar. Swagger no tiene mucho soporte en cuanto a la definición de roles y permisos, pero si para la definición de claves. Asimismo se decidió definir una serie de parámetros de seguridad que reflejaran la necesidad de esas claves o permisos para la ejecución de ciertos endpoints.

Para definir la seguridad de un endpoint se han creado una serie de "*scopes*" que definan estos parámetros de seguridad. Para la creación de estos parámetros se debe utilizar la etiqueta "*securitySchemes*" que actúa de envoltorio para todos las definiciones de seguridad. A continuación, ponemos un

nombre, y dentro de esta etiqueta definimos el tipo de seguridad, su nombre y donde va alojado.

En el ejemplo que se presenta a continuación, se define la seguridad "*idToken*" de tipo "*apiKey*", con el nombre de "COOKIE_ID_TOKEN" e indicando que va alojado en la cabecera de la petición.

```
917   securitySchemes:
918     idToken:
919       type: apiKey
920       name: COOKIE_ID_TOKEN
921       in: header
```

Ilustración 30. Seguridad de tokens o contraseñas (YAML)

Por otro lado, para definir un permiso debemos indicar un nombre, dentro de la etiqueta "*securitySchemes*" mencionada anteriormente, y dentro del nombre deberemos indicar el tipo de seguridad, una descripción y una serie de ámbitos, que en nuestro caso serán los permisos.

```
937   securitySchemes:
938     permissionManagement:
939       type: oauth2
940       description: This define all permisions required in Core API
941       flows:
942         authorizationCode:
943           scopes:
944             PERMISSION_MANAGEMENT: Permission required to add or remove user permissions
945           authorizationUrl: https://dev.moonshot.ceo/api/federation/login
946           tokenUrl: https://dev.moonshot.ceo/api/federation/token
```

Ilustración 31. Seguridad de permisos (YAML)

En la interfaz generada por Swagger se pueden diferenciar cuales son los endpoints que requieren de una seguridad especial porque se ve un candado al final de la definición, donde al pinchar podemos ver cuáles son los parámetros de seguridad requeridos para ese endpoint.

Ilustración 32. Interfaz de Swagger generada por el YAML

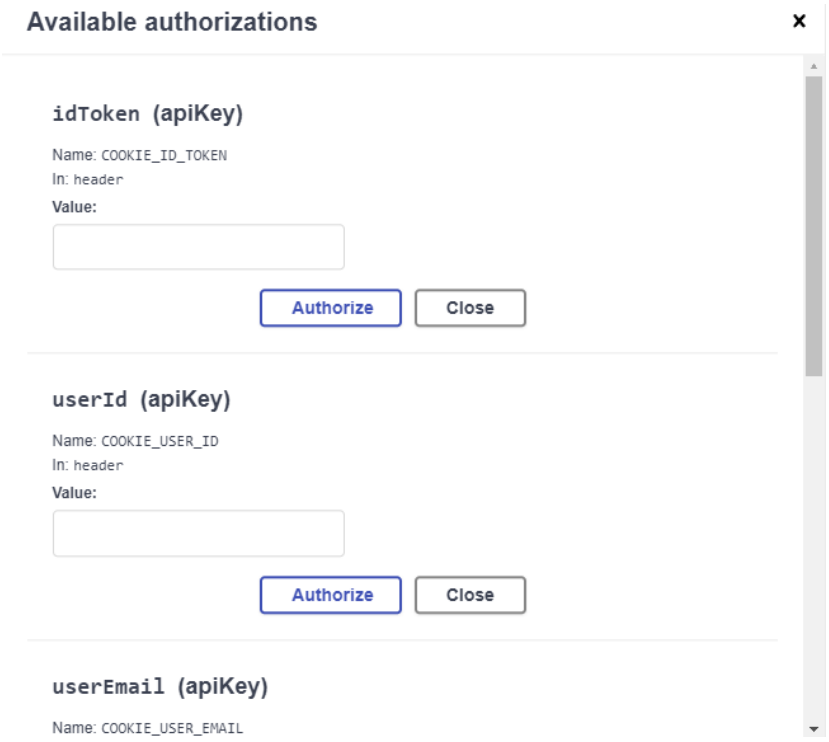


Ilustración 33. Permisos en la interfaz de Swagger

Por tanto, en este sprint, hemos conseguido integrar Swagger dentro la aplicación web que se ofrecerá a los clientes, además de refinar algunos detalles como la clasificación de los endpoints y añadir la seguridad de las API´s.

4.4.3 Tercer sprint

Este tercer sprint comenzó con la reunión para validar y verificar los cambios realizados en la segunda iteración. La conclusión a la que llegamos fue que el

incremento obtenido era correcto y una vez ya estaba bien definida la documentación de Federation y estaba hecha una primera versión de la página web que iba a alojar todas las funcionalidades ahora debíamos documentar el resto de los módulos.

En este sprint no se produjo ninguna novedad a la hora de implementar nuevas tecnologías simplemente fue una repetición de las operaciones realizadas para obtener los resultados de las iteraciones anteriores, pero ahora en el resto de los módulos de Moonshot.

Sin embargo, en este sprint nos dimos cuenta de que había algunas especificaciones creadas dentro de las API´s de Moonshot que no estaba bien definidas ya que incumplían las especificaciones que dicta OpenAPI. Por ejemplo, existían algunos endpoints de tipo GET en los que se enviaba un cuerpo en la petición, cosa que no es correcta según OpenAPI, pero si según por las especificaciones de API REST. Asimismo, se detectó que existía el mismo problema para todos los endpoints de tipo DELETE donde también requerían de un cuerpo en las peticiones para su correcto funcionamiento.

En el caso de las peticiones GET, este cuerpo simbolizaba un evento que únicamente tenía una propiedad “id” y su valor asociado, para que una vez llegara la información al endpoint este sea capaz de leer el cuerpo y obtener la entidad según el valor de la id.

Por otro lado, en el caso de las peticiones DELETE, el cuerpo también simbolizaba un evento que únicamente tenía la propiedad “id” y su valor asociado, para que una vez llegara la información al endpoint este sea capaz de leer el cuerpo y borrar la entidad de la base de datos en función del valor de la id.

Es por ello, que en esta iteración se tuvo que realizar una refactorización de las API´s cambiando todos los endpoints que no seguían las especificaciones de OpenAPI para que, en vez de recibir un evento en el cuerpo, simplemente

recibieran esa “id” a través de la ruta y una vez se ejecutara el método correspondiente a la llamada al endpoint se generara el evento.

En resumen, este sprint ha servido para tener documentados los módulos de Federation, Core y Community, cada una con la definición de sus endpoints, su seguridad y sus modelos. Además, hemos conseguido mejorar las propias API´s de Moonshot Innovation refactorizando los endpoints para que se ajusten a los estándares que dictamina OpenAPI.

4.4.4 Cuarto sprint

Este sprint comenzó, al igual que los anteriores, con la reunión que valida y verifica los cambios realizados en la tercera iteración. Decidimos que la documentación de los módulos era correcta y reflejaba a la perfección la manera en la que se debe utilizar cada endpoint. Sin embargo, pensamos que no es una documentación completa ya que el uso de la arquitectura *event sourcing* nos obliga a generar una serie de eventos que deberían estar reflejados en la documentación. Además, pensamos que sería interesante tener unos fragmentos de códigos de diferentes lenguajes de programación que ayuden a los desarrolladores externos a utilizar estas API´s en el desarrollo de sus propios módulos.

Para la realización de los ejemplos de código, inicialmente definimos que los lenguajes más usados por nuestros clientes iban a ser Javascript, Python y Java.

Para los ejemplos de Javascript, se hizo uso del método “*fetch*” proporcionado por el lenguaje de programación que nos permite realizar todo tipo de llamadas HTTP, incluyendo parámetros en las rutas y cuerpos en las peticiones. Beneficiándonos de esta manera de las herramientas que nos ofrece el lenguaje y evitando tener que incluir cualquier tipo de dependencia como una librería o *framework* a nuestro código.

Estos fragmentos de código se representan dentro de la aplicación como texto plano, pero para su construcción se hizo uso de la herramienta online “*PlayCode*”

que es una aplicación web que sirve de editor de código permitiéndonos escribir y ejecutar código para ver las respuestas en tiempo real.

A continuación, se presenta una imagen del uso de la herramienta, donde se hace una petición de tipo GET a la API del módulo de Federation para obtener la versión del mismo.

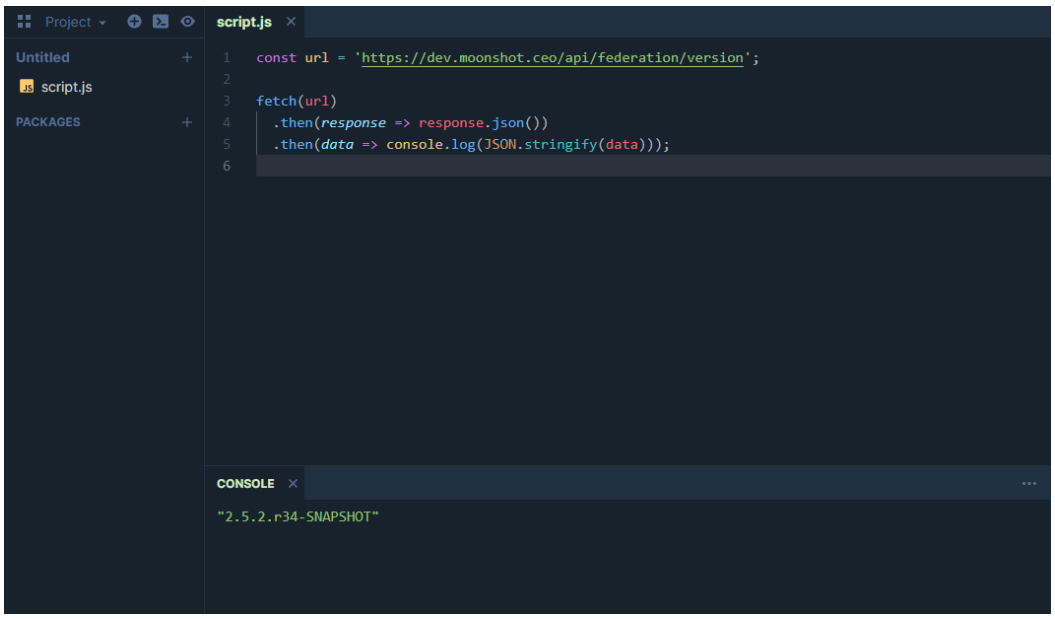


Ilustración 34. Ejemplo de código Javascript en PlayCode

Para los ejemplos de código de Python no se pudo hacer uso de ninguna herramienta interna del lenguaje, ya que este no ofrece la posibilidad de realizar llamadas HTTP. Es por ello por lo que se ha tenido que hacer uso de la librería "*request*" para poder realizar estas peticiones. No obstante, esta dependencia está reflejada en todos los ejemplos de la aplicación.

Para generar estos fragmentos de código, igualmente se utilizó un editor de texto online que nos permite ejecutar código en tiempo real. En este caso la herramienta se llama "*Programiz*" y a continuación se presenta una imagen del uso de la misma, donde se realiza una petición de tipo GET a la API del módulo de Core para obtener su versión.

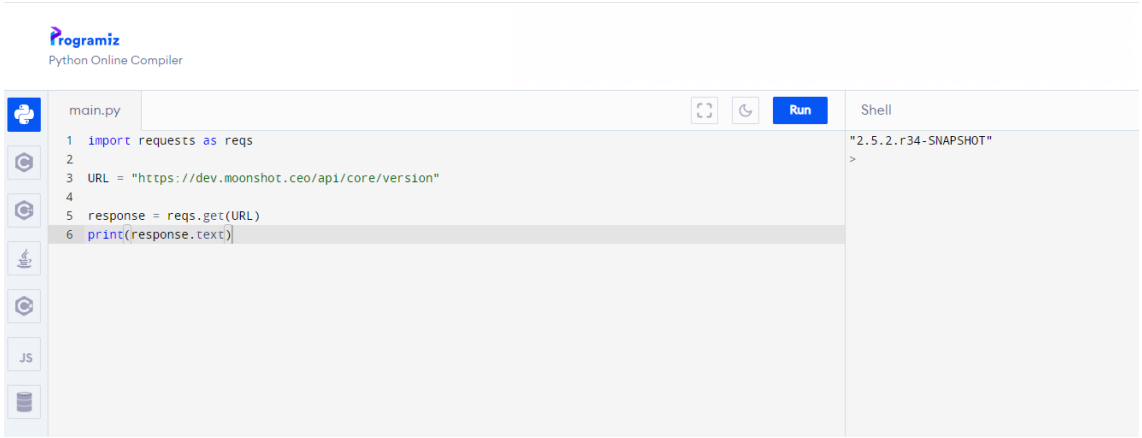


Ilustración 35. Ejemplo de código python en Programiz

Por último, para hacer los ejemplo de código de Java, primero hubo que crear las librerías a partir del código y posteriormente realizar su publicación en el repositorio de Maven Central, facilitando así el uso a los usuarios dando la posibilidad de poder importar las dependencias en un proyecto Maven a partir del archivo "pom.xml" sin la necesidad de tener las librerías descargadas. Además de facilitar su uso, también resulta una tarea sencilla actualizar la versión en caso de que la plataforma realice cambios en sus librerías.

Para generar estos fragmentos de código, se empleó el editor de texto "*IntelliJ*" que es una aplicación creada por la compañía "*JetBrains*" que nos permite crear un proyecto Maven para ejecutar código en Java.

A continuación, se presenta el uso de la herramienta, donde se hace una petición GET a la API del módulo de Community para obtener su versión.

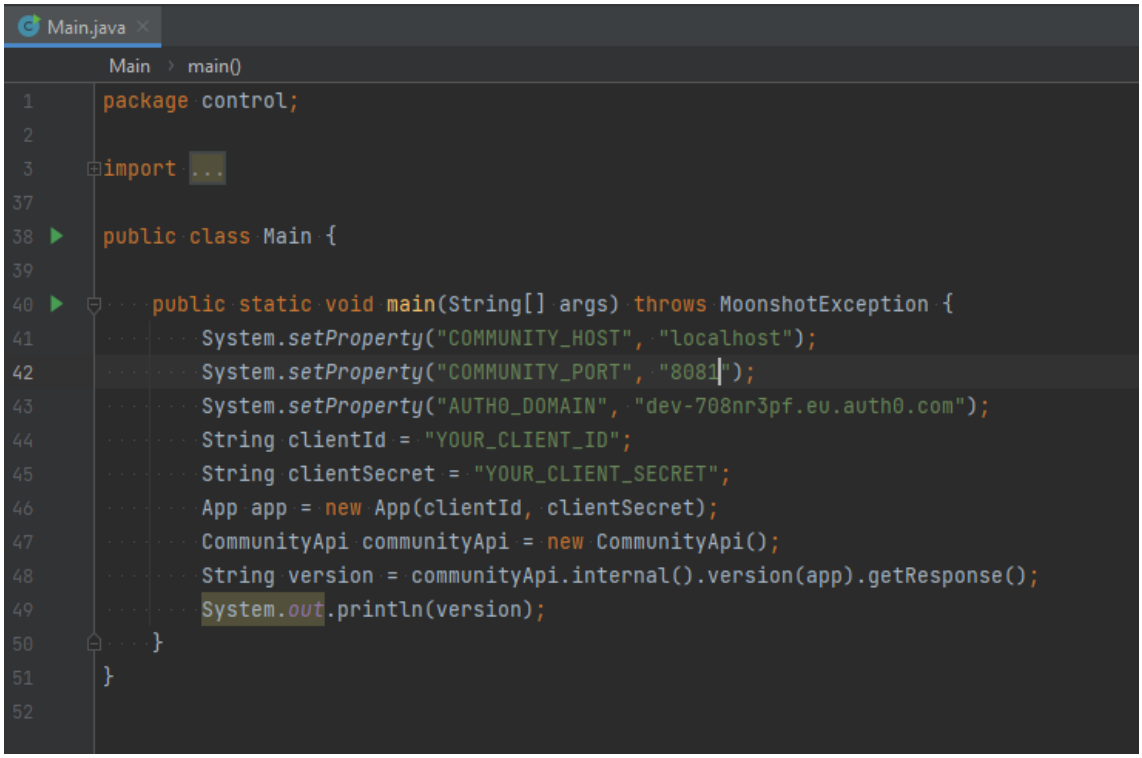


Ilustración 36. Ejemplo de código Java en IntelliJ

Una vez se habían creado e implementado los fragmentos de código dentro de la aplicación, la segunda parte del sprint se destinó a crear los modelos de los eventos necesarios para llevar a cabo la arquitectura de *event sourcing* en las peticiones a las API´s de Moonshot.

Para el desarrollo de esta funcionalidad se realizó un estudio previo de los eventos que se utilizaban en cada uno de los módulos para posteriormente poder clasificarlos en función de la entidad que maneja.

Cada evento está dividido en dos vistas: la primera vista refleja un ejemplo del evento, es decir, un objeto json con unos valores de ejemplo. En segundo lugar, nos encontramos con una vista que refleja el tipo de los valores puestos en el ejemplo.

No todos los valores utilizados, son valores primitivos, algunos hacen referencia a otros objetos dentro del modelo de Moonshot. Es por ello por lo que se han desarrollado también estos esquemas de las clases de modelo necesarias para enviar correctamente las peticiones.

4.4.5 Quinto sprint

Al igual que en el resto de las iteraciones, se validó y verificó que los cambios realzados en el anterior sprint eran correctos y se propusieron dos ideas para completar el producto.

En primer lugar, faltaba el módulo de Datahub por documentar, ya que no habíamos visto la necesidad porque este módulo constaba de dos únicos endpoints que eran usados de manera interna en Moonshot. Pero decidimos que, una vez implementados los esquemas de los eventos, era conveniente que un desarrollador externo también pudiera hacer uso de este módulo pudiendo crear sus propios eventos.

Esta tarea fue sencilla ya que consistía en repetir el trabajo realizado con los anteriores módulos siendo este el módulo más pequeño

Por otro lado, durante la revisión del producto vimos que sería recomendable que cuando un usuario visitara la documentación de un endpoint tuviera accesible un enlace para ver los ejemplos de código, por lo que utilizando la etiqueta de Swagger "*externalDocs*" pudimos mostrar en la interfaz de Swagger un enlace interno al ejemplo de código del endpoint en cuestión.

A continuación, se muestra un ejemplo de uso.

```
51    externalDocs:
52      description: Code Example
53      url: http://localhost:4200/code-examples/federation#version_get
```

Ilustración 37. Documentos externos (YAML)

Tras la integración de estas mejoras y la validación del producto por parte de los Scrum Masters y el Product Owner se dio por finalizado el desarrollo de este proyecto.

5. Conclusiones

5.1 Resultados

Los resultados obtenidos cumplen con las especificaciones y estándares de calidad que ofrece la empresa en su producto. Este proyecto ha estado en constante revisión por el Product Owner y por los Scrum Masters, siendo mejorado de manera continua para poder cumplir con los requisitos de los usuarios. En primer lugar, la documentación generada con Swagger, se presenta a través de una interfaz clara y sencilla, sin sobrecargas de elementos donde el usuario puede acceder a todo lo necesario con un máximo de tres clicks.

Para clarificar este punto, se pueden ver las siguientes imágenes de referencia: Ilustración 6, Ilustración 7, Ilustración 8, Ilustración 9.

Por otro lado, los fragmentos de código hechos en tres lenguajes de programación diferentes (Javascript, Java y Python) también nos aporta una interfaz clara y sencilla donde vemos tanto el código necesario para la llamada al endpoint por parte de un desarrollador así como un ejemplo de la respuesta que nos ofrece este endpoint en caso de que hayamos realizado la llamada a la API de manera correcta.

Esta funcionalidad queda reflejada en las siguientes imágenes: Ilustración 15, Ilustración 16, Ilustración 17.

La última funcionalidad implementada, clarifica la manera en la que se deben mandar los objetos en las peticiones a las API´s de Moonshot e indica al usuario cuales son los datos que se están guardando por el uso de la arquitectura de *event sourcing*. Además de tener un ejemplo de uso, se puede ver el tipado de cada propiedad y en caso de que el tipo de la propiedad sea un objeto del modelo de Moonshot, también se podrá ver un ejemplo de este y los tipos de sus propiedades.

Esta funcionalidad se muestra en las siguientes imágenes: Ilustración 10, Ilustración 11, Ilustración 12, Ilustración 13, Ilustración 14.

Por último, estas tres funcionalidades quedan envueltas en la página web, conformando así el nuevo módulo de *API Management*. Esta página nos permite presentar a los usuarios las tres funcionalidades implementadas como un producto único en lugar de tres distintos, haciendo mas sencilla su distribución y venta. Esto queda reflejado en las siguientes imágenes: Ilustración 3, Ilustración 4, Ilustración 5

5.2 Contribuciones

En esta sección se presentan a las personas que han hecho posible la realización de este trabajo.

David Suriol, dueño de la empresa de Moonshot Innovation, que ha sido el máximo responsable de llevar a cabo este proyecto. Comenzando por darme la posibilidad de realizar las prácticas extracurriculares lo cual me ha facilitado el desarrollo del producto porque ya tenía un amplio conocimiento de la lógica del negocio, hasta contar con su presencia en todas las reuniones que tuvimos durante el desarrollo del producto, ejerciendo la función de Product Owner y siendo el encargado de establecer la comunicación con los usuarios y *stakeholders*.

Iñigo Aramburu y Jose Juan, que han ejercido el rol de Scrum Managers y me han aportado sus conocimientos y experiencias para poder analizar, estructurar y ejecutar el proyecto de la mejor manera. Además de darme la posibilidad de acceder a los recursos e infraestructura de la empresa.

5.3 Experiencia personal

A nivel personal, el desarrollo de este proyecto me ha dado la posibilidad de enfrentarme a un proceso de desarrollo real aportándome innumerables ventajas de cara a enfrentarme al mundo laboral.

En primer lugar, pertenecer a un equipo de trabajo que prime el uso de las metodologías ágiles para el desarrollo del producto final me ayuda a mejorar como programador porque todas las funcionalidades que se implementan en la aplicación son revisadas por los tutores del proyecto y me ayudan a conseguir

una mejor calidad de código. Además, me introducen en el uso diario de un tipo de metodología que esta estandarizada y utiliza un sinfín de empresas.

Por otro lado, también me ha dado la capacidad de enfrentarme a un proyecto real que tenga que cumplir unos mínimos requisitos de calidad, ya que este proyecto a parte de tener una finalidad académica pretende llegar a ser usado por usuarios reales, siendo implementado en un aplicación real en funcionamiento.

Además de esto, me permite acercarme de primera mano a tecnologías con las que nunca había trabajado y no tenía conocimiento de estas antes de comenzar este trabajo como OpenAPI, Swagger, API Development. Y también me ha permitido mejorar habilidades en otro tipo de tecnologías de las que sí tenía conocimiento previo como Angular, Typescript, HTML o CSS.

En resumen, la sensación que tengo tras finalizar este proyecto es que he aprendido bastante a nivel teórico y he mejorado mis habilidades como programador, siendo capaz de enfrentarme a un proyecto real ya que siempre estuve presente y me tomaron en consideración a la hora de tomar las decisiones que afectaron al futuro del proyecto. Además, las personas involucradas en este producto me han aportado una gran cantidad de experiencias y conocimientos ya que han participado en diferentes proyectos y tienen el rodaje para desbloquear todos los problemas que me han ido surgiendo durante el desarrollo haciéndome aprender en todo momento.

5.4 Trabajo futuro

Este proyecto puede ser integrado en la plataforma en el estado en el que se encuentra, sin embargo, para que los usuarios puedan aprovecharlo al completo es necesario realizar primero unas refactorizaciones por parte de Moonshot Innovation. Esto se debe a que la empresa utiliza una clave única que permite verificar que los clientes que realizan las peticiones a las API´s están autorizados para ello. Sin embargo, los clientes externos a Moonshot que quieran realizar llamadas a las API´s desde sus códigos no pueden hacerlo porque no tienen esa clave. Es decir, para completar la gestión de las API´s de Moonshot, la empresa

debe proveer a cada usuario una clave que les permita y le otorgue los permisos para poder realizar llamadas a las API´s correctamente.

Por otro lado, esta documentación se actualizó hasta el momento de la finalización del desarrollo del proyecto, por lo que la propia necesidad de la empresa de seguir avanzando y desarrollar nuevas funcionalidades hace que el proyecto tenga la necesidad de estar en constante mejora y actualización. Siendo probable que al momento de la presentación de este trabajo este se encuentre desactualizado.

Además, en el futuro se podría considerar la idea de realizar unas mejoras de la estética del producto, ya que, aunque este se finalizó con la aprobación de todo el equipo (tanto *Scrum Managers* como *Product Owner*) y fue presentado así a algunos clientes, es la primera versión del mismo y puede recibir mejoras para adecuarse más a la estética de la plataforma.

Por último, en función de las necesidades de los usuarios y del crecimiento de la plataforma puede ser necesario la inclusión de incluir más lenguajes de programación en la exposición de los fragmentos de código. Aunque dependiendo del lenguaje de programación, a lo mejor se tienen que realizar trabajos extras como el desarrollo y publicación de librerías.

6. Competencias

Durante el desarrollo de este trabajo de fin de grado se han adquirido y perfeccionado algunas de las competencias exigidas por la Universidad de Las Palmas de Gran Canaria. Esta especie de contrato ayuda a los estudiantes a ser forzados a aprender nuevos conocimientos, permitiendo así mejorar técnicas y el desempeño como ingeniero informático.

El objetivo de la realización de este trabajo es enfrentarnos a un proyecto real para tener vivencias y experiencias mejorando ciertas facetas que los estudiantes no adquieren durante su periodo de formación. Contribuyendo así a la inserción en un mundo laboral real.

A continuación, se definen las competencias relacionadas con el Grado de Ingeniería Informática de la Universidad de Las Palmas de Gran Canaria.

En cuanto a las competencias generales, se definen las siguientes:

- **TFG01**. *Ejercicio original para realizar individualmente y presentar y defender ante un tribunal universitario, consistente en un proyecto en el ámbito de las tecnologías específicas de la Ingeniería en Informática de naturaleza profesional en el que se sinteticen e integren las competencias adquiridas en las enseñanzas*.

En cuanto a las competencias específicas, se definen las siguientes:

- **IS1**. *Capacidad para desarrollar, mantener y evaluar servicios y sistemas software que satisfagan todos los requisitos del usuario y se comporten de forma fiable y eficiente, sean asequibles de desarrollar y mantener y cumplan normas de calidad, aplicando las teorías, principios, métodos y prácticas de la ingeniería del software.*

  Este proyecto no solo se ha desarrollado con la supervisión de los tutores, asegurando en todo momento que se cumplen los requisitos definidos por los usuarios, sino que también se han tenido que llevar a cabo labores de mantenimiento ya que este trabajo se encuentra en continuo cambio por la propia evolución de la empresa y el negocio. Además, el código

generado para la cumplimentación de estos requisitos tiene una cierta calidad acorde a la experiencia adquirida en la empresa y a los conocimientos aprendidos en el grado.

- **IS2**. *Capacidad para valorar las necesidades del cliente y especificar los requisitos software para satisfacer estas necesidades, reconciliando objetivos en conflicto mediante la búsqueda de compromisos aceptables dentro de las limitaciones derivadas del coste, del tiempo, de la existencia de sistemas ya desarrollados y de las propias organizaciones*.

  Dado que este proyecto cuenta con la experiencia de usuarios reales, se ha podido conocer de primera mano si las necesidades de estos han sido satisfechas.

  Además, se ha podido asegurar que es un proyecto rentable en función de los costes derivados del mismo como el tiempo, el personal y los recursos ofrecidos por la empresa.

- **IS3**. *Capacidad de dar solución a problemas de integración en función de las estrategias, estándares y tecnologías disponibles.*

  Este proyecto satisface la necesidad que tiene la empresa de tener una gestión de las API´s de la plataforma, pudiendo integrar nuevas tecnologías y respetando todos los estándares que estás ofrecen.

  No solo aporta una nueva funcionalidad al producto creado por la empresa, sino que mejora considerablemente el valor del mismo pudiendo ofrecer una tecnología que ofrecen otras marcas mejor posicionadas en el mercado.

7. Bibliografía

[1] Colaboradores de Ticnegocios, *"Caminar con éxito hacia la Industria 4.0: Plataformas"*, [Online], Ticnegocios, 2021. Disponible en: https://ticnegocios.camaravalencia.com/servicios/tendencias/caminar-con-exito-hacia-la-industria-4-0-capitulo-20-plataformas/

[2] Tehreem Naeem, *"Definición de API REST: comprensión de los conceptos básicos de las API REST"*, [Online], Astera, enero 2020. Disponible en: https://www.astera.com/es/type/blog/REST-api-definition/#What-is-REST-API?

[3] Colaboradores de Swagger, *"What Is OpenAPI?"*, [Online], Swagger, 2021. Disponible en: https://swagger.io/docs/specification/about/

[4] Nina Osipova, *"Introducción a OpenAPI: ventajas e implementación"*, [Online], Sdos, julio 2020. Disponible en: https://sdos.es/en/node/8208

[5] Colaboradores de arquisoft, *"Event-Sourcing"*, [Online], Arquisoft, 2020. Disponible en: https://arquisoft.github.io/slides/course2021/seminars/DocEs3-06.pdf

[6] Colaboradores de Wikipedia, *"Gestión de API"*, [Online], Wikipedia, junio 2020. Disponible en:

https://arquisoft.github.io/slides/course2021/seminars/DocEs3-06.pdf

[7] Colaboradores de RedHat, *"Diferencias entre REST y SOAP"*, [Online], RedHat, abril 2019. Disponible en:

https://www.redhat.com/es/topics/integration/whats-the-difference-between-soap-rest

[8] Colaboradores de StackOverflow, *"¿Qué diferencia hay entre SOAP y REST?"*, [Online], StackOverflow abril 2019. Disponible en: https://es.stackoverflow.com/questions/76615/qu%C3%A9-diferencia-hay-entre-soap-y-rest

[9] Colaboradores de ProyectosAgiles, *"Qué es SCRUM"*, [Online], ProyectosAgiles, 2021. Disponible en: https://proyectosagiles.org/que-es-scrum/

[10] Manuel José Goncalves, *"¿Qué es Angular y para qué sirve?"*, [Online], Hiberus, octubre 2021. Disponible en: https://www.hiberus.com/crecemos-contigo/que-es-angular-y-para-que-sirve/

[11] Colaboradores de Wikipedia, *"Angular (framework)"*, [Online], Wikipedia, julio 2021. Disponible en: https://es.wikipedia.org/wiki/Angular_(framework)

[12] Colaboradores de Wikipedia, *"TypeScript"*, [Online], Wikipedia, mayo 2022. Disponible en: https://es.wikipedia.org/wiki/TypeScript

[13] Colaboradores de Wikipedia, *"HTML"*, [Online], Wikipedia, junio 2022. Disponible en: https://es.wikipedia.org/wiki/HTML

[14] Colaboradores de Wikipedia, *"CSS"*, [Online], Wikipedia, mayo 2022. Disponible en: https://es.wikipedia.org/wiki/CSS