



Escuela Técnica Superior de Ingeniería Informática

Máster de Ingeniería del Software:
Cloud, Datos y Gestión TI

Desarrollo y operación de un SaaS en función del pricing

Realizado por
Daniel Arellano Martínez

Tutorado por:
Prof. Dr. D. Jose María García Rodríguez
Prof. Dr. D. Antonio Ruiz Cortés

Sevilla, junio 2023

ÍNDICE

ÍNDICE	3
CONTROL DE VERSIONES	6
ABREVIATURAS Y ACRÓNIMOS.....	7
GLOSARIO DE TÉRMINOS	7
RESUMEN	8
ABSTRACT.....	8
PARTE I: INTRODUCCIÓN.....	10
1. Introducción.....	11
1.1. Introducción	11
1.2. Contexto del TFM	11
1.3. Motivación.....	12
1.4. Estructura del documento.....	12
2. Estudio previo	13
2.1. Objetivos.....	13
2.2. Metodología	13
2.3. Planificación	14
2.4. Esfuerzo y presupuesto	15
PARTE II: MATERIAS RELACIONADAS	17
3. Gestión de productos propios	18
3.1. Metodologías ágiles.....	18
3.2. Sprints.....	19
3.3. Ciclo de vida de un producto propio	20
4. Pricing	21
4.1. Precio.....	21
4.2. Limitaciones de capacidad (rates y cuotas).....	22
4.3. Funcionalidades (feature toggles).....	23
5. Microservicios.....	24
5.1. Fundamentos.....	24
5.2. API Gateway	25
5.3. Spring WebFlux.....	25
5.4. Spring Cloud	26

6. Frontend	27
6.1. Angular	27
7. Bases de datos	28
8. Despliegue	28
8.1. Okteto.....	28
8.2. AWS Lambda	28
8.3. AWS IAM.....	29
8.4. MongoDB Compass	29
PARTE III: SISTEMA DESARROLLADO	31
9. Historias de usuario	32
9.1. Sistema	32
9.2. Gestión de proyectos	33
10. Análisis del sistema.....	34
10.1. Diagramas UML.....	34
10.2. Mockups.....	37
11. Arquitectura del sistema	41
11.1. API Gateway.....	41
11.2. Microservicios.....	42
11.3. Bases de datos	43
12. Análisis de capacidad.....	44
13. Implementación del sistema	45
13.1. Entorno de desarrollo.....	45
13.2. Fichero de configuración de planes.....	51
13.3. Proyecto padre	52
13.4. Microservicios.....	54
13.5. Redireccionamiento.....	55
13.6. OpenAPI	58
13.7. Autorización y autenticación	62
13.8. JWT.....	62
13.9. Rates y cuotas	65
13.10. Feature toggles	67
13.11. Pricing	69
14. Validación	75
14.1. Validación de rates y cuotas	75
14.2. Validación de permisos.....	76

PARTE IV: CONCLUSIONES	79
15. Cumplimiento de objetivos.....	80
16. Esfuerzo y costes totales	80
16.1. Esfuerzo empleado	80
16.2. Costes totales.....	81
17. Mis conclusiones.....	82
PARTE V: BIBLIOGRAFÍA CONSULTADA	84
18. Bibliografía.....	85
19. Imágenes.....	89
20. Código	90
ANEXOS.....	92

CONTROL DE VERSIONES

N.º de edición	Fecha	Descripción
v0.1	04/07/2021	Creación de la plantilla.
v0.2	21/02/2023	Se añade "ABSTRACT".
v0.3	16/03/2023	Se añade "INTRODUCCIÓN".
v0.4	26/03/2023	Se añade "MATERIAS RELACIONADAS".
v0.5	27/03/2023	Se añaden las secciones: "Estudio previo" y "Ciclo de vida prod. propio".
v0.6	28/03/2023	Se añaden las secciones: "Historias de usuario" y "Análisis del sistema".
v0.7	28/03/2023	Se añaden las secciones: "Arquitectura del sistema".
v0.8	18/04/2023	Se añade la sección "Sprints" y explicación a los diagramas UML.
v0.9	19/04/2023	Se añaden secciones dentro de "Arquitectura del sistema" y "Implementación del sistema".
v1.0	20/04/2023	Se añaden secciones dentro de "Implementación del sistema".
v1.1	21/04/2023	Se añaden secciones dentro de "Implementación del sistema".
v1.2	23/04/2023	Se solucionan problemas con la navegación de los estilos del documento y se añaden secciones dentro de "Implementación del sistema".
v1.3	22/05/2023	Se añaden secciones del <i>pricing</i> .
v1.4	23/05/2023	Se realizan correcciones menores.
v1.5	29/05/2023	Se corrige la planificación y asuntos relacionados con el <i>pricing</i> .
v1.6	04/06/2023	Se realizan correcciones menores.
v1.7	08/06/2023	Se añade la sección "OpenAPI" y "Pruebas del sistema".
v1.8	12/06/2023	Se realizan mejoras con respecto la revisión realizada el 09/06/2023.
v1.9	13/06/2023	Se realizan mejoras con respecto la revisión realizada el 13/06/2023.
v2.0	14/06/2023	Se realizan comprobaciones finales y se exporta a PDF.

ABREVIATURAS Y ACRÓNIMOS

- **API:** *Application Programming Interface*, fragmento de Código que permite a diferentes aplicaciones comunicarse entre sí. En este proyecto, se usa concretamente una API REST.
- **AWS:** *Amazon Web Services*, servicios en la nube de *Amazon Inc.*
- **BBDD:** *Base de Datos*, programa capaz de almacenar gran cantidad de datos, relacionados y estructurados.
- **CAPEX:** *Capital Expenditure*, costo relacionado con la adquisición, desarrollo, implementación y gestión de una API.
- **CD:** *Continuous Delivery*, práctica en la que se preparan automáticamente los cambios en el código y se despliega en su entorno de producción.
- **CSS:** *Cascade Style Sheet* (hoja de estilos en cascada), para definir la apariencia de una página HTML.
- **DTO:** *Data Transfer Object*, objeto que contiene los atributos del objeto padre que se desean para el envío/recepción de estos mismos.
- **GCP:** *Google Cloud Platform*, servicios en la nube de *Alphabet Inc.*
- **HTML:** *HyperText Markup Language*, lenguaje de marcas con el que se describen páginas web.
- **JWT:** *Json Web Token*, tokens de acceso que permiten la identificación y privilegios (*claims*).
- **NPM:** *Node Package Module*, Sistema de gestión de paquetes por defecto para *Node.js*.
- **OpEX:** *Operation Expenditure*, costo relacionado con las operaciones y servicios.
- **REST:** *Representational State Transfer*, consiste en una interfaz que usa HTTP para realizar operaciones.
- **SaaS:** *Software-As-A-Service*, modelo de distribución de software en la nube.
- **SPA:** *Single Page Application*, aplicación web que está contenido en una única página.
- **SSO:** *Single-Sign-On*, procedimiento de autenticación que habilita el acceso a un usuario con una sola instancia de identificación.

GLOSARIO DE TÉRMINOS

- **Aplicación de escritorio:** aquella aplicación que se encuentra instalada en un dispositivo físico de sobremesa.
- **Interfaz gráfica:** representación visual de un programa ante un usuario.
- **Pricing:** estructura que define precio, límite de capacidad y funcionalidades de una API. En el contexto de este proyecto, se usará pricing multiplan, en el que un usuario podrá suscribirse a un plan en función de las necesidades que disponga.

RESUMEN

En los últimos años, ha aumentado el uso de metodologías ágiles sobre las metodologías en cascadas o tradicionales, principalmente por la gran adaptación a cambios que existe en el entorno empresarial y, en concreto, en consultoría software. Consciente de la necesidad, se desarrolló una solución o aplicación en 2021, con el objetivo de gestionar las prácticas ágiles de una organización.

Esta propuesta inicial carecía de ningún enfoque económico, por lo que se pretende rediseñar la aplicación, con el objetivo de convertirla en un producto comercialmente viable y comercializarlo a corto plazo.

Para lograr este propósito, se ha implementado un modelo de *pricing multiplan* que consta de 3 planes diferenciados, los cuales abordan aspectos relacionados con el precio, limitaciones de capacidad y funcionalidades. Cada uno de estos, está relacionado con la viabilidad y capacidad del producto a desarrollar.

Además, se ha llevado a cabo un análisis de capacidad y la determinación del coste de operación (OpEX), verificando que el *pricing* establecido previamente es económicamente viable para la infraestructura basada en microservicios y *API Gateway* en la que se encuentra desplegado el producto.

Por último, indicar que se han cumplido todas las expectativas y que el producto se encuentra en fase de producción.

ABSTRACT

In recent years, the use of agile methodologies has increased over waterfall or traditional methodologies, mainly due to the great adaptation to changes that exist in the business environment and, specifically, in software consulting. Aware of the need, a solution or application was developed in 2021, with the objective of managing the agile practices of an organization.

This initial proposal lacked any economic approach, so it is intended to redesign the application, with the aim of turning it into a commercially viable product and commercialize it in the short term.

To achieve this purpose, a multi-plan pricing model has been implemented, consisting of 3 differentiated plans, which address aspects related to price, capacity limitations and functionalities. Each of these is related to the viability and capacity of the product to be developed.

In addition, a capacity analysis and the determination of the operating cost (OpEX) has been carried out, verifying that the pricing previously established is economically viable for the infrastructure based on microservices and API Gateway in which the product is deployed.

Finally, it should be noted that all expectations have been met and that the product is now in production.

PARTE I: INTRODUCCIÓN

1. Introducción

1.1. Introducción

La gestión de proyectos es un proceso crítico para el éxito de cualquier empresa, y en la actualidad, muchas empresas han adoptado metodologías ágiles para mejorar la eficiencia en la gestión de proyectos. Sin embargo, muchas de estas empresas enfrentan problemas al intentar implementar estas metodologías de manera eficiente y efectiva, lo que puede llevar a una falta de control y seguimiento, lo que finalmente puede resultar en retrasos en los proyectos, costes elevados y una mala calidad del producto final.

Con el fin de abordar estos desafíos, se ha desarrollado un Trabajo de Fin de Máster que se enfoca en diseñar e implementar una aplicación de gestión de metodologías ágiles que incorpora una amplia gama de herramientas para la gestión de proyectos, *sprints*, historias de usuario, tareas, *kanban*, esfuerzos, diagramas de *Gantt*, gestión de incidencias e informes. Además, se han considerado principios de escalabilidad, costes, rendimiento y bajo mantenimiento para garantizar una solución eficiente y efectiva para las empresas.

La aplicación busca brindar una solución completa para la gestión de proyectos ágiles, incorporando una amplia gama de herramientas para la planificación, seguimiento y gestión de proyectos, además de la gestión de incidencias y la generación de informes. Se ha prestado especial atención a principios de escalabilidad, costes, rendimiento y bajo mantenimiento, para garantizar que la aplicación sea capaz de manejar grandes proyectos y una gran cantidad de usuarios, y que sea fácil de mantener y actualizar con costes razonables.

1.2. Contexto del TFM

El presente trabajo tiene sus antecedentes en el TFG realizado en 2021 como parte de la carrera de Ingeniería Informática del Software en la Universidad de Sevilla. En dicho trabajo se desarrolló una aplicación de gestión de proyectos que incluía funcionalidades como gestión de proyectos, *sprints*, tareas, *kanban*, esfuerzos, y que estaba diseñada y estructurada bajo un enfoque monolítico con un API Rest como backend y un frontend.

Posteriormente, en 2022, como parte del TFM del Máster Universitario en Seguridad de la Información y las Comunicaciones, se ampliaron las funcionalidades de la aplicación para incluir la gestión de activos, vulnerabilidades y riesgos.

Desde la presentación del TFG, se han estado desarrollando nuevas funcionalidades en la aplicación de manera análoga, como la incorporación de diagramas de *Gantt* y la gestión de incidencias.

En 2023, se adquirieron conocimientos sobre el despliegue en *Okteto* (permitted proceder con su correspondiente despliegue) y conceptos básicos del *pricing*. Además, me estuve informando sobre trabajos de investigación relacionados con el *pricing* como el [plan nacional IRIS](#) y [PERSEO@BUBO](#).

Actualmente el producto está pensado para ser usado de forma personal, pero no como un producto comercial preparado para obtener beneficios, que sea escalable y pueda satisfacer las necesidades anteriormente descritas.

1.3. Motivación

Durante la realización del *TFG* (Trabajo de Fin de Grado, Grado en Ingeniería Informática del Software) y *TFM* (Trabajo de Fin de Máster, Máster en Seguridad de Sistemas de la Información) desarrollé una solución o aplicación que integra la gestión de proyectos, sprints, tareas, esfuerzos y ciberseguridad de manera ágil, sencilla, atractiva y altamente funcional. Como el resto de los servicios *SaaS*, su comercialización implica la creación de un *pricing* asociado a él.

La solución creada carece de *pricing*, por lo que se va a ofrecer una solución a este problema y, a corto plazo, comercializarla. Un buen análisis del pricing y su *OpEX* me permitirá determinar la mejor solución para obtener el máximo rendimiento económico posible antes de su producción.

1.4. Estructura del documento

La estructura tomada para desarrollar este documento ha sido una combinación de la planteada en el *TFG*, *TFM* del máster anterior y los tutores del TFM (**Prof. Dr. D. Jose María García Rodríguez** y **Prof. Dr. D. Antonio Ruiz Cortes**), la cual se estructura en 5 partes:

1. RESUMEN / ABSTRACT

Resumirá de forma breve (no más de media página) el proyecto, de forma que facilite una lectura más rápida y abstracta a lo aquí presente.

2. PARTE I: INTRODUCCIÓN

Se realiza una breve introducción sobre el proyecto, indicando metodología, planificación, presupuesto y un análisis de mercado.

3. PARTE II: MATERIAS RELACIONADAS

Se indicará el contexto de estudio en cada una de las áreas implicadas: gestión de productos propios, limitaciones en APIs, pricing, microservicios, frontend, bases de datos y despliegue.

4. PARTE III: SISTEMA DESARROLLADO

Se describirá la información que se ha tenido en cuenta para la construcción de la solución, así como las decisiones que se han tomado.

5. PARTE IV: CONCLUSIONES

Se informará si se han cumplido los objetivos planeados en el proyecto y del tiempo/recursos empleados.

6. PARTE V: BIBLIOGRAFÍA

Indicará las referencias tanto de las imágenes como de las secciones tomadas de otras fuentes ajenas a este proyecto.

7. ANEXOS

Indicará los documentos relacionados con la creación de este trabajo.

2. Estudio previo

En esta sección, se detallarán los conceptos a tener en consideración antes de comenzar la realización del proyecto.

2.1. Objetivos

En esta sección, se indicarán las metas/objetivos a desarrollar en el proyecto. Siendo estos los siguientes:

OBJ-01	Enfocar producto comercialmente
Descripción	Consiste en establecer una serie de necesidades y expectativas del mercado, que permitan generar beneficios comerciales.
Subobjetivos	<ul style="list-style-type: none"> • OBJ-01-A: Establecer un <i>pricing multiplan</i> • OBJ-01-B: Adaptar la aplicación por el plan suscrito del usuario • OBJ-01-C: Construir una arquitectura basada en microservicios

OBJ-02	Analizar el coste de operación
Descripción	Consiste en realizar un estudio sobre el coste de operación en función de las posibles licencias adquiridas y de las limitaciones establecidas.

2.2. Metodología

Este proyecto se fundamenta en un proceso iterativo e incremental, de forma que cada 2 semanas se tenga una reunión de seguimiento con los tutores asignados. Esta forma permite realizar pequeños incrementos que permitan una entrega en el tiempo previsto. Su funcionamiento puede verse en la imagen inferior en la que se representa la metodología empleada.

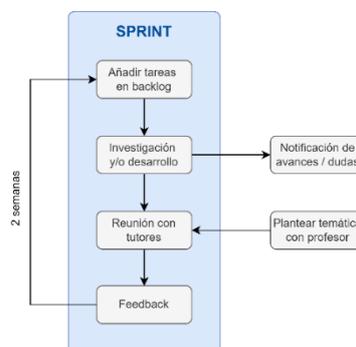


Imagen 1: Proceso metodológico en el TFM

Dentro de las figuras pueden apreciarse los siguientes puntos a destacar:

- **Plantear temática con profesor:** se menciona qué se va a realizar en el proyecto, alcance y fijar entregas de proyecto.

- **Reunión con tutores:** se realizará una retrospectiva en la que se analizarán los progresos y posibles mejoras. Así mismo, se determinará nuevas tareas a realizar.
- **Feedback:** se obtendrá una conclusión de la reunión que será añadida en el backlog.
- **Añadir tareas en backlog:** se añaden al documento *Word "backlog.docx"* los puntos tratados en la reunión y posibles mejoras y puntos a tratar en el siguiente sprint.
- **Investigación y/o desarrollo:** se llevarán a cabo las tareas en el backlog.

Este proceso se realizará hasta que el entregable esté en un estado de calidad óptimo para proceder con su entrega. Cuando esto pase, los documentos implicados, el código fuente y un vídeo de demostración será enviado en la correspondiente plataforma de entrega.

2.3. Planificación

En primera instancia, se decidió segmentar la planificación por sprints y que en cada uno se realizase acciones concretas (investigar, documentar, desarrollar, implementar...) en función de la fase en la que se encuentre. Dado que en cada uno de los sprints en la vida real se realizan múltiples acciones de forma simultánea, se decidió dedicarle una dedicación por acción diferente a cada uno de los sprints y adaptar la **Metodología de Proceso Unificado (RUP)** al contexto del proyecto a desarrollar.

Como se ha mencionado previamente, los sprints se llevarán a cabo cada 2 semanas, comenzando en la segunda mitad de noviembre y, terminando en la última de diciembre. Supone un total de 14 sprints distribuidos a lo largo de 6 meses.

La adaptación del RUP para este proyecto implica la segmentación del Trabajo de Fin de Máster (TFM) en las diferentes partes del RUP de la siguiente manera:

- **Inicio:** se organizan las ideas, planificación y objetivos del proyecto.
- **Elaboración:** se investiga una solución al problema y se plantea un caso teórico.
- **Construcción:** se crea una implementación práctica sobre el trabajo a analizar y se establece el *pricing*.
- **Transición:** se elabora la documentación final.

A continuación, se representará la evolución del porcentaje de dedicación con el paso del tiempo:

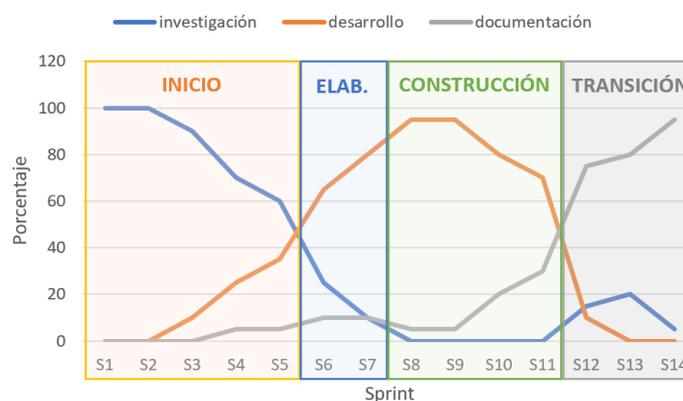


Imagen 2: Planificación del proyecto mediante RUP

2.4. Esfuerzo y presupuesto

En este apartado, se va a realizar una estimación sobre las horas previstas para la realización del proyecto, desglosado en función de las tareas a realizar.

Dada la envergadura del proyecto y, una aproximación de 300 horas para su desarrollo es conveniente estudiar qué porcentaje o división se le va a dedicar a cada uno de los desgloses. A continuación, se dividirá el tiempo en los siguientes aspectos:

- **Investigación (50 horas):** esto implicaría la búsqueda de soluciones en el mercado, tecnologías, herramientas, comparación entre propuestas de cloud y análisis de mercado. Dado que mi experiencia previa en el desarrollo del TFG y el TFM (del anterior máster), considero que suele emplear 16,67% del tiempo total.
- **Desarrollo (175 horas):** dado que es el mayor peso del proyecto, suelen encontrarse errores a última hora y un alcance de diferentes objetivos interrelacionados, creo conveniente una estimación del 58,33%.
- **Documentación (60 horas):** supone un uso razonablemente más ligero que el desarrollo (por problemas imprevistos normalmente) y hasta el momento se han empleado un total de 15 horas y supone aproximadamente un 33% del total, podría decirse que el total de documentación representaría un total de 60 horas.
- **Otros (15 horas):** son estimaciones de otros aspectos no contemplados en los puntos anteriores, como puede ser reuniones con el tutor o revisión de una reunión previa grabada. Solamente en reuniones con el tutor, se planifican un total de 10 reuniones (se debe de realizar en 5 meses, con 2 "sprints" por cada mes) que tienen una duración media de 45 min de duración, 15 min de preparación y 15 min de recogida de información, dando un total de 12,5 horas solamente dedicadas a las reuniones. Si a eso se añaden otros conceptos es entendible que se alcance las 15 horas indicadas.

Dado que tenemos un esfuerzo cerrado en tiempo, el presupuesto se determinará en base a las horas empleadas y en función del coste por hora del desarrollo.

Para el cálculo de coste por hora de desarrollo se han tenido en cuenta los siguientes aspectos:

- **Personal (20,83€/hora):** se ha tomado como referencia el sueldo medio de un puesto similar al que se realiza en este trabajo, que sería "Cloud Architect". Para este puesto, según [Glassdor](#) supone un sueldo promedio de 49.721€ brutos/año, suponiendo que un perfil sin experiencia de este tipo sea ligeramente inferior hasta los 40.000€ brutos/año y se trabaje de forma autónoma (ignorando el coste empresa por parte del trabajador), supondría 55€/hora. [1]

$$\text{Coste} = \frac{40000\text{€ al año}}{12 \text{ meses} * 20 \text{ días al mes} * 8 \text{ horas al día}} = 20,83 \text{ € la hora}$$

- **Ordenador (0,08€/hora):** teniendo en cuenta que el ordenador se compró hace 1 año y su valor es de 625€ (teniendo en cuenta los componentes nuevos, los antiguos se descartan ya que su antigüedad es anterior a 4 años), y un periodo de amortización según convenio de 4 años.

$$\text{Coste} = \frac{625\text{€}}{48 \text{ meses} * 20 \text{ días al mes} * 8 \text{ horas al día}} = 0,0813\text{€ la hora}$$

- **Otros (1.3€/hora):** engloba al consumo de electricidad, agua, internet, sillas, mobiliario de oficina... Se tienen en cuenta estos aspectos dado que es más barato el teletrabajo para un trabajador por cuenta ajena.

$$\text{Coste electricidad} = \frac{125€ \text{ al mes}}{20 \text{ días al mes} * 8 \text{ horas al día}} = 0.78€ \text{ la hora}$$

$$\text{Coste internet} = \frac{35€ \text{ al mes}}{20 \text{ días al mes} * 8 \text{ horas al día}} = 0.21€ \text{ la hora}$$

$$\text{Coste mobiliario} = \frac{50€ \text{ al mes}}{20 \text{ días al mes} * 8 \text{ horas al día}} = 0.31€ \text{ la hora}$$

Sumando todo lo anterior, tenemos un coste total de 22,21€ por hora de trabajo. Tras estimar que se va a realizar en 300 horas, podría indicarse que un total antes de impuestos de 6663€. En caso de aplicar el IVA, resultaría una **cuantía total de 8062,23€**.

PARTE II: MATERIAS RELACIONADAS

3. Gestión de productos propios

La gestión de productos propios es el proceso de planificación, desarrollo y comercialización de un producto diseñado y creado por una empresa para satisfacer las necesidades de sus clientes y usuarios. Es un proceso que implica múltiples tareas y responsabilidades, que van desde la identificación de oportunidades de mercado, la definición de los requerimientos del producto, el desarrollo y lanzamiento del producto, hasta el seguimiento y mejora continua del mismo.

La gestión de productos propios involucra la toma de decisiones importantes y la coordinación de múltiples equipos y departamentos, incluyendo el equipo de desarrollo, el equipo de marketing, el equipo de ventas, y el equipo de soporte al cliente, entre otros. Es un proceso crítico para el éxito de una empresa, ya que un producto exitoso puede impulsar el crecimiento y la rentabilidad de la empresa, mientras que un producto que no cumpla con las expectativas del mercado puede tener consecuencias negativas en la reputación y los resultados financieros de la empresa.

Por lo tanto, es esencial que la gestión de productos propios se realice de manera cuidadosa y estratégica, considerando las necesidades y expectativas de los clientes y usuarios, y manteniendo un enfoque constante en la mejora continua del producto a lo largo de su ciclo de vida.

3.1. Metodologías ágiles

En este apartado, se explicará en qué consiste las metodologías ágiles y sus tipos.

Son metodologías enfocadas en la colaboración y la flexibilidad, y se basan en el desarrollo iterativo e incremental. Existen numerosas metodologías ágiles, aunque lo más común es adaptar una o más de ellas al proceso de creación de productos propios.

A continuación, se va a explicar las metodologías que suelen emplearse para la creación de productos software:

- **Scrum:** se enfoca en el trabajo en equipo y en la entrega de software funcional de manera iterativa e incremental. El equipo de desarrollo trabaja en ciclos cortos de tiempo llamados sprints, durante los cuales se enfoca en entregar software funcional.
- **Kanban:** se enfoca en la entrega constante de software y en la mejora continua. Se utiliza un tablero Kanban para visualizar el flujo de trabajo y se establecen límites en el trabajo en progreso para evitar la sobrecarga del equipo.
- **XP (eXtreme Programming):** se enfoca en la entrega de software de alta calidad de manera rápida y eficiente. Se enfoca en la colaboración constante entre el equipo de desarrollo y los clientes, y en la mejora continua del proceso de desarrollo.
- **Lean:** se enfoca en maximizar el valor del software para el usuario final y eliminar cualquier desperdicio en el proceso de desarrollo. Busca entregar software funcional de manera rápida y eficiente, sin comprometer la calidad.
- **DevOps:** se enfoca en la colaboración y la integración continua entre los equipos de desarrollo y operaciones. Busca eliminar los silos y fomentar la comunicación constante para poder entregar software de manera más rápida y eficiente.

- **Crystal:** se enfoca en adaptarse a la naturaleza del proyecto y del equipo de desarrollo. Se utiliza una serie de prácticas ligeras y se ajusta la metodología según las necesidades específicas del proyecto.
- **FDD (Feature-Driven Development):** se enfoca en la entrega de características (features) específicas del software de manera rápida y eficiente. Se utiliza un enfoque basado en características para el desarrollo de software y se enfoca en la colaboración constante entre el equipo de desarrollo y los clientes.
- **DSDM (Dynamic Systems Development Method):** Esta metodología se enfoca en la entrega de software de alta calidad de manera rápida y eficiente. Se utiliza un enfoque iterativo e incremental y se enfoca en la colaboración constante entre el equipo de desarrollo y los clientes.

3.2. Sprints

Son periodos de tiempo fijo en el que un equipo de *Scrum* trabaja para completar una cantidad de trabajo establecida. [2]

Cada uno de los sprints estará compuesto de los siguientes elementos [3]:

- **Iniciativas:** son colecciones de épicas que están orientadas en un objetivo común.
- **Épicas:** son grandes agrupaciones de trabajo, que a su vez se pueden desglosar en historias de usuario.
- **Historias de usuario:** son requisitos cortos o peticiones que se realizan con perspectiva del usuario final [4]. Suele estar compuesta por un título y las 3C's (Card, Conversation & Confirmation), siendo estas últimas asociadas a los campos "As I" (rol interesado), "To do" (qué hay que realizar) y "For" (necesidad de la funcionalidad).
- **Milestone:** es un punto específico dónde se marca una etapa del desarrollo. Por ejemplo: Primer despliegue (16 de diciembre). [5]



Imagen 3: Iniciativas, épicas e historias de usuario según Atlassian [3] [4]

3.3. Ciclo de vida de un producto propio

PLM (Product Lifecycle Management) o ciclo de vida de un producto software es el conjunto de etapas que atraviesa un proyecto de software desde su ideación hasta su finalización. Dado que no hay estándar al respecto, existen diferentes propuestas de *PLM*. En concreto, *SAP* propone las siguientes etapas [6]:

1. **Concepto y diseño:** se plantea la idea, define de requisitos, define un plan de negocio y análisis de viabilidad del producto.
2. **Desarrollo:** se diseña el producto, construye, valida y creación de primera versión de producto (versión piloto).
3. **Producción y lanzamiento:** se obtiene feedback de la versión piloto y se pone en producción al mercado.
4. **Servicio y soporte:** periodo de tiempo en el que se ha garantizado el soporte.
5. **Declive:** deja de ser rentable el producto y se retira del mercado.

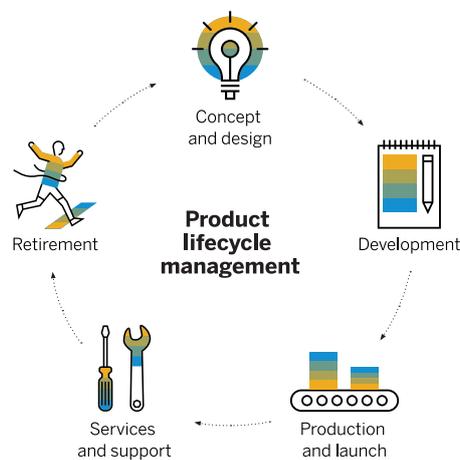


Imagen 4: Propuesta PLM de SAP [6]

4. Pricing

El *pricing* es una estructura que define una serie de elementos comunes, como son el precio, límites de capacidad y funcionalidades por cada uno de los planes que se definen (*pricing multiplan*).

Como ejemplo, quería destacar el *pricing* propuesto para esta aplicación (véase explicación en sección 13.11).

Características	Free	Premium	Enterprise
Pago anual	GRATIS	6 \$ / mes	18 \$ / mes
Pago mensual	GRATIS	8 \$ / mes	24 \$ / mes
Periodo de prueba	N/A	30 días	30 días
Peticiones API / seg. (rate)	5	10	100
Peticiones API / min. (cuota)	100	200	2000
Nº proyectos máx.	2	20	200
SSO	NO	NO	SI
SLA / Soporte	NO	SI	SI
Tiempo respuesta	N/A	3 días	1 día
Disponibilidad (tasa de acierto)	N/A	95%	98%
Características	Proyectos	Proyectos Seguridad CMDB RRHH	Proyectos Seguridad CMDB RRHH

Tabla 1: Ejemplo de plan de precio (propuesta del proyecto)

Como puede verse, se trata de un pricing multiplan en el que existen 3 planes de precio, en el que se indican los precios, limitaciones y funcionalidades a las que se puede acceder el usuario por medio de una suscripción.

A continuación, se profundizará en cada uno de los conceptos indicados previamente.

4.1. Precio

Es la estrategia utilizada para establecer el precio de un producto o servicio de software cuyo objetivo es encontrar el precio que maximice la rentabilidad del producto.

Aquí hay algunos modelos comunes son:

- **Precio único:** cliente paga una tarifa única por el software. Este precio puede ser fijo o basado en el número de usuarios o dispositivos. Este modelo es comúnmente utilizado por software de consumo.

- **Freemium:** software se ofrece de forma gratuita con funcionalidades limitadas, y se cobra por funcionalidades adicionales o versiones premium del software. Este modelo es comúnmente utilizado por software de consumo y empresarial.
- **Suscripción:** cliente paga una tarifa periódica (mensual, trimestral o anual) por el acceso al software. Este modelo es comúnmente utilizado por software empresarial o por servicios en la nube.
- **Licencias:** software se licencia a una empresa o entidad gubernamental para su uso en un número limitado de dispositivos o usuarios. Este modelo es usado por software empresarial.
- **Dinámico:** precio estará determinado en base a la demanda o al tiempo. Este modelo es usado por servicios en la nube (PaaS, SaaS...), APIs...

Es importante tener en cuenta que el precio del software no sólo incluye los costos de desarrollo, sino también los costos de soporte técnico, actualizaciones y mantenimiento. Por lo tanto, el precio debe ser suficiente para cubrir estos costos y generar ganancias para la empresa. Para ello, normalmente se suele obtener el *OpEX* de una transacción, observando solamente los costes de infraestructura.

4.2. Limitaciones de capacidad (rates y cuotas)

Las limitaciones de uso o de capacidad son aquellas restricciones para garantizar un uso equitativo y óptimo de los recursos y servicios, garantizando disponibilidad y viabilidad económica. Existen 2 tipos de limitaciones de uso: *rates* y *cuotas*.

Los *rates* son limitaciones en el uso de una API, servicio web u otro tipo de recurso software que establecen la cantidad máxima de solicitudes que se pueden realizar a una API o servicio web en un periodo corto de tiempo (sobre 1 segundo), mientras que las cuotas se aplican en un periodo más largo de tiempo (desde 1 hora hasta 1 año).

Son importantes para evitar el uso excesivo y garantizar la estabilidad y la disponibilidad del servicio.

Además, pueden estar relacionados con los planes de precio, en el que cada plan tendrá rates y cuotas establecidas. En algunos casos, pueden aplicarse “*overage costs*”, que son cargos adicionales que se aplican cuando se exceden estos límites de uso.

Dado que se está trabajando con arquitectura basada en microservicios, es importante mencionar que LAMA (*Limitation-Aware Microservices Architectures*) consiste en un enfoque arquitectónico que se centra en el diseño y desarrollo de sistemas basados en microservicios, teniendo en cuenta las limitaciones y restricciones inherentes a cada servicio. [7] [8]

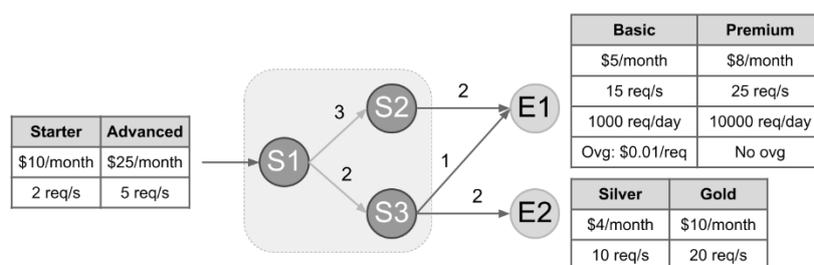


Imagen 5: Ejemplo de LAMA propuesto por Rafael Fresno [7]

4.3. Funcionalidades (feature toggles)

También conocidos como “*feature flags*”, son componentes software que permiten activar o desactivar a voluntad funciones específicas de la aplicación. Pueden agruparse en “*feature groups*”.

Según Martín Fowler, se pueden identificar los siguientes tipos de *feature toggles* [9]:

- **Release toggles:** permite activar o desactivar características en función de la versión de software que se esté ejecutando.
- **Experiment toggles:** permite activar funcionalidades solamente a una pequeña porción de los usuarios, normalmente con el propósito de realizar una prueba y evaluar su impacto antes de lanzarla a todos los usuarios. También pueden llamarse “*canary release*”.
- **Ops toggles:** son toggles que se usan para su uso en infraestructura. Normalmente suele usarse para indicar una mayor escalabilidad en el sistema cuando se considere necesario.
- **Permission or user toggles:** estos toggles se utilizan para activar o desactivar características en función del usuario que esté utilizando el software. Normalmente suele estar relacionado con un plan asociado.

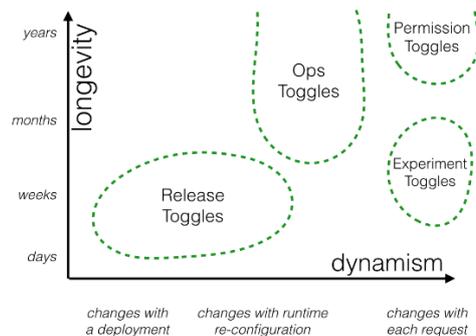


Imagen 6: Tipos de feature toggles [10]

Pueden implementarse en forma de configuración mediante ficheros, mediante bases de datos o mediante un servicio externo como [ConfigCat](#), [Split](#), [AWS AppConfig](#) o [Flagsmith](#).

5. Microservicios

En este apartado, se hablará de cómo los microservicios son una arquitectura de software que permite crear aplicaciones escalables y flexibles, y dos tecnologías que se usan hoy en día.

5.1. Fundamentos

La arquitectura en microservicios se basa en la **creación de aplicaciones a partir de pequeños servicios** independientes, cada uno de los cuales se centra en realizar una tarea específica. Esto permite que los componentes sean más pequeños, autónomos y escalables que en una arquitectura monolítica. Cada microservicio puede estar escrito en un lenguaje de programación diferente y por un equipo diferente. [11] [12]

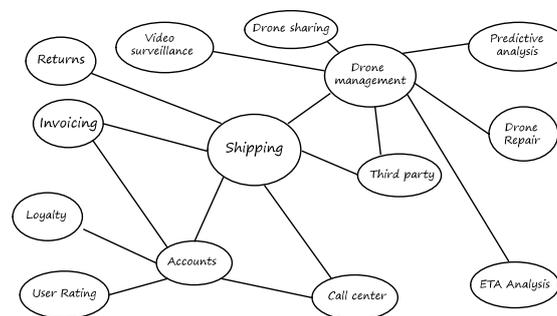


Imagen 7: Ejemplo de segmentación en microservicios [12]

Normalmente para la identificación de los microservicios se usa la técnica DDD (Domain-Driven Design), en el que cada microservicio tiene que estar diseñado en cuanto a un ámbito de negocio o de dominio concreto. Según la recomendación de Microsoft, se utiliza el siguiente proceso para la identificación de los microservicios [12]:

1. **Analizar dominio:** se empieza por analizar el modelo de negocio de la empresa, con el propósito de conocer los requisitos funcionales.
2. **Definir bounded contexts:** existen múltiples modelos de dominio en cada contexto delimitado, que representan subdominios.
3. **Definir entidades, agregados y servicios de dominio**
4. **Identificar microservicios:** se usa el paso anterior para identificar los microservicios.



Imagen 8: Pasos en Drone Delivery [12]

5.2. API Gateway

Es un componente que se utiliza para gestionar el tráfico de entrada y de salida de los microservicios. Actúa como un enrutador de tráfico interno que redirige la petición al microservicio indicado y realiza labores de identificación y autorización de recursos, balanceo de carga, caché y transformación de datos. [13]

Se usa principalmente por las siguientes circunstancias:

1. **Simplificación de la gestión de microservicios:** se facilita la implementación y el mantenimiento de la arquitectura de microservicios.
2. **Control de acceso y seguridad:** permite políticas de seguridad y autenticación de forma uniforme y transparente a todos los microservicios.
3. **Escalabilidad:** puede manejar grandes volúmenes de tráfico y balancear la carga.
4. **Monitoreo y análisis:** permite la detección de problemas y el análisis de la actividad de la aplicación.

En la imagen a continuación, se podrá ver un ejemplo en el que el *API Gateway* recepciona las peticiones y en función de estas las conduce al microservicio apropiado.

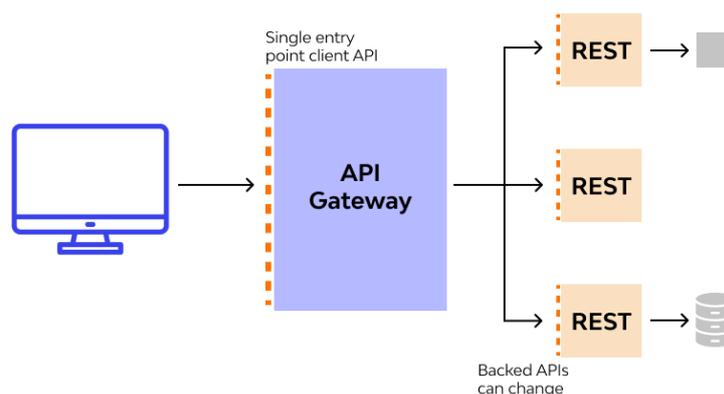


Imagen 9: Funcionamiento de un API Gateway [13]

5.3. Spring WebFlux

Spring Framework es un framework de desarrollo de aplicaciones web y/o APIs que usa los lenguajes de programación *Java* o *Kotlin*, y el sistema de dependencias de *Maven* o *Gradle*. *Spring WebFlux* es una variante que permite la programación reactiva, en el que se usa una pila de procesos no bloqueante o, en otras palabras, que desacopla las entradas y las salidas. Esto permite entre otras muchas otras cosas, que el sistema sea más escalable y eficiente, tal y cómo puede verse en la gráfica.

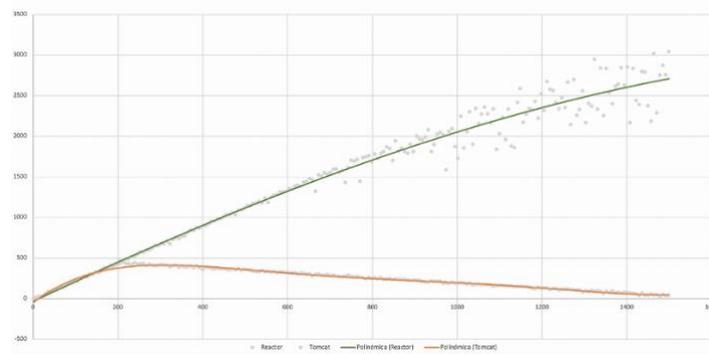


Imagen 10: Número de peticiones máximas en modelo reactivo vs modelo bloqueante [14]

5.4. Spring Cloud

Es un plugin que incluye un conjunto de herramientas y bibliotecas basadas en Spring Framework. Permite la configuración distribuida, gestión de (micro)servicios, balanceo de carga y tolerancia a fallos. Permite la creación de aplicaciones escalables y resistentes. [15]

Algunos de los módulos más importantes son:

- **Spring Cloud Azure, Alibaba, AWS, GPC, Kubernetes:** gestionan los servicios cloud.
- **Spring Cloud Gateway:** permite gestionar un *API Gateway* de forma no bloqueante (requiere *Spring WebFlux*).
- **Spring Cloud Security:** gestiona la seguridad de la aplicación mediante OAuth2, SSO, interceptores...
- **Spring Cloud Netflix:** permite la gestión del servidor Eureka, permitiendo así la sincronización de los diferentes microservicios.

6. Frontend

Los servicios frontend son un conjunto de tecnologías y herramientas que se utilizan para desarrollar y mantener la **interfaz de usuario** de una aplicación web, de escritorio o móvil. Esta debe intentar proporcionar una experiencia de usuario atractiva y fácil de usar. Existen numerosas tecnologías como *Angular*, *React* y *VueJS*, así como gestores de paquetes como *npm* y *yarn*.

6.1. Angular

Framework de código abierto para aplicaciones web desarrollado en *TypeScript*, usado para la creación y mantenimiento de aplicaciones de una sola página o *SPA (Single Page Application)*. Es mantenido por *Google Inc.* y la comunidad.



Imagen 11: Logotipo de Angular

Entre sus **ventajas** se encuentran:

- **Framework completo:** a diferencia de librerías como *React* o *VueJS* (librerías de *JavaScript*), es un framework completo que proporciona una estructura organizada de trabajo.
- **Altamente escalable:** se centra en la modularidad, por lo que la aplicación se puede dividir en pequeñas partes llamadas módulos.
- **Documentación detallada:** dispone de gran variedad de ejemplos de código y todos sus componentes.
- **Uso de Typescript:** permite una mejor sintaxis, búsqueda de errores,
- **CLI de Angular:** facilita el trabajo mediante una serie de herramientas que permiten la creación de componentes, ejecución...
- **Soporte de Google:** garantiza que se podrá recibir soporte en tiempo y forma, con un líder en el sector como es Google.

Entre sus **desventajas** se encuentran:

- **Curva de aprendizaje pronunciada:** al ser complejo, es más difícil de aprender y utilizar todas las características.
- **Dependencias adicionales:** usa otras dependencias adicionales como *RxJS* y *Zone.js*.

Por otra parte, incluye una librería de visualización gráfica llamada [Material Angular](#), en la que se podrán crear contenidos siguiendo la estética descrita por *Google* en *Material Design*.

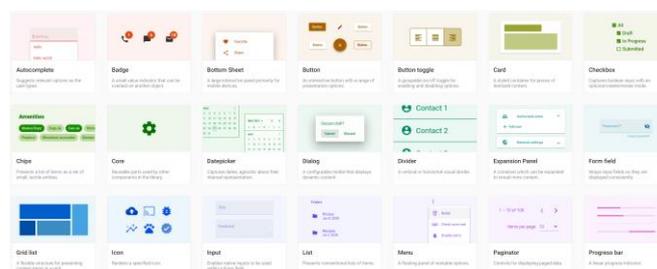


Imagen 12: Componentes de Material Angular [16]

7. Bases de datos

Una base de datos es un conjunto de datos relacionados entre sí, organizados y almacenados en un sistema de informático. Se utilizan para almacenar y gestionar grandes cantidades de información de manera eficiente y confiable.

Existen 2 tipos de sistemas de gestión de bases de datos (*DBMS*): *SQL* (se enumeran en tablas referenciadas) y *NoSQL* (utilizan otros patrones de diseño). En función de los requerimientos de la aplicación en cuestión, puede elegirse un *DBMS* u otro.

Los sistemas *SQL* cumplen los principios de diseño *ACID* (Atomicidad, Consistencia, Aislamiento y Durabilidad), mientras que los sistemas *NoSQL* cumplen los principios *CAP* (Coherencia, Disponibilidad y Tolerancia a la dispersión). Existen varios tipos de *NoSQL*, cada uno suele centrarse en 2 aspectos de *CAP*, es decir: C-A, C-P y A-P. [17] [18]

8. Despliegue

En este apartado se definirá el concepto de despliegue y las algunas de las tecnologías más usadas hoy en día.

El despliegue es el proceso de hacer que una aplicación esté disponible para usarse. Normalmente, estos suelen estar automatizados mediante *CD* (Despliegue Continuo), permitiendo así una mejor calidad, reducción de tiempo y riesgos.

Existen numerosas plataformas que se usan para el despliegue, entre las más importantes se encuentran: *AWS* (*Amazon Web Services*), *Microsoft Azure*, *GCP* (*Google Cloud Platform*), *DigitalOcean*, *IBM Cloud* y *Alibaba Cloud*.

8.1. Okteto

Okteto es una **plataforma de desarrollo** en la nube que permite a los desarrolladores crear, ejecutar y depurar aplicaciones en *Kubernetes* sin tener que preocuparse por la complejidad de la infraestructura. Esto permite entre otras cosas, que los desarrolladores puedan centrarse en el código y la funcionalidad de la aplicación, sin preocuparse por la configuración y el mantenimiento de la infraestructura subyacente. Además, es bastante fácil de usar y dispone de plan gratuito.

8.2. AWS Lambda

AWS Lambda es un servicio de cómputo **sin servidor** que permite a los desarrolladores ejecutar código en la nube sin tener que administrar servidores. El servicio se encarga automáticamente de todo lo demás, como aprovisionamiento y escalado de servidores, y cobra solo por petición recibida.

AWS Lambda es compatible con varios lenguajes de programación, como *Node.js*, *Python*, *Java*, *Go* y *C#*, y algunos de sus frameworks, como *Spring* en *Java*. Además, se integra con varios servicios de AWS, como *Amazon S3*, *Amazon DynamoDB* y *Amazon IAM*.

8.3. AWS IAM

AWS IAM (Identity and Access Management) es un servicio de AWS que permite a los administradores de cuenta **controlar y administrar el acceso** a los servicios y recursos de AWS. Los administradores pueden crear y administrar usuarios, grupos y roles, y definir políticas de acceso que especifiquen qué usuarios y grupos pueden hacer en los recursos de AWS.

Además, permite la integración con *Active Directory* (muy valorado por el SSO de *Microsoft*) y otros sistemas de identidad empresarial, lo que permite a los administradores mantener una única fuente de autenticación para todos los recursos de AWS.

8.4. MongoDB Compass

MongoDB Atlas es un servicio de **base de datos en la nube** completamente administrado por *MongoDB*, que permite a los desarrolladores crear y escalar fácilmente clústeres de *MongoDB* en la nube sin tener que preocuparse por la infraestructura.

Ofrece una variedad de características y beneficios para los desarrolladores, incluyendo la capacidad de escalar de forma elástica, la seguridad avanzada, la disponibilidad garantizada, la compatibilidad con múltiples nubes y la gestión de respaldos automáticos. Dado que este servicio está desplegado en las plataformas de *Google Cloud Platform*, *Microsoft Azure* y *Amazon Web Service*, se puede indicar el proveedor, así como la distribución de los clústeres.

PARTE III: SISTEMA DESARROLLADO

9. Historias de usuario

En este apartado, se detallarán las diferentes historias de usuario que establecerán el funcionamiento del sistema. Serán agrupados según funcionalidad o propósito, así como añadidos conforme aumente el tiempo.

9.1. Sistema

Representa todas las historias de usuario que deben de cumplirse en todo el sistema.

US1-01	Crear planes de precios
Como	Jefe de la empresa
Quiero	Crear planes de precios que incluya precio, <i>feature toggles</i> o funcionalidades, <i>rates</i> y <i>cuotas</i>
Para	Obtener beneficios

US1-02	Garantizar escalabilidad
Como	Jefe de la empresa
Quiero	Garantizar escalabilidad y mantenibilidad
Para	Evitar caídas en el rendimiento del servicio

US1-03	Interfaz de usuario simple
Como	Jefe de la empresa
Quiero	Crear una interfaz de usuario simple y bonita, que sea responsive y fluida
Para	Atraer más clientes

US1-04	Acceso a las APIs
Como	Jefe de la empresa
Quiero	Dar acceso a los clientes a nuestra API, de forma que puedan ver su documentación en Swagger y usarla.
Para	Atraer a nuevos clientes que les interese usar las APIs

9.2. Gestión de proyectos

Representa las historias de usuario en las que un Jefe de Proyectos o Project Manager pueda estimar y llevar el seguimiento de las tareas que se realizan mediante metodologías ágiles.

US2-01	Gestión de proyectos
Como	Project Manager
Quiero	Crear proyectos a los que se le pueda asignar unos usuarios con unos roles
Para	Asignar recursos a un proyecto concreto

US2-02	Gestión de sprints
Como	Project Manager
Quiero	Adaptarme a la metodología ágil y crear procesos iterativos mediante sprints, que integren entre otras cosas <i>milestones</i> y <i>epics</i>
Para	Asignar recursos a un sprint concreto

US2-03	Visualización de un kanban
Como	Project Manager
Quiero	Ver las tareas asignadas a los usuarios ordenadas mediante unas columnas, así como una barra que indique el tiempo imputado con respecto el esperado (verde indica excedente y rojo un exceso)
Para	Monitorizar en tiempo real es estado de las tareas y del tiempo empleado en cada una

US2-04	Visualización de un diagrama Gantt
Como	Project Manager
Quiero	Realizar un cronograma con las tareas descritas en el backlog
Para	Optimizar el desempeño del conjunto de las tareas

US2-05	Imputación de esfuerzos
Como	Integrante del equipo
Quiero	Imputar el tiempo dedicado a una tarea
Para	Mejorar las estimaciones realizadas previamente

Estas son un planteamiento teórico para la distribución de la arquitectura basada en microservicios y *API Gateway*, no será implementada.

10. Análisis del sistema

En esta sección, se indicarán los modelos de análisis que se han llevado a cabo durante el desarrollo de la aplicación.

10.1. Diagramas UML

Consiste en una serie de diagramas de clases para estructurar los datos con el objetivo de organizar cómo estos van a ser introducidos en las respectivas BBDD.

- **Usuarios y proyectos**

Trata la relación entre usuarios y los proyectos asociados a este. Dicha asociación se realiza mediante un rol que identificará los recursos a los que puede acceder el usuario para el proyecto mencionado.

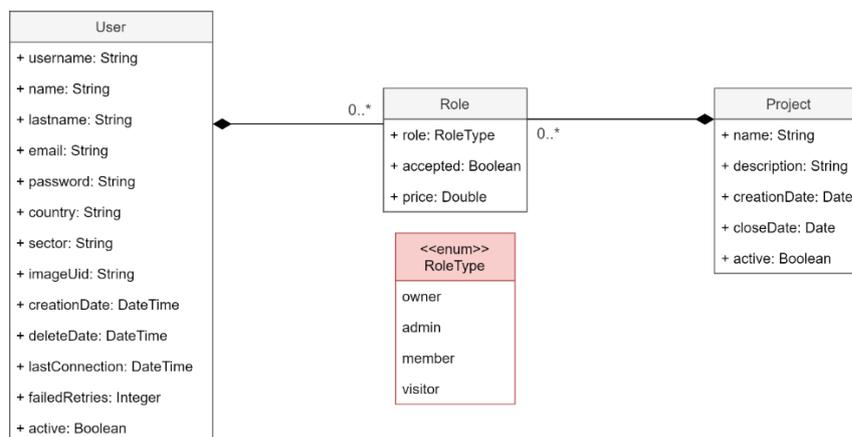


Imagen 13: UML usuarios y proyectos

- **Empresa**

Representa la relación entre la empresa y los usuarios. Cada empresa tendrá asociados una serie de departamentos, que a su vez tendrá una serie de roles asociados a múltiples usuarios.

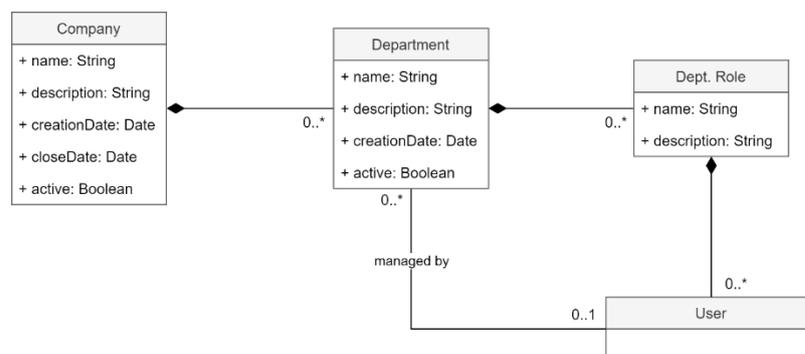


Imagen 14: UML empresa

• **Suscripción**

Modela las limitaciones que tiene un usuario asociadas en función del plan que haya pagado. En esta ocasión una empresa o el propio usuario puede tener una suscripción que haya contratado, la cual tendrá un periodo de validez y estará ligada a un plan.

En esta ocasión, el plan indicará *rates* y cuotas asociadas al usuario, así como su relación con limitaciones de dominio (por ejemplo: no podrá ser dueño de más de 10 proyectos) y los grupos de funcionalidades a los que podrá acceder. Cada grupo de funcionalidades controlará el acceso a determinadas funcionalidades indicadas en la aplicación. La definición de planes, límites de dominio y *feature groups*, se definirá en un fichero externo tal y como se menciona en la sección 13.2.

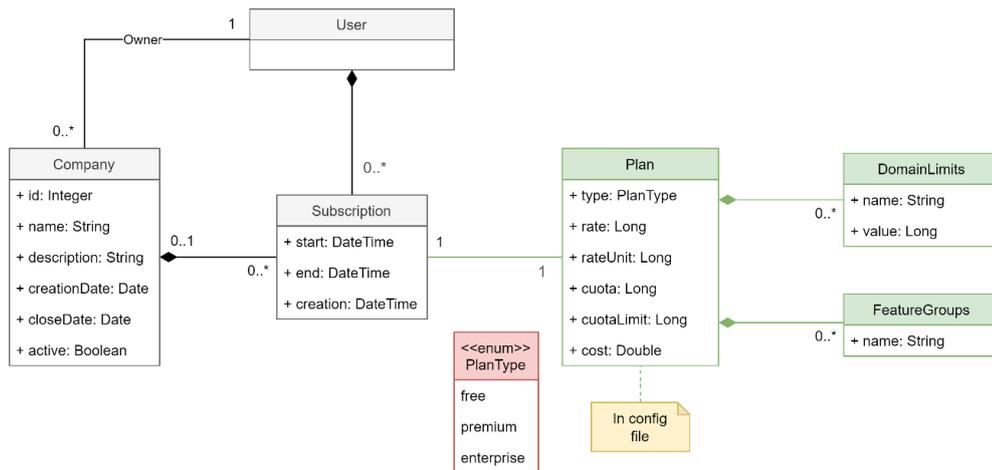


Imagen 15: UML suscripción

• **Sprint**

Modela la segmentación de periodos de tiempo en metodología ágil de *Scrum*. Cada sprint tendrá un número asociado y periodos de actividad. Cada uno de estos, podrá tener a su vez *Milestones*, *Epics* e historias de usuario.

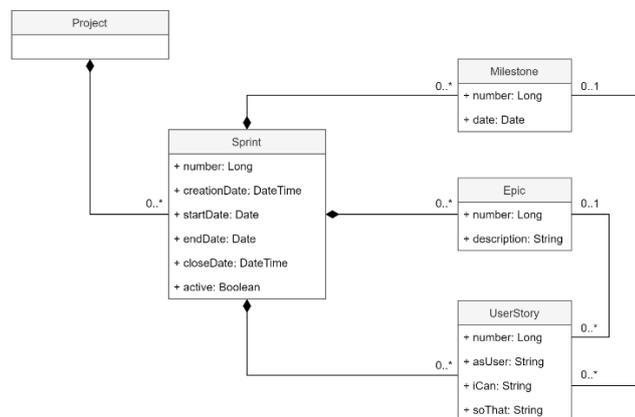


Imagen 16: UML sprint

• **Kanban**

Representa aquellas entidades relacionadas con la creación de un tablero basado en columnas y tareas. Una columna puede representar los estados en los que se puede encontrar una tarea (por ejemplo: “Por hacer”, “En progreso”, “En testing” o “Finalizado”). Una tarea representa la unidad mínima de trabajo y vendrá acompañada de una descripción y tiempo estimado. A su vez, podrá tener asociadas unas etiquetas (clasifican la tarea), subtareas asociadas (formularios que indiquen cuando la tarea esté completa), tareas hijas/padres, *milestones*, *epics* e historias de usuario.

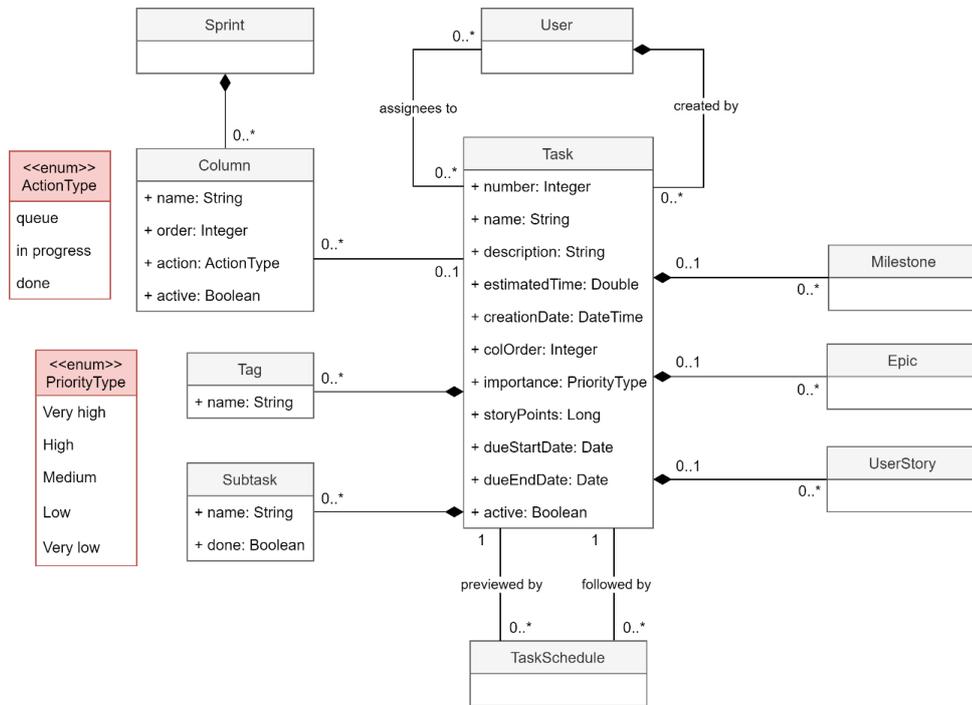


Imagen 17: UML kanban

• **Esfuerzo**

Es aquella entidad encargada de contabilizar el tiempo empleado de un usuario en específico en una tarea existente. Para ello, relacionará la tarea y el usuario, estableciendo una fecha de inicio y una fecha de fin.



Imagen 18: UML efforts

10.2. Mockups

Consiste en crear un prototipo de interfaz gráfica antes de empezar a construir/desarrollar el producto. Esto ha ayudado a la creación del proyecto fundamentalmente al establecer los requisitos, el alcance que se iba a tener y una primera idea de la interfaz a llevar a cabo durante el desarrollo. Para su desarrollo, se ha usado *Figma* debido a su previo uso, facilidad de uso y estética similar a la que se podrá general al final.

A continuación, se explicarán los distintos mockups por secciones:

- **Acceso**

Consiste en una pantalla que permitirá al usuario acceder al sistema insertando el email y su

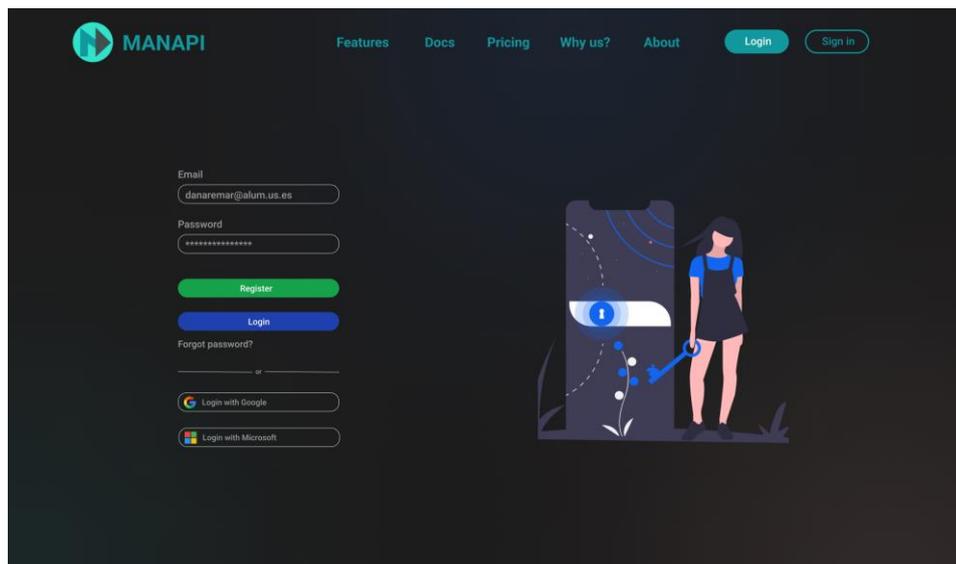


Imagen 19: Mockup de acceso

- **Proyectos**

Permitirá crear agrupaciones de personas que trabajen en los mismos objetivos comunes. El usuario podrá crear, eliminar y editar (si dispone de permisos) sprints u otros elementos.

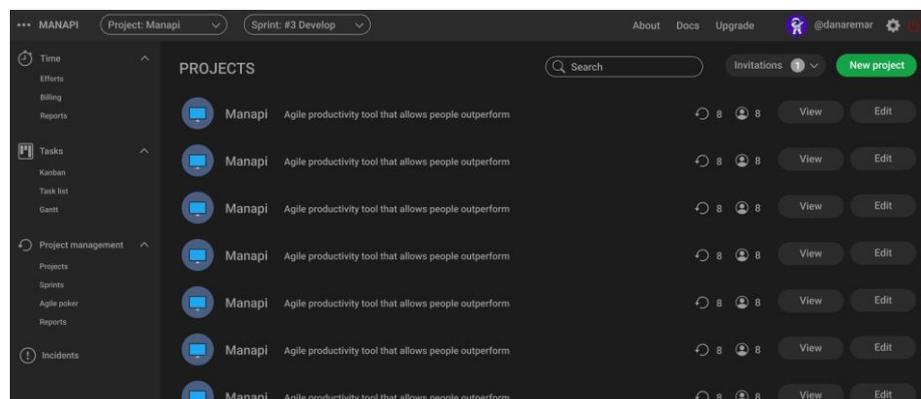


Imagen 20: Mockup listado de proyectos

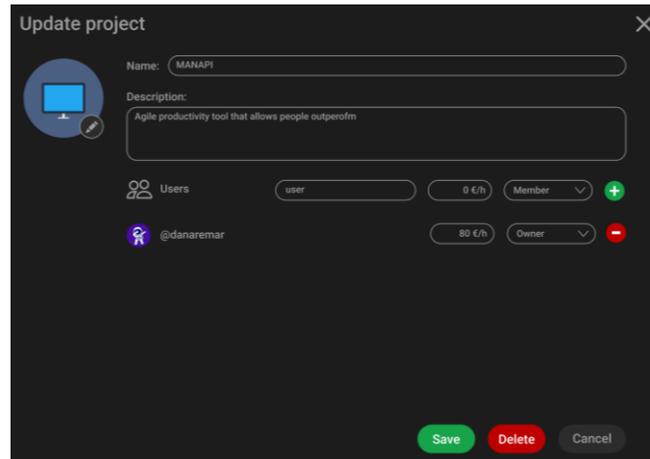


Imagen 21: Mockup actualizar proyecto

- **Sprints**

Cada sprint representaría una subdivisión de tiempo en el que se divide un proyecto. En esta vista se mostrarían el listado de estos para un proyecto dado, así como arrancar y finalizar el sprint. Podrá añadirse un sprint y editarlo si se disponen los permisos necesarios.

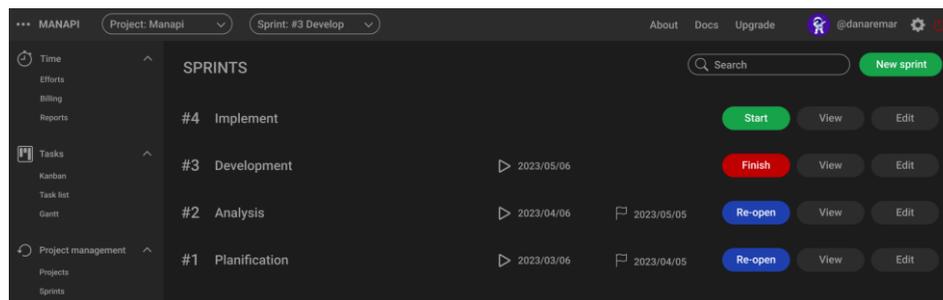


Imagen 22: Mockup listado de sprints

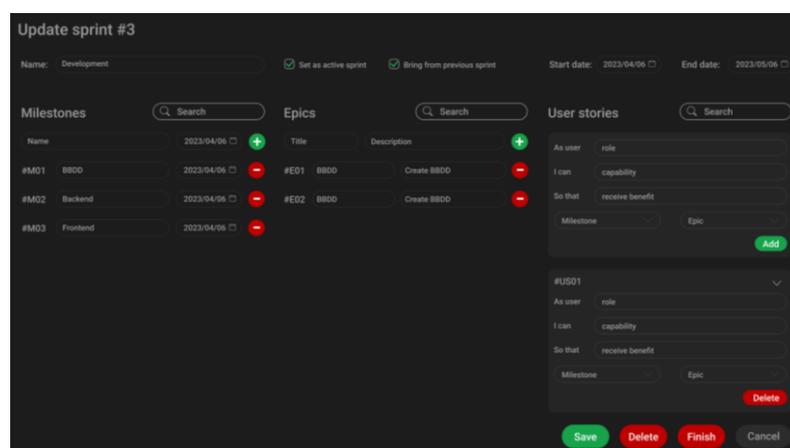


Imagen 23: Actualizar sprint

- **Kanban**

Permitirá al usuario crear columnas y tareas, tal y como dicta la práctica de tableros Kanban para metodologías ágiles. El usuario podrá filtrar en función del sprint que desee ver, gestionar (añadir, editar y eliminar) las columnas y gestionar las tareas, así como desplazar las tareas entre distintas posiciones dentro de su misma columna u otra y comenzar/parar esfuerzos.

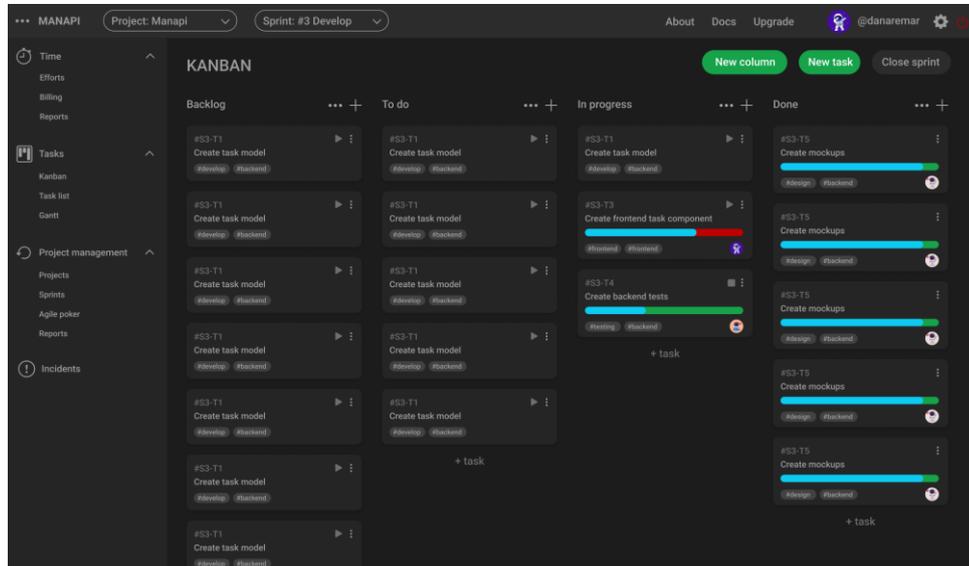


Imagen 24: Mockup mostrar kanban

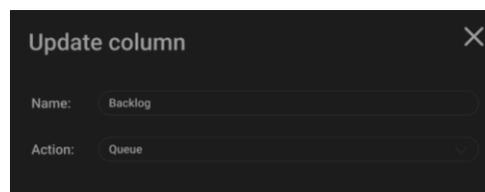


Imagen 25: Mockup actualizar columna

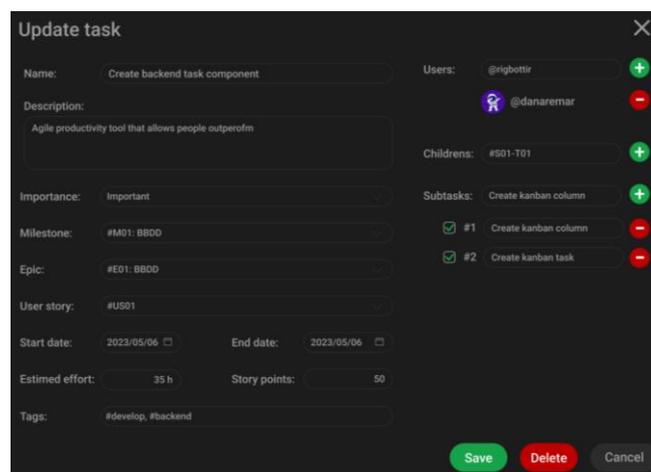


Imagen 26: Mockup actualizar tarea

- **Esfuerzos**

Esta ventana la usará el usuario para ver todos los esfuerzos que ha imputado en su proyecto. Tendrá una parte superior para imputar un esfuerzo, y la parte inferior para visualizar los que ha realizado.

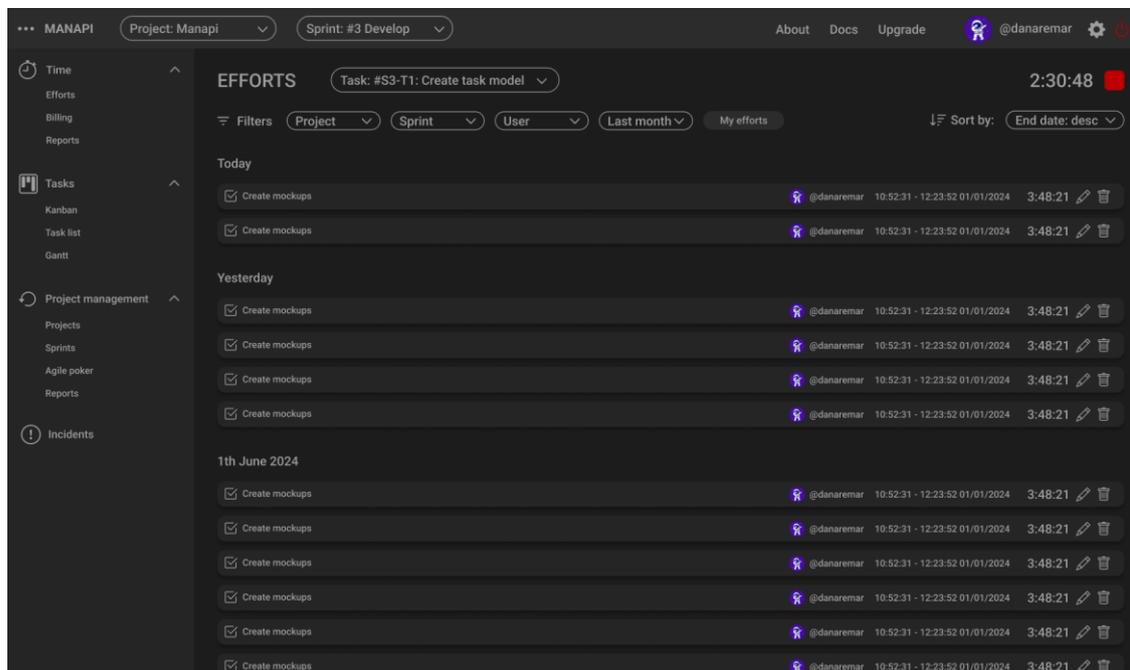


Imagen 27: Mockup listado de esfuerzos

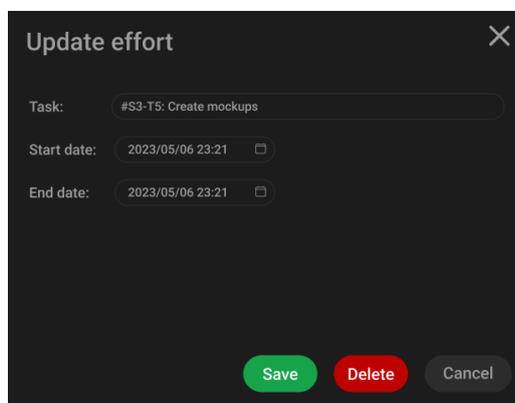


Imagen 28: Mockup actualización de esfuerzo

11. Arquitectura del sistema

En este apartado, se explican la relación entre los distintos componentes del sistema desarrollado.

Antes de nada, se va a mostrar un diagrama que muestre las tecnologías empleadas, así como su organización.

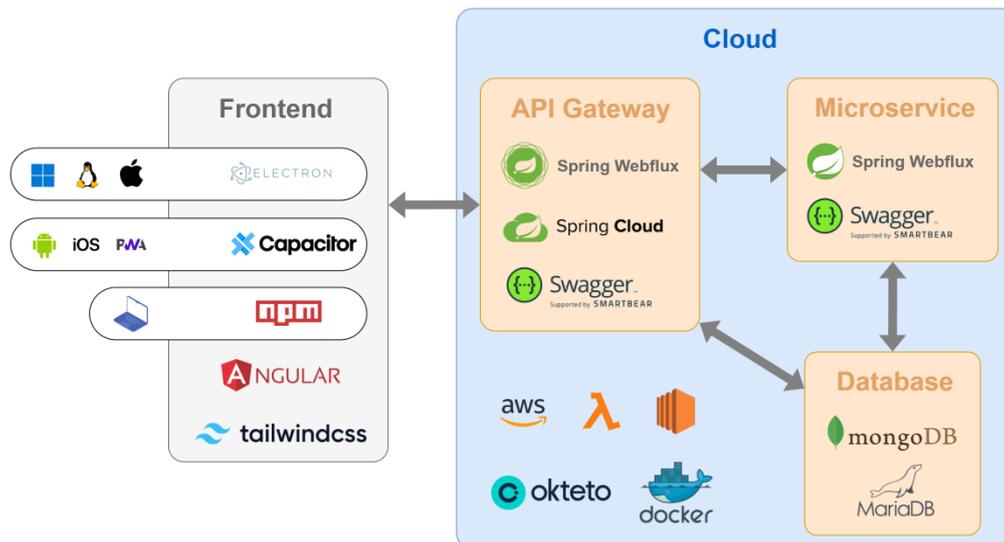


Imagen 29: Arquitectura y tecnologías para la construcción

11.1. API Gateway

Tal como se describe en la sección 5.2, un *API Gateway* puede usarse para la simplificación de la gestión de microservicios, control de acceso y seguridad, escalabilidad y monitorización. En este caso, he implementado un API Gateway con el objetivo principal de homogeneizar la seguridad, simplificar el proceso de desarrollo y segmentar el sistema por módulos o partes, lo que lo convierte en una herramienta muy útil para el equipo de desarrollo.

Al utilizar este componente, se pueden establecer **políticas de seguridad** comunes que aplican a todas las partes del sistema, reduciendo riesgos de seguridad y una mejora de integridad de los datos. Estas políticas pueden indicar limitaciones de **rates** y **cuotas**, **feature toggles** o **permisos** en función de los recursos demandados.

Por otro lado, la aplicación está orientada en un **desarrollo modular**, en el que existen múltiples módulos o partes enfocadas en la gestión integral de una organización. Es por ello que, para simplificar el desarrollo, el equipo de desarrolladores asociados a un módulo concreto podrá abstraerse de todos los demás, así como de la autenticación, autorización, planes de precio, **rates**, cuotas y feature toggles.

Con respecto la tecnología usada, se ha decidido optar por implementar un *API Gateway* basado en **Spring WebFlux** con la librería de **Spring Cloud Gateway** por las siguientes razones:

- **Buen rendimiento:** permite un gran número de peticiones por segundo, poca latencia (del orden de 32ms [19]) y peticiones no bloqueantes o reactivas (tal y como se indica en la sección 5.3).

- **Independiente del proveedor:** permite no estar asociado a un proveedor (*AWS, GCP, Azure...*), lo que permite reducir riesgos a largo plazo.
- **Spring Cloud Functions:** puede convertirse a funciones lambda de los distintos proveedores (*AWS, GCP y Azure*). Estas funciones permiten un ahorro considerable por cada petición, ya que su precio ronda los 20 céntimos por un millón de peticiones. [20]
- **Escalabilidad:** es altamente escalable, tanto vertical como horizontalmente.
- **Uso de bases de datos:** a diferencia de otras opciones como los servidores Nginx, puede conectarse a bases de datos de diferentes tipos. Esta base de datos puede usarse para almacenar los usuarios y la información asociada a los planes de precio entre otras.
- **Flexibilidad de enrutamiento:** el enrutador puede configurarse en la propia configuración de *Spring* o mediante "filtros".
- **Compatibilidad de protocolos:** puede usar *HTTP, WebSocket, gRPC, TCP, UDP...* [21]

Por otra parte, el desarrollo de un *API Gateway* a nivel de código en comparación con otras soluciones como *AWS Gateway*, supone un mayor desempeño de tiempo y de complejidad. Dado que las ventajas de su uso superan a los inconvenientes, se ha decidido por esta opción.

11.2. Microservicios

En un planteamiento inicial y con perspectiva de futuro, la aplicación se compone de los siguientes módulos:

- **Project Management:** permite la gestión integral de un proyecto (principalmente software). Entre otras se encuentra la gestión de sprints, *agile poker*, tareas, esfuerzos dedicados por tareas, tableros kanban, historias de usuarios, iniciativas, *milestones* y épicas.
- **HR Management:** permite la gestión integral de un departamento de *Recursos Humanos*. Entre otras se encuentra la gestión de contratos, entrada-salida de la oficina, organización del tiempo (ej. un trabajador realiza 8 horas de lunes a jueves y 7 horas los viernes), vacaciones, calendario y reclutamiento.
- **CMDB:** permite la gestión de la *CMDB* (o base de datos de la configuración) descrita en *ITIL*. Se compone de configuraciones como hardware, software, sistemas, instalaciones e incluso personal. Controla entre otras cosas el ciclo de vida, responsabilidades, ubicaciones, responsabilidades asociadas a cada una de las configuraciones. [22]
- **Incident:** permite la gestión de incidencias asociadas a una serie de activos y de usuarios.
- **Security Management:** permite la gestión de vulnerabilidades (vulnerabilidades por activos) y de riesgo de una organización a nivel de ciberseguridad.

Cada módulo representa un dominio diferente, por lo que podría separarse en diferentes aplicaciones aisladas entre sí, tal como se establece en *DDD (Domain-Driven Design)* en la sección 5.1. Teniendo en

cuenta la sección anterior 11.1 sobre *API Gateway* y los diferentes dominios de aplicación, se ha creado un siguiente diagrama ilustrativo:

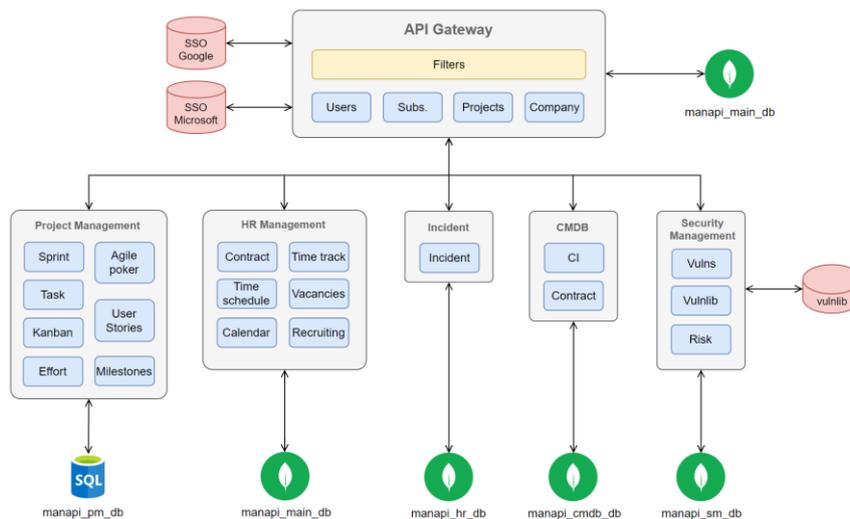


Imagen 30: Diagrama de funcionalidad de microservicios

Como puede apreciarse en el diagrama, *API Gateway* está conectado a los diferentes microservicios y a su vez estos, aunque funcionen de forma aislada, pueden relacionarse entre sí si fuese necesario. Cada microservicio tendrá una base de datos diferente, tal y como se explica en la sección 11.3.

11.3. Bases de datos

Previamente, la aplicación tenía un enfoque monolítico basado en una base de datos de tipo relacional *MariaDB*. Dado que ahora se dispone de una arquitectura basada en microservicios, se pretende usar el tipo de base de datos más adecuada a cada uno de ellos. Para ello, se determinará el tipo de base de datos en función del siguiente contexto:

- **Servicio de proyectos:** usará una base de datos relacional (*SQL*) llamada *MariaDB*. Se ha decidido su uso debido a que existen una gran relación entre las diferentes tablas de la base de datos, al igual que se encontraba en la aplicación previa al *TFM*.
- **Otros servicios:** usarán una base de datos *NoSQL* de tipo documental llamada *MongoDB*. Se prefiere por su menor coste de mantenimiento (véase en sección 13.11), gran escalabilidad y disponibilidad, así como dispone de latencias menores de acceso (comparado con las *SQL*).

En el futuro, conforme vayan surgiendo nuevas necesidades, podrá usarse alguno de los tipos presentes de base de datos (por estandarizar) o podrá usarse una nueva más apropiada al ecosistema a tratar.

12. Análisis de capacidad

En este apartado, se va a realizar un análisis de cuánto de escalable es la aplicación.

En primer lugar, se va a definir análisis de capacidad de un *MSA* (*MicroServices Architecture*) como la máxima carga que es capaz de soportar. [23]

Dado que la aplicación se desea que sea altamente escalable, es necesario buscar tecnologías que cubran cada uno de los diferentes microservicios descritos previamente. Intencionadamente, se buscan tecnologías *serverless*, que permitan adaptarse automáticamente a la demanda y una capacidad superior al resto de soluciones a igualdad de costes. Las diferentes soluciones que se plantean son:

- **AWS Lambda**: para la ejecución *serverless* de las diferentes aplicaciones. El proveedor no indica límites de capacidad, por lo que **no existe un factor limitante en la capacidad**.
- **MongoDB Atlas (plan serverless)**: como base de datos NoSQL documental MongoDB de forma *serverless*. El proveedor no indica límites de capacidad, por lo que **no existe un factor limitante en la capacidad**.
- **Amazon Aurora**: como base de datos SQL MySQL *serverless* de forma auto escalable. El proveedor indica limitación mediante ACU (unidades de capacidad de Aurora, cada una establece 2GiB y 1 sólo *vCPU*), en el que no indica número máximo de instancias y consumo de ACU, pero si un límite que duplica su capacidad entre 5 y 50 segundos. Es por ello, que **no existe un factor limitante en la capacidad máxima, pero si en la capacidad de aumento**.

Estas disponen de un coste prácticamente lineal, ya que son servicios escalables horizontalmente y los precios están relacionados con el número de peticiones que se les realice.

A modo de simplificación, se excluyen los costes fijos, ya que estos son relativos al tiempo que se encuentran activos los servicios y su coste en conjunto no llega a superar ni los 100€ al mes, incluyendo aspectos como *backups*, *gigabytes* transferidos y servicios adicionales como los de seguridad y *AWS Gateway*...

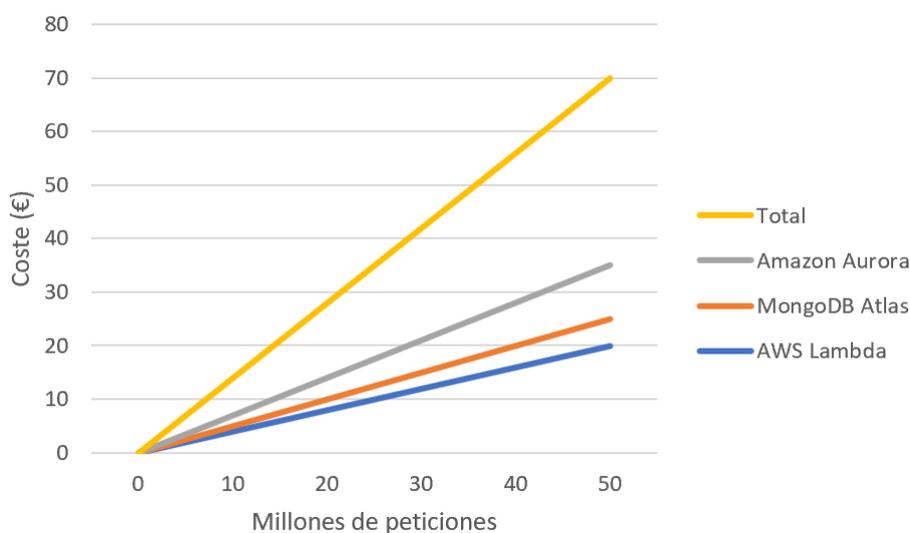


Imagen 31: Evolución del coste en función de las peticiones del cliente

13. Implementación del sistema

13.1. Entorno de desarrollo

En esta sección, se va a describir el entorno de desarrollo que se ha utilizado, así como las versiones y plugins empleados en cada caso.

- **Github**

Se usa para almacenar el código fuente del proyecto y su integración con las herramientas de despliegue. Se puede encontrar el repositorio [en este enlace](#).

Dado que el proyecto solamente está siendo desarrollado por un integrante, la gestión de ramas del repositorio será por cada mes (que incluye 2 sprints) con el siguiente patrón “mes-año”. Por ejemplo, la rama asociada a al mes de febrero de 2023 será “feb-2023”. Cuando el mes finalice, se realizará una *pull request* a “develop” y se creará una nueva rama asociada al nuevo mes a partir de “develop”. La rama “main” representará la versión en producción que se encuentra desplegada.

Dentro del repositorio se pueden encontrar estos elementos:

- **Backend:** incluye el código que se realizó previamente sobre *Spring Mvc*, sobre el que se basó posteriormente el desarrollado bajo *Spring WebFlux*.
- **Frontend:** incluye el código asociado a la interfaz de usuario de la aplicación.
- **Webflux:** incluye el código del *API Gateway* y los microservicios que se ha realizado de forma definitiva.

Todos los ficheros tienen en su interior un archivo “.gitignore” que impide que se suba código no necesario para la ejecución del propio repositorio, como es el caso de archivos compilados como “node_modules” (en *NodeJS*) y “.class” en *Java*.

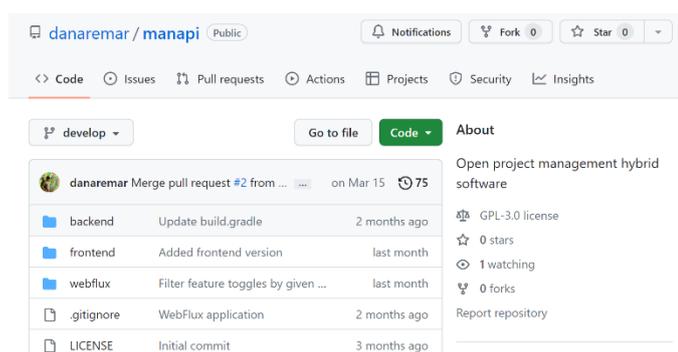


Imagen 32: Repositorio en Github

Por otra parte, para el control de versiones en local, se usa la aplicación de **Github for Windows**, en la que se realizarán todos los *clone*, *pull*, *push*, *stash*, creación de ramas y visualización de cambios.

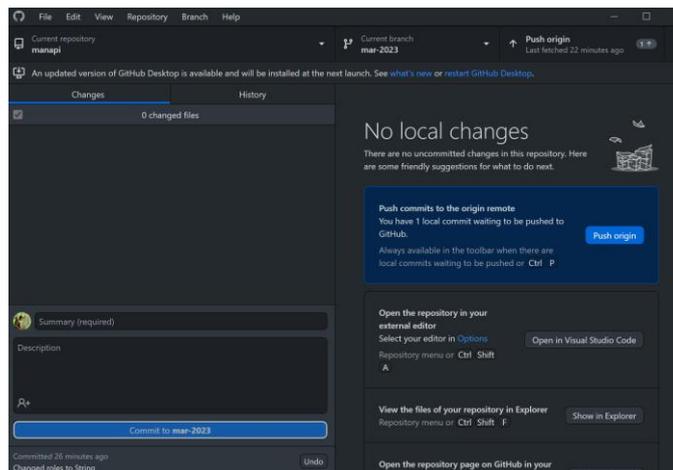


Imagen 33: Github for Windows

- **OneDrive / Teams**

Dada la experiencia previa con *Microsoft Teams*, uso por los tutores del *TFM* y su integración con otras herramientas de *Microsoft*, se ha optado como medio de comunicación por excelencia.

Para ello, se ha creado un canal llamado "*TFM Daniel Arellano*" que incluye todo lo relativo al trabajo, desde planificación de reuniones, archivos y publicaciones (equivalente a un chat).

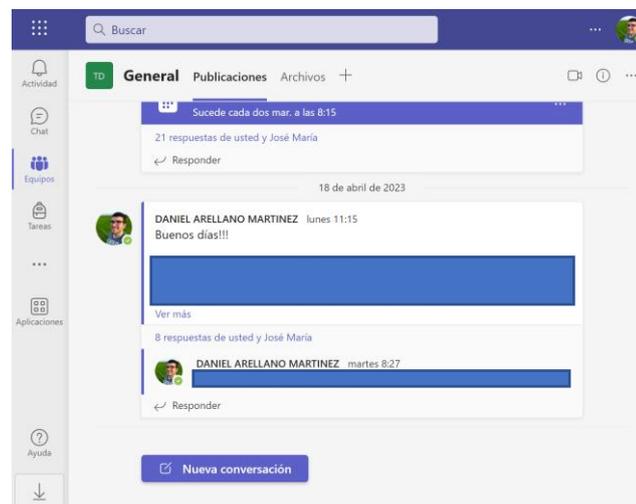


Imagen 34: Canal TFM en Microsoft Teams

Además, se planificarán las reuniones de seguimiento en la sección de "*Calendario*" y podrá verse las distintas grabaciones en el propio canal de *Teams* o mediante *Outlook*.

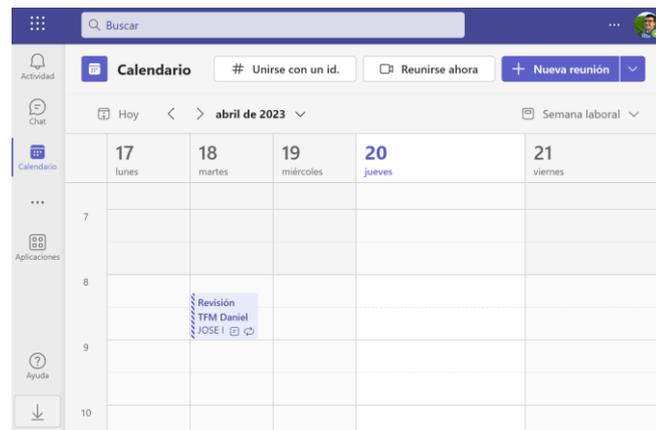


Imagen 35: Reunión de seguimiento en Teams

• OneDrive

Dada la experiencia previa de *OneDrive*, su integración con *Microsoft Teams* y las ventajas de control de versiones sobre archivos propios de *Microsoft*, se ha optado esta como repositorio principal de documentación del proyecto.

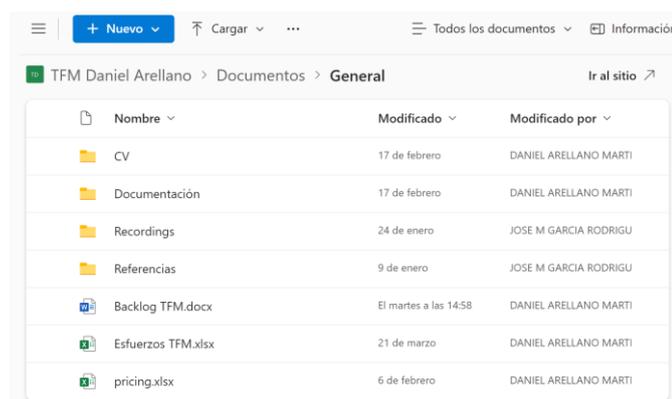


Imagen 36: Carpeta TFM en OneDrive

Los documentos se dividen siguiendo la siguiente estructura:

- **Documentación:** carpeta que incluye todos los documentos relacionados con la realización de la memoria.
- **Recordings:** grabaciones realizadas del *Teams*.
- **Referencias:** documentos recomendados por los tutores.
- **Backlog TFM:** indica las tareas a realizar en base a las reuniones de seguimiento del proyecto.
- **Esfuerzos TFM:** se indicará las horas realizadas con respecto las horas planificadas.
- **Pricing:** indica un análisis de precios en función del plan y de las peticiones.

- **Mongo Atlas**

Dada la experiencia previa, su facilidad de usar, enfoque multi-proveedor (*AWS*, *GCP* y *Azure*) y su coste gratuito (con límites bastante aceptables para un estado inicial de la aplicación) se ha decidido usar Mongo Atlas como la solución de despliegue de bases de datos NoSQL basadas en documentos.

Se definen 3 entornos: *manapi-dev* (para desarrollo), *manapi-pre* (para pre-producción) y *manapi-pro* (para producción). Cada uno de ellos estará basado en el plan gratuito, que tendrá 512MB de almacenamiento, sin copias de seguridad, *vCPUs* (*Virtual CPUs*) y memorias compartidas.

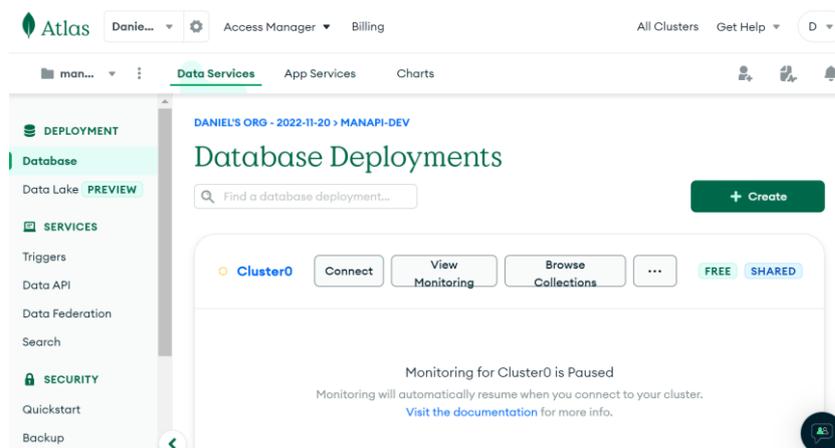


Imagen 37: Mongo Atlas

- **MariaDB**

Base de datos relacional que ha sido desarrollada por los creadores de *MySQL* y de código abierto (licencia *GPLv2*).

Se ha decidido el uso de este *SGDBR* (Sistema de Gestión de Bases de Datos Relacionales) para la base de datos usada en el proyecto por su fácil integración con Java (compatible con casi todos los plugins de *MySQL*) y por ser totalmente gratuito a todos sus niveles. Se ha usado la versión 10.11.2 de *MariaDB*.

- **DBeaver**

Herramienta para bases de datos cuya versión gratuita o “Community” permite diferentes herramientas para varios *SGBDR*, incluyendo *MariaDB* y *MySQL*.

Se ha elegido esta herramienta porque permite generación de sentencias SQL desde la propia tabla de datos (INSERTs por ejemplo para añadirlo después al fichero SQL de población de datos del servidor), mostrar el diagrama que interrelaciona las tablas (Diagrama ER), las propiedades de todos los atributos de una tabla y los datos representados en forma de tabla (que además te permite insertar, editar y eliminar directamente desde esta rejilla).

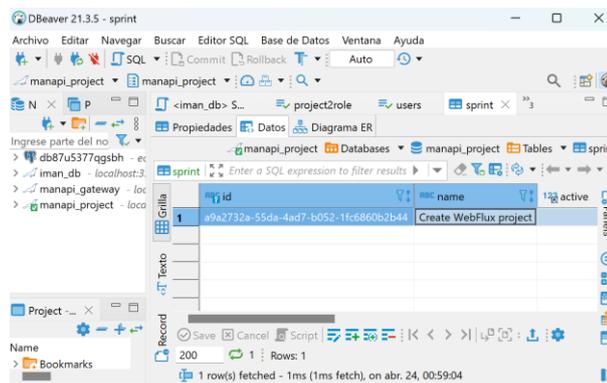


Imagen 38: DBeaver

• Visual Studio Code

Editor de código fuente desarrollado por Microsoft con licencia *MIT* (código abierto, libre y gratuito). Integra entre otras cosas: Intellisense (permite completar el código), depuración avanzada, comandos / panel de Git y gran cantidad de plugins.

Se ha decidido su uso por su experiencia previa y por que permite editar todo el código fuente de manera centralizada desde este mismo (*API Gateway*, microservicios, *frontend* y conexión con *MongoDB*). La versión usada es 1.77.3.

Se han instalado los siguientes plugins:

- **Angular Language Service:** permite el soporte para Angular.
- **Colorize:** para visualizar los colores de CSS y poder cambiarlos de forma simple.
- **Debugger for Java:** permite depurar código Java.
- **Extension Pack for Java:** permite ejecutar código Java.
- **Gradle for Java:** permite usar las funcionalidades de Gradle en el editor.
- **IntelliCode:** permite mediante IA ayudar al desarrollo de las aplicaciones.
- **Language Support for Java™ by RedHat:** permite realizar funciones de corrección, refactorización, Gradle e Intellisense.
- **MongoDB for VS code:** permite la conexión con una base de datos de *MongoDB*, permitiendo ver los diferentes documentos, crearlos, editarlos o eliminarlos.
- **Snyk:** permite identificar posibles vulnerabilidades, así como las versiones que no están actualizadas.
- **SonarLint:** para detectar y solucionar los problemas de calidad.
- **Spring Boot Dashboard:** permite tener un panel de control de Spring Boot, que entre otras cosas permite inicializar las aplicaciones.
- **Spring Boot Tools:** permite validar y asistencia para Spring Boot.
- **YAML:** permite el soporte para archivos con formato YAML.

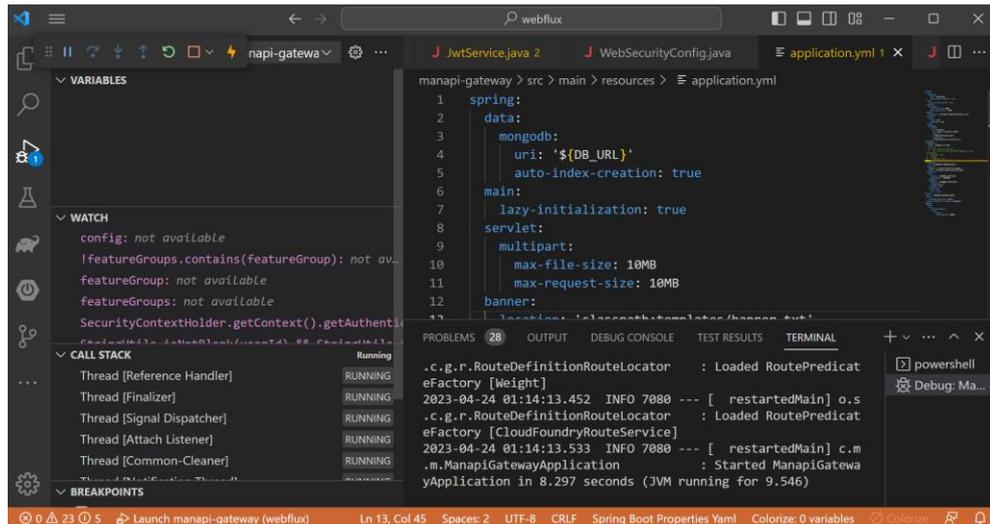


Imagen 39: Visual Studio Code

- **NodeJS**

Entorno de ejecución de *JavaScript* de código abierto, usado para la ejecución de la interfaz de usuario y su compilación en versión de escritorio. Se ha decidido su uso porque es el intérprete que se requiere para *Angular*.

13.2. Fichero de configuración de planes

Tras realizar un análisis de UML en la sección 10.1, se ha determinado que las propiedades de planes de precio, precios, *rates*, cuotas y funcionalidades estén determinadas de forma estática en un fichero. Cada usuario se relacionará con un plan de precios, así como una serie de funcionalidades.

Para la creación de un fichero de configuración me he inspirado en la investigación de Rafael Fresno y la he adaptado a este contexto [7].

```
"plans": [
  {
    "name": "Basic",
    "cost": 5.0,
    "rate": 15,
    "rateunit": 1,
    "quota": 1000,
    "quotaunit": 86400,
    "ovg": 0.01
  },
  {
    "name": "Premium",
    "cost": 8.0,
    "rate": 25,
    "rateunit": 1,
    "quota": 10000,
    "quotaunit": 86400
  }
]
```

Código 1: JSON (fichero de configuración) de la herramienta de SmartLAMA Rafael Fresno [7]

En primer lugar, el formato JSON supone un formato con ciertas carencias frente a YAML para archivos de configuración por los dos siguientes aspectos:

- **Legibilidad:** YAML es más legible que JSON debido a su indentación y su puntualización mínima.
- **Comentarios:** YAML permite comentarios en el archivo, lo que lo hace más interesante para ser usado en archivos de configuración, ya que los desarrolladores pueden entender y modificar este archivo.

Por ello, el fichero de configuración de planes será de formato YAML e integrará los mismos campos que indica Rafael Fresno en su propuesta. El fichero planteado tiene la siguiente forma:

<pre>plans: free: cost: 0.0 rate: 5 rateunit: 1 quota: 100 quotaunit: 60 domain-limits: projects: 2 featuregroups: - project ...</pre>	<pre>featuregroups: project: - effort - gantt - kanban - sprint incident: - incident cmdb: - actives ...</pre>
--	--

Código 2: *plans.yaml*

Como puede apreciarse, se ha conservado la parte de los planes y se ha integrado la gestión de las funcionalidades de la aplicación, siendo agrupadas estas por grupos de funcionalidades. A continuación, se describirán los campos establecidos en esta:

- **plans:** incluye el conjunto de planes de precio de la aplicación. Bajo esta, cada nombre será un tipo de plan diferente. El plan por defecto (si el usuario no tiene asociado ninguno o se encuentra su plan caducado) será el plan “*free*”.
- **cost:** representa el precio mensual que tiene el plan asociado por cada usuario.
- **rate:** indica el número de peticiones máximas asociadas al *rate* deseado.
- **rateunit:** indica el número de segundos para que el *rate* tenga efecto. Normalmente esta unidad será 1, para limitar el número de peticiones por segundo. No es capaz de soportar cifras decimales (por la librería que implementa las limitaciones de peticiones, véase en 13.9)
- **quota:** indica el número máximo de peticiones máximas asociadas a la cuota.
- **quotaunit:** indica el número de segundos para que la cuota tenga exceso. Normalmente esta será una unidad superior a 1 segundo.
- **domain-limits:** indica un diccionario de limitaciones del dominio, en el que se establece la propiedad y su valor numérico. En este caso concreto, se muestra que el número máximo de proyectos en los que el usuario puede ser el dueño es de 2.
- **featuregroups:** indica qué grupo de funcionalidades puede acceder el usuario con la suscripción. En la sección inferior se describen las funcionalidades asociadas a cada grupo.

Este fichero estará presente en la implementación del *API Gateway*, para que permita realizar *rate / quota limit, feature toggles* y *pricing*.

13.3. Proyecto padre

Como se indicó previamente, el proyecto está basado en una serie de microservicios gestionados por un *API Gateway*. Para ello, se ha configurado un proyecto padre a todos los microservicios y *API Gateway*, que permitirá gestionar entre otras cosas la gestión de dependencias, construcción del software y pruebas software.

Primero, se indican todas las aplicaciones hijas que existen en el fichero “*settings.gradle*”. Las aplicaciones hijas corresponderán con las indicadas en la sección 11.2.

```
rootProject.name = 'manapi-backend'  
include 'manapi-gateway'  
include 'manapi-project'  
include 'manapi-cmdb'  
include 'manapi-hr'  
include 'manapi-security'
```

Código 3: *settings.gradle* padre

Segundo, se configuran todas las dependencias comunes entre todas las aplicaciones hijas dentro del fichero padre *“build.gradle”*.

```
plugins {
    id 'java'
    id 'org.springframework.boot' version '2.7.10'
    id 'io.spring.dependency-management' version '1.0.13.RELEASE'
}

allprojects {
    ...
    sourceCompatibility = '19'
    ...
    dependencies {
        // SPRING
        annotationProcessor '...:spring-boot-configuration-processor'
        developmentOnly 'org.springframework.boot:spring-boot-devtools'

        // SWAGGER
        implementation 'org.springdoc:springdoc-openapi-ui:1.6.14'

        // OTHERS
        implementation 'org.modelmapper:modelmapper:3.1.1'
        compileOnly 'org.projectlombok:lombok'
        annotationProcessor 'org.projectlombok:lombok'

        // TESTING
        testImplementation '...:spring-boot-starter-test'
        testImplementation 'junit:junit:4.13.2' }
    ...
}
```

Código 4: *build.gradle* padre

Tras estos dos pasos anteriores, tenemos el entorno listo para que todos los proyectos hijos se sincronicen de forma automática con las dependencias indicadas. En cuanto a versión de *Spring Boot*, se ha optado usar la versión 2, ya que la versión 3 es incompatible con la librería de *spring-doc* que permite la generación de *Swagger-ui*.

13.4. Microservicios

Para la creación de los microservicios, se puede crear de diferentes formas como puede ser mediante plugins en el *IDE* o mediante [Spring Initializr](#). En mi caso particular, he decidido optar por esta última opción por su facilidad de uso y que no requiere instalación en el dispositivo. Dentro de este, se podrá añadir todas las librerías necesarias, así como las distintas versiones de *Java*, *Spring Boot* o el gestor de dependencias que se desee usar.

Código 5: Spring Initializr para generar un microservicio

Cada microservicio creado deberá ser añadido a “*settings.gradle*” del proyecto padre, para que pueda sincronizarse. Una vez creado, deberá eliminarse las dependencias duplicadas en el padre para el fichero “*build.gradle*” del microservicio en cuestión y el resto de código que no sea ni dependencias ni “*group*”. En la imagen a continuación se mostrará un ejemplo para el microservicio de “*project*”:

```
group = 'com.manapi.manapiproject'

dependencies {

    // SPRING
    implementation 'org.springframework.boot:spring-boot-starter-jersey'
    implementation 'org.springframework.boot:spring-boot-starter-web'
    implementation 'org.springframework.boot:spring-boot-starter-security'

    // BBDD -> SQL
    implementation 'org.springframework.boot:spring-boot-starter-data-jdbc'
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
    runtimeOnly 'org.mariadb.jdbc:mariadb-java-client'
    runtimeOnly 'org.postgresql:postgresql:42.4.1'
}
```

Código 6: *build.gradle* en *manapi-project*

13.5. Redireccionamiento

Como se indicó en la sección 11.1, se debe de crear un API Gateway usando la librería de Spring Cloud Gateway y su programación reactiva tal y como se explica en el apartado anterior.

```
group = 'com.manapi.manapigateway'

dependencies {

    // SPRING
    implementation 'org.springframework.boot:spring-boot-starter-webflux'
    implementation 'org.springframework.boot:spring-boot-starter-security'
    implementation 'org.springframework.boot:spring-boot-starter-validation'
    implementation 'org.springframework.session:spring-session-core'

    // OTHERS
    implementation 'org.springdoc:springdoc-openapi-webflux-ui:1.6.14'
    implementation 'io.jsonwebtoken:jjwt:0.9.1'
    implementation 'com.github.vladimir-bukhtoyarov:bucket4j-core:7.6.0'

    // CLOUD
    implementation 'org.springframework.cloud:spring-cloud-starter-gateway:3.1.5'

    // DB -> MongoDB (NoSQL document oriented)
    implementation 'org.springframework.boot:spring-boot-starter-data-mongodb'

    // TEST
    testImplementation 'io.projectreactor:reactor-test'
}
```

Código 7: build.gradle en manapi-gateway

Tras la creación de esta, es necesario indicarle la configuración que debe tomar. Originalmente se construyó con el fichero *“application.properties”*, impidiendo una configuración simple en un solo fichero, ya que el redireccionamiento debía de estar localizado en una clase del código. Para solucionar este problema, se usó mejor el fichero de configuración *“application.yml”*, solucionando así dicha problemática. [24]

Dentro de este fichero de configuración, deberá introducirse una lista de rutas delimitadas por el carácter *“-”*. Para cada elemento será necesario la siguiente información:

- **id:** nombre con el que se llamará al servicio.
- **uri:** dirección en la que se encuentra desplegado el servicio.
- **filters:** en caso de que se quiera usar algún filtro preestablecido por Spring Boot o que haya sido creado a nivel de código.
- **predicates:** es el patrón de caracteres que acciona el redireccionamiento a la *URI* indicada previamente.

Es importante mencionar, que la generación de documentación sobre el *API REST* de *Swagger* u *OpenAPI* tomará como referencia la del resto de microservicios, tal y como se establece en el apartado *13.6 OpenAPI*.

```

...
spring:
  security:
    user:
      name: root
      password: root
  cloud:
    gateway:
      routes:
        - id: projects
          uri: http://localhost:8081
          filters:
            - FeatureGroup=project
          predicates:
            - Path=/project/*/projects/**
springdoc:
  swagger-ui:
    path: /swagger-ui.html
...

```

Código 8: application.yml en manapi-gateway

Por otro lado, se debe de implementar un filtro que permita indicar al microservicio hijo 3 parámetros: proyecto, identificador del usuario y el rol que ocupa en el proyecto. Para ello se crea una clase llamada “GlobalPreFiltering” que extienda de “GlobalFilter”, tal y como se indica en la documentación de Spring Boot. [25]

```

@Component
public class GlobalPreFiltering implements GlobalFilter {
    ...
    @Override
    public Mono<Void> filter(ServerWebExchange exchange,
                            GatewayFilterChain chain) {

        String url = exchange.getRequest().getURI().toString();
        String projectId = url.split("/project/")[1].split("/")[0];
        String token = jwtService.getTokenFromRequest(exchange.getRequest());
        String userId = jwtService.getUserIdFromToken(token);
        String role = getRole(projectId, userId);

        ServerHttpRequest req = exchange.getRequest()
            .mutate()
            .header("X-project-id", projectId)
            .header("X-user-id", userId)
            .header("X-user-role", role)
            .build();
        ServerWebExchange mutatedExchange = exchange.mutate()
            .request(req).build();
        return chain.filter(mutatedExchange);
    }
    ...
}

```

Código 9: GlobalPreFiltering en manapi-gateway

Esta información es recibida mediante el microservicio hijo y la añade en el contexto de seguridad de *Spring Boot* (para que pueda ser usada posteriormente por ejemplo para filtrar roles o extraer el usuario).

```

@Component
public class SecurityFilter extends OncePerRequestFilter {

    @Override
    protected void doFilterInternal(HttpServletRequest request,
        HttpServletResponse response, FilterChain filterChain)
        throws ServletException, IOException {

        // get from headers
        String userId = request.getHeader("X-user-id");
        String projectId = request.getHeader("X-project-id");
        String role = request.getHeader("X-user-role");

        // instantiate roles, userId & projectId in SecurityContextHolder
        if(StringUtils.isNotBlank(userId)
            && StringUtils.isNotBlank(projectId)
            && StringUtils.isNotBlank(role)){
            List<GrantedAuthority> roles = List.of(
                new SimpleGrantedAuthority(role));
            HeaderInfo headerInfo = new HeaderInfo(userId, projectId, role);
            UsernamePasswordAuthenticationToken auth =
                new UsernamePasswordAuthenticationToken(headerInfo, null, roles);
            SecurityContextHolder.getContext().setAuthentication(auth);
        }
        // apply filter
        filterChain.doFilter(request, response);
    }
}

```

Código 10: SecurityFilter en manapi-project

A continuación, se mostrará un ejemplo de cómo restringir los permisos a un usuario asociado a un proyecto en función de los roles indicados en la petición.

```

@PreAuthorize("hasAuthority('OWNER')")
@PostMapping(value =("/{projectId}/projects/testOwner")
public ResponseEntity<Object> getSprintOwnerTest(...) {
    return new ResponseEntity<>("OK OWNER", HttpStatus.OK);
}

```

Código 11: Permiso en la petición del microservicio consultado

Para probar este redireccionamiento, se realizará una petición al *API Gateway*, que la redijera a un *endpoint* del microservicio que se desea probar con el rol en el proyecto seleccionado de "owner". Como podemos observar en la imagen inferior, el funcionamiento es el esperado.

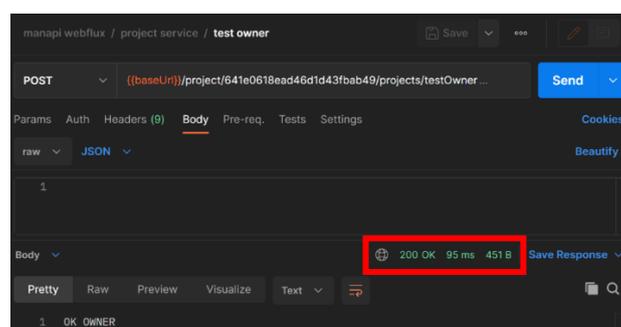


Imagen 40: Petición de prueba de redireccionamiento

13.6. OpenAPI

Para la generación de documentación *REST* de los microservicios se ha decidido usar *OpenAPI*, principalmente por su fácil integración con *Spring Boot* y *Spring WebFlux*.

En el entorno de servicios (*Gateway* y microservicios), se ha configurado *OpenAPI* para generar automáticamente la documentación de cada servicio. Esto se logra mediante la creación de una clase de configuración de *OpenAPI* y la inclusión de anotaciones correspondientes en cada uno de los servicios involucrados.

```
@Configuration
public class OpenApiConfig {

    @Bean
    public OpenAPI addInfo() {
        return new OpenAPI()
            .info(new Info().title("MANAPI Project Service")
                .version("v0.1")
                .license(new License().name("GPLv3")
                    .url("https://danaremar.github.io/iMan/login")));
    }
}
```

Código 12: Configuración de OpenAPI en el microservicio de proyectos

Se pueden generar anotaciones asociadas a cada controlador e incluso a cada petición, aunque por simplificación solamente se han usado en el controlador las siguientes:

- **Tag:** para indicar el nombre del controlador
- **Security Requirement:** para indicar qué tipo de algoritmo de autenticación se precisa. Como se usa JWT, y más concretamente la notación de Bearer, se deberá de indicar "Bearer Authentication". Si carece de esta anotación, indicará que no precisa de ningún método de seguridad para realizar dichas peticiones.

```
@RestController
@SecurityRequirement(name = "Bearer Authentication")
@Tag(name = "Sprints")
@RequestMapping("/project/{projectId}/projects/sprint")
public class SprintController { ... }
```

Código 13: Anotaciones de OpenAPI en el controlador de Sprints

Hasta el momento, cada servicio tiene asociada una documentación de *OpenAPI* accesible en función del tipo de servidor de Spring que esté usando:

- **Microservicios (Spring MVC):** accesible desde ".../swagger-ui/index.html".
- **API Gateway (Spring WebFlux):** accesible desde "../webjars/swagger-ui/index.html".

Originalmente, fue un problema porque *OpenAPI* usa una librería en función del tipo de servidor de Spring que se esté usando. En el caso de *Spring WebFlux*, se carga de forma estática en su inicialización dentro de los *webjars*.

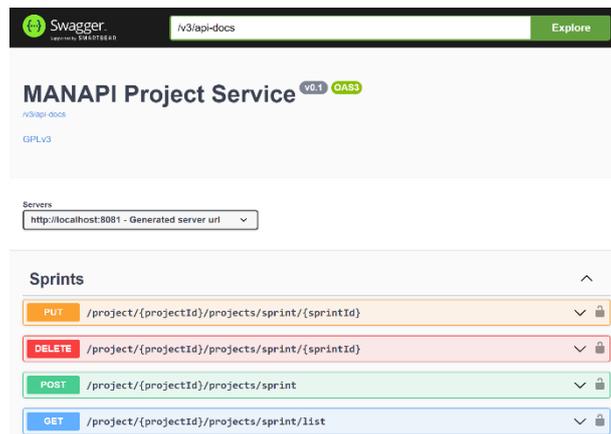


Imagen 41: Documentación REST del microservicio de proyectos

Dada su arquitectura, no es posible llamar correctamente a un microservicio hijo, ya que no tiene implementado el redireccionamiento ni la capa de seguridad asociada (tiene que pasar por el *API Gateway*). Es por ello, que será necesario unificar toda la documentación en una y que resulte más simple y uniforme ante los desarrolladores y/o clientes.

Tras una investigación al respecto, se encontraron librerías que permiten unificar (en inglés “merge”) varias documentaciones de *OpenAPI* como, por ejemplo, “*openapi-merge*” (librería que se encuentra para *NodeJS* y *Typescript*). [26]

En Java no existen librerías al respecto, así que se decidió implementar un nuevo código que permitiese capturar la documentación de los respectivos microservicios. Para ello, se han configurado 3 métodos en la clase de configuración (*OpenAPIConfig*, dentro de *API Gateway*):

- **getOpenAPISpec:** se extrae la especificación de la *API* de *OpenAPI* de la *URL* indicada.

```
private OpenAPI getOpenAPISpec(String specificationUrl) {
    WebClient webClient = WebClient.builder().build();
    String specJson = webClient.get().uri(specificationUrl)
        .retrieve().bodyToMono(String.class).block();
    return new OpenAPIV3Parser().readContents(specJson).getOpenAPI();
}
```

Código 14: Método para extraer especificación de OpenAPI

- **mergeOpenAPISpec:** se combina la especificación de ambas documentaciones.

```
private OpenAPI mergeOpenAPISpecs(OpenAPI target, OpenAPI source) {
    if (target == null) return source;
    if (source == null) return target;
    if (target.getPaths() != null && source.getPaths() != null)
        target.getPaths().putAll(source.getPaths());

    ... // same in components, parameters & responses

    return target;
}
```

Código 15: Método para combinar especificaciones de OpenAPI

- **customOpenAPI:** se recorren cada una de las rutas de los microservicios descritas en “*application.yml*” y se les añade la documentación de *OpenAPI*.

```
@Bean
public OpenAPI customOpenAPI() {
    OpenAPI openAPI = new OpenAPI();

    // microservices
    List<RouteDefinition> routeDefinitions = routeDefinitionLocator
        .getRouteDefinitions().collectList().block();
    for (RouteDefinition r : routeDefinitions) {
        openAPI = mergeOpenAPISpecs(getOpenAPISpec(
            r.getUri().toString() + "/v3/api-docs"), openAPI);
    }

    setInfo(openAPI); // gateway information
    openAPI.setServers(null); // set server to default
    return openAPI;
}
```

Código 16: Método para unificar todas las especificaciones de OpenAPI

Para que esto funcione de forma correcta, será necesario que *API Gateway* arranque posteriormente a todos los microservicios, ya que este se genera de forma estática y debe de realizarse en su inicialización.

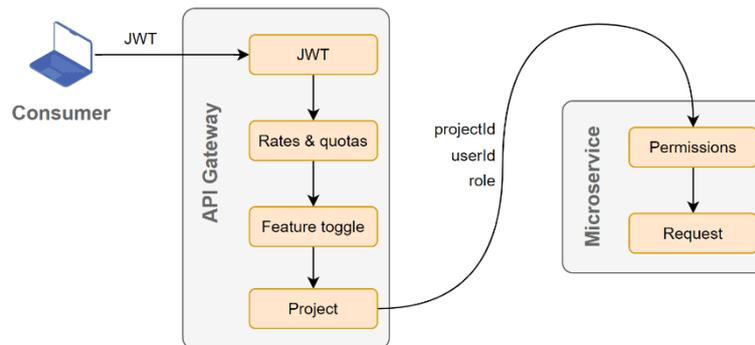
A continuación, se mostrará una representación de la documentación de *OpenAPI* para todos los servicios implicados.



Imagen 42: Documentación de todos los servicios en OpenAPI

13.7. Autorización y autenticación

En este proyecto, se llevan a cabo diferentes técnicas de autorización y de autenticación, para poder dar solución ante identificación, *feature toggles*, *rates*, cuotas, roles y permisos del usuario asociado al proyecto. Para ello se sigue el siguiente modelo con diferentes capas de seguridad que serán explicadas posteriormente:



Código 17: Modelo de seguridad a capas propuesto

En este modelo se distinguen 2 componentes del sistema: *API Gateway* y los microservicios. Se indicará el proceso en el siguiente orden:

1. **JWT:** se comprueba que JWT está correcto.
2. **Rates y cuotas:** se comprueba que no se exceden los límites establecidos en el plan indicado en el token JWT del usuario.
3. **Feature toggles:** se comprueba que el recurso al que accede el usuario está contenido dentro del grupo de funcionalidades del usuario.
4. **Project:** se comprueba que el usuario que accede al recurso de un proyecto tenga asociado un rol a este, en caso contrario indicará un código de error. Para esto, será necesario consultar en la base de datos, ya que cualquier dueño o administrador del proyecto puede cambiar en caliente los roles de este.
5. **Permissions:** se comprueba que la petición a la que acceda tenga un rol asociado igual al usuario que se ha identificado en *JWT*. Esto se realizará mediante las anotaciones propias de *Spring Boot*: `@PreAuthorize`, `@Secured` o `@RolesAllowed`. [27]
6. **Request:** se ejecuta la petición indicada por el usuario.

En las siguientes secciones, se indicará en más profundidad aspectos más detallados con los diferentes procesos.

13.8. JWT

Para garantizar la autenticación y autorización, se ha usado **JWT** (*Json Web Token*), un estándar que permite autenticar y autorizar mediante un token codificado en base64. Este token está firmado (con su clave privada) y solamente el servidor puede cambiar los datos asociados a este. Dispone de los siguientes elementos:

- **Cabecera:** indica el tipo de algoritmo de firma que se usa.

- **Fecha de expiración** (*exp*): fecha en la que dejará de tener validez el token.
- **Marca de tiempo** (*iat*): fecha en la que fue creado el token.
- **Firma**: esta podrá ser validada y cambiada solamente por aquel conocedor de la clave privada con la que fue firmada inicialmente.
- **Claims**: se refiere a otras piezas de información que puedan ser añadidas en el token JWT.

Para su generación, basta con que el usuario acceda a la ruta de identificación, e inserte los datos solicitados. Si esta respuesta es válida, se generará el token con la siguiente información:

- **plan**: indica el plan que tiene el usuario, con el objetivo de establecer las *rates* y cuotas. Dependerá de la suscripción que haya realizado el usuario y se añadirá como “*claims*”.
- **featureGroups**: indica el grupo de funcionalidades que tiene acceso el usuario. Al igual que en el caso anterior, dependerá de la suscripción que haya realizado el usuario y se añadirá como “*claims*”.
- **features**: indica todas las funcionalidades a las que tiene acceso el usuario. Indexará todas las funcionalidades diferentes que existen en los grupos de funcionalidades del usuario y se añadirá como “*claims*”.
- **subject**: se indicará el identificador asociado al usuario, ya que es un campo que permanecerá constante para cada usuario. En un inicio, se realizó con el nombre de usuario, pero este podría ser cambiado y podría suponer una renovación del token (aumenta la complejidad ligeramente).

Tras esto, se va a mostrar a continuación las dos funciones para la generación de los tokens JWT en la aplicación.

```
public String generateToken(User user) {
    Map<String, Object> claims = new HashMap<>();
    claims.put("plan", user.getActualPlan().getType());
    claims.put("featureGroups", user.getActiveFeatureGroups());
    claims.put("features", user.getActiveFeatures());
    return generateToken(claims, user);
}

public String generateToken(Map<String, Object> extraClaims, User user) {
    return Jwts.builder()
        .setClaims(extraClaims)
        .setSubject(user.getId())
        .setIssuedAt(new Date(System.currentTimeMillis()))
        .setExpiration(new Date(System.currentTimeMillis()+1000000*exp))
        .signWith(SignatureAlgorithm.HS512, secret).compact();
}
```

Código 18: Funciones para generar tokens JWT en *JwtService*

Una vez se combina con los servicios de autenticación, se procede a identificarse en la aplicación, comprobar la generación y visualización del token JWT.

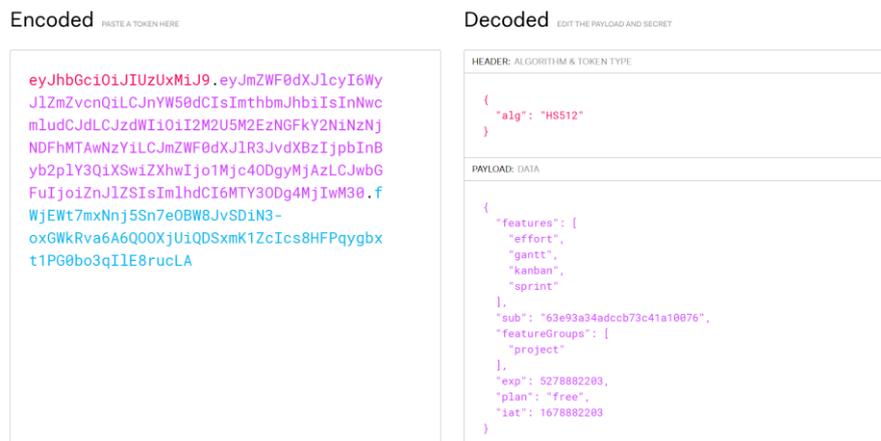


Imagen 43: Token JWT generado en la aplicación

Por otra parte, para comprobar que una petición es legítima mediante el uso de JWT, es necesaria la implementación de un filtro que determine si es correcta o no el token proporcionado. Para ello, se añadirá en la configuración de Spring Boot una extensión de “*AuthenticationManager*” que incorpore en su interior el filtro específico.

```
@Bean
public SecurityWebFilterChain
securityWebFilterChain(JwtAuthenticationManager jwtAuthMan...) {
    AuthenticationWebFilter filter =
        new AuthenticationWebFilter(jwtAuthMan);
    filter.setServerAuthenticationConverter(jwtAuthenticationConverter);
    return http
        .addFilterAt(authenticationWebFilter,
            SecurityWebFiltersOrder.AUTHENTICATION)
}

```

Código 19: Configuración JWT en WebSecurityConfig

```
@Override
public Mono<Authentication> authenticate(Authentication authentication) {
    String token = authentication.getCredentials().toString();
    return Mono.just(jwtService.validateToken(token))
        .filter(x -> x)
        .switchIfEmpty(Mono.empty())
        .map(x -> {
            String userId = jwtService.getUserIdFromToken(token);
            User user = userService.findUserById(userId);
            PrincipalUser principalUser = PrincipalUser.build(user);
            UsernamePasswordAuthenticationToken auth =
                new UsernamePasswordAuthenticationToken(
                    principalUser,
                    null,
                    principalUser.getAuthorities());

            SecurityContextHolder.getContext().setAuthentication(auth);
            return auth;
        });
}

```

Código 20: Configuración JWT en JwtAuthenticationManager

Cuando se acciona la función “validateToken(String token)”, se comprueba si es válido, y en caso contrario devolverá un error que indique un estado HTTP 401 (*unauthorized*) e indica el posible motivo de esta respuesta.

```
public boolean validateToken(String token) {
    try {
        Jwts.parser().setSigningKey(secret).parseClaimsJws(token);
        return true;
    } catch (ExpiredJwtException e) {
        log.debug("validateToken() - Expired jwt bearer token");
        throw new ResponseStatusException(HttpStatus.UNAUTHORIZED,
            ManapiMessages.TOKEN_EXPIRED);
    } catch (Exception e) {
        log.debug("validateToken() - An exception happened: "
            + e.getMessage());
        throw new ResponseStatusException(HttpStatus.UNAUTHORIZED,
            ManapiMessages.TOKEN_BAD_FORMAT);
    }
}
```

Código 21: Función validateToken en JwtService

13.9. Rates y cuotas

Para limitar mediante *rates* y cuotas se usa la declaración del fichero usado en el apartado 13.2, del que se extraerán las diferentes limitaciones.

Para ello será necesario configurar una clase que implemente la interfaz “WebFilter”, llamada “RateFilter”.

Dentro de esta, se implementará un algoritmo de tipo “bucket” o cubo, que es un algoritmo que mediante unos tokens establece limitaciones en el tiempo [28]. Para nuestro contexto, se define que un usuario tiene tantos tokens como peticiones máximas soportadas en el *rate* y en la cuota, en un tiempo indicado en el fichero de configuración para ambas.

```
@Component
public class RateFilter implements WebFilter {

    Map<String, Bucket> bucketCache = new ConcurrentReferenceHashMap<>();

    public Bucket resolveBucket(String username) {
        return bucketCache.computeIfAbsent(username, this::newBucket);
    }

    private Bucket withoutUser() {
        return Bucket.builder().addLimit(Bandwidth.classic(2L,
            Refill.intervally(10, Duration.ofSeconds(1))))
            .build();
    }

    private Bucket newBucket(String userId) { . . . }

    public Mono<Void> filter(. . .) { . . . }
}
```

Código 22: Limitar rates y cuotas en RateFilter

En la implementación se ha usado la librería **Bucket4j**, que es compatible con el framework *Spring Boot* y *Gradle*. En esta se crea una caché correspondiente al cubo, en el que cada petición se verificará si existe, tomando las siguientes casuísticas:

- **No existe:** se crea un nuevo cubo extrayendo los datos del usuario y del fichero de configuración de planes, con el identificador del usuario que accede. Este identificador del usuario se extrae del token JWT.
- **Si existe:** se decreenta el número de tokens en uno para el identificador del usuario asociado al cubo.

Para la creación del “*bucket*” o cubo, se utiliza una función llamada “*newBucket()*”. En esta función, se extrae el usuario que realiza la petición (mediante el token de JWT y la base de datos) y se establece el *rate* y la cuota. Para establecer ambos valores es necesario extraer el plan del usuario y su correspondiente limitación, así como el tiempo máximo definido previamente en el fichero de configuración.

En caso de que no existan usuario, este invocará la función “*withoutUser()*” que establecerá un consumo de 5 peticiones (se generan 10 tokens y se consume 2 por petición) por segundo como máximo.

```
private Bucket newBucket(String userId) {
    User u = userService.findUserById(userId);
    if (StringUtils.isEmpty(userId) || u == null) {
        return withoutUser();
    }
    Plan plan = u.getActualPlan();

    // limit rate
    Bandwidth rate = Bandwidth.simple(
        plan.getRate(),
        Duration.ofSeconds(plan.getRateUnit()));

    // limit quota
    Bandwidth quota = Bandwidth.simple(
        plan.getQuota(),
        Duration.ofSeconds(plan.getQuotaUnit()));

    return Bucket
        .builder()
        .addLimit(rate)
        .addLimit(quota)
        .build();
}
```

Código 23: Nuevo bucket en RateFilter

En caso de que sea necesario comprobar si tiene tokens, se ejecutará la función “*filter*”, permitiendo así consumir 1 token por cada petición de entrada por usuario registrado. Primero, se extrae el usuario de la misma forma que en el caso anterior. Posteriormente, se invoca el método “*tryConsumeAndReturnRemaining(1)*”, que restará 1 token al dato que se encuentra en el cubo. Tras esto, tendrá los siguientes comportamientos:

- **No hay exceso:** indica en la cabecera “*X-Rate-Limit-Remaining*” el número de peticiones disponibles (tantas como tokens).
- **Hay exceso:** indica la cabecera anterior a “0” e indica en la cabecera “*X-Rate-Limit-Retry-After-Seconds*” el tiempo para la renovación de las peticiones. Además, declara un error que indica

que se ha excedido la cuota, dando como resultado una respuesta HTTP 429 (*Too Many Requests*).

Para establecer estas cabeceras, se han tenido en cuenta las buenas prácticas que se establecen en la industria [29] [30].

```
@Override
public Mono < Void > filter(
    ServerWebExchange serverWebExchange,
    WebFilterChain webFilterChain) {

    List <String> lsHeadersAuth = serverWebExchange
        .getRequest().getHeaders().get("Authorization");

    // empty if not contains bearer
    if (lsHeadersAuth != null
        && !lsHeadersAuth.isEmpty()
        && lsHeadersAuth.get(0).contains("Bearer")) {

        // get username
        String token = lsHeadersAuth.get(0).replace("Bearer ", "");
        String userId = jwtService.getUserIdFromToken(token);

        // consume 1 token
        Bucket bucket = resolveBucket(userId);
        ConsumptionProbe probe = bucket.tryConsumeAndReturnRemaining(1);

        // check if can be consumed
        if (probe.isConsumed()) {
            serverWebExchange.getResponse().getHeaders()
                .add("X-Rate-Limit-Remaining",
                    String.valueOf(probe.getRemainingTokens()));
        } else {
            long secToRefill = probe.getNanosToWaitForRefill()
                / 1_000_000_000;
            serverWebExchange.getResponse().getHeaders()
                .add("X-Rate-Limit-Remaining", "0");
            serverWebExchange.getResponse().getHeaders()
                .add("X-Rate-Limit-Retry-After-Seconds",
                    String.valueOf(secToRefill));
            return Mono.error(new QuotaExceededException());
        }
    }
    return webFilterChain.filter(serverWebExchange);
}
```

Código 24: Filtrar en función del bucket en RateFilter

13.10. Feature toggles

En mi proyecto anterior, se evidenció una falta de enfoque comercial, lo cual generó la necesidad de realizar ajustes significativos. Con el objetivo de corregir esta deficiencia, se ha establecido diferentes funcionalidades y características que se ofrecen de manera segmentada, en función de los distintos planes disponibles para nuestros clientes. Por este motivo, se ha decidido emplear feature toggles en este proyecto.

Con el propósito de establecer limitaciones en las funcionalidades disponibles para los usuarios, se ha implementado un enfoque mediante el uso de un fichero de configuración. Este fichero es utilizado para especificar las funcionalidades a las cuales los usuarios tienen acceso, en función del plan que han adquirido. Esta metodología permite una gestión más precisa y controlada de las características y servicios disponibles para cada usuario, permitiendo cumplir así los requisitos comerciales y políticas establecidas previamente en el proyecto.

Además, es necesario indicar qué grupos de funcionalidades permiten el acceso a cada microservicio concreto, por lo que en la configuración de *Spring Cloud Gateway* será necesario añadirle un nuevo filtro personalizado (se explicará a continuación) que indique el grupo de funcionalidad que permite el acceso a este. En caso de no disponer de ese grupo de funcionalidad, *API Gateway* impediría su acceso.

```
spring:
  cloud:
    gateway:
      routes:
        - id: projects
          uri: http://localhost:8081
          filters:
            - FeatureGroup=project
          predicates:
            - Path=/project/*/projects/**
```

Código 25: Configuración de rutas en *application.yml* en *API Gateway*

Se ha implementado un filtro con el mismo nombre que se establece en el fichero de configuración "*application.yaml*", llamándolo "*FeatureGroupGatewayFilterFactory.java*". Este extenderá de la clase "*AbstractGatewayFilterFactory*" de *Spring Cloud Gateway*, permitiendo así realizar una serie de comprobaciones para la redirección al microservicio.

En esta, existirá el método "*apply()*" que validará si el usuario tiene el grupo de funcionalidad mencionado en el archivo de configuración. Para ello, se extraerá el token *JWT* y su usuario, tras esto se extraerán todos los grupos de funcionalidad. En caso de que no disponga el usuario el grupo de funcionalidad del microservicio al que quiere acceder, este rechazará su acceso e indicará que no está autorizado a realizar esa petición.

```
@Override
public GatewayFilter apply(Config config) {
    return (exchange, chain) -> {
        String token = jwtService.getTokenFromRequest(
            exchange.getRequest());
        String featureGroup = config.getFeatureGroup();
        List<String> featureGroups =
            jwtService.getFeatureGroupsFromToken(token);
        if(!featureGroups.contains(featureGroup)) {
            throw new ResponseStatusException(
                HttpStatus.UNAUTHORIZED,
                ManapiMessages.NOT_AUTHORIZED_FEATURE);
        }
        return chain.filter(exchange);
    };
}
```

Código 26: Función *apply()* en *FeatureGroupGatewayFilterFactory*

13.11. Pricing

En este apartado, se va a analizar el *OpEX (Operating Expenses)* o coste de operación, así como establecer los planes de precio del producto.

En el contexto previo de mi proyecto, no se consideraron adecuadamente los gastos de operación, lo cual representa un factor diferenciador crucial al analizar los beneficios esperados de la aplicación a largo plazo. Es por ello, que se quiere analizar estas cuestiones para mejorar significativamente la rentabilidad y sostenibilidad del proyecto.

La implementación de este enfoque consiste en analizar exhaustivamente la viabilidad económica del producto desde la perspectiva de operación en la nube, elaborando planes de precio y analizando si estos son congruentes con los costos asociados y gastos inherentes al funcionamiento del producto.

- **Planes de precio**

En este apartado, se definirá las características que tendrá cada uno de los planes, de forma que sea competitivo y económicamente viable.

Primero, se ha decidido usar un modelo de *pricing* basado en suscripciones (véase en el apartado 0), ya que resulta beneficioso por los siguientes aspectos:

- **Ingresos recurrentes/periódicos:** ofrece ingresos recurrentes y predecibles a lo largo del tiempo. Esto significa que, si se cuenta con 300 usuarios suscritos a planes de pago inicialmente, es probable que continúen pagando de forma periódica en el futuro.
- **Escalabilidad económica:** en función del crecimiento de los clientes aumenta el beneficio. Además, pueden ser adquiridos tanto por personas independientes como por empresas, cosa que no suele ser común en la compra de licencias (más frecuente en entornos empresariales).
- **Fidelización de clientes:** se establece una relación a largo plazo que permite una mejor retención y lealtad.

Por otro lado, se ha decidido ofrecer un plan gratuito para poder así atraer a usuarios potenciales, generar interés en el producto o servicio, y establecer una base de usuarios que podría convertirse en clientes de pago en el futuro.

Para ello, se ha basado en la creación de 3 planes de precio con propósitos diferentes:

- **Free:** permite captar público y generar interés por tener carácter de acceso gratuito.
- **Premium:** ofrece características, funcionalidades adicionales y soporte para usuarios dispuestos a pagar por ello.
- **Enterprise:** se dirige a clientes empresariales, permitiendo la integración de servicios de SSO y un mejor soporte.

A continuación, se han indicado las siguientes características para los distintos planes de precio:

Características	Free	Premium	Enterprise
Pago anual	GRATIS	6 \$ / mes	18 \$ / mes
Pago mensual	GRATIS	8 \$ / mes	24 \$ / mes
Periodo de prueba	N/A	30 días	30 días
Peticiones API / seg. (rate)	5	10	100
Peticiones API / min. (cuota)	100	200	2000
Nº proyectos máx.	2	20	200
SSO	NO	NO	SI
SLA / Soporte	NO	SI	SI
Tiempo respuesta	N/A	3 días	1 día
Disponibilidad (tasa de acierto)	N/A	95%	98%
Características	Proyectos	Proyectos Seguridad CMDB RRHH	Proyectos Seguridad CMDB RRHH

Tabla 2: Planes de precio

Para la creación de estos planes, se han tenido en cuenta las siguientes consideraciones:

- **Plan anual:** ofrecer un descuento si se pagan múltiples meses, con el objetivo de incentivar a la permanencia del usuario.
- **Precios base:** se han estimado en base a otras aplicaciones como FactorialHR (4€/mes plan Operation Hub), Toggl Track (9€/mes plan Starter), Toggl Plan (8€/mes plan Team), Microsoft Project (8,40€/mes plan 1) ... Se pretende competir ofreciendo estos servicios por un precio más bajo a cada uno de ellos de forma separada, para alcanzar el máximo público posible y que sea económicamente viable.
- **Periodo de prueba:** se establece un periodo de prueba que permita captar a futuros clientes. En otros productos propios, suele establecerse en 15 días o en 1 mes. En el caso concreto de la aplicación a desarrollar, se ha optado por el periodo de prueba de 1 mes, ya que de esta forma permitiría a un equipo la prueba del producto durante al menos el periodo en el que se incluye un *sprint* (entre 2-3 semanas normalmente). Por ejemplo, la suite de Microsoft 365 dispone de un plan gratuito durante 1 mes y Toggl Plan tiene 14 días de prueba. [31] [32]
- **Límites por segundo (rate):** limita el número máximo de peticiones en un segundo, evitando una saturación del servidor por parte de un usuario concreto. La competencia no refleja este dato en sus planes de precio, por lo que no es posible realizar una comparación.
- **Límites por minuto (cuota):** limita el número máximo de peticiones en un minuto, permitiendo que no aumenten los costes con el paso del tiempo. La competencia no

refleja este dato en sus planes de precio, por lo que no es posible realizar una comparación.

- **Límites de proyectos:** permiten diferenciar los diferentes planes limitando el máximo número de proyectos que puede ser dueño un usuario (tener el rol "Owner"). En el plan "Free", se indican 2 proyectos como máximo, para que el usuario realice un pago a otros planes de pago si desea disponer de más proyectos. En el plan "Enterprise", se indican 200 proyectos como factor limitante, estableciendo un límite que prácticamente ningún usuario va a alcanzar y sea la mejor opción para usuarios que dirijan más de 20 proyectos. Esta característica no se ha llegado a implementar, porque se ha querido focalizar en otros aspectos del proyecto.
 - **SSO:** permite la integración de *Single Sign-On* propio que tenga configurado la empresa. Este método permite acceder de forma segura y sencilla a todas las aplicaciones del entorno de trabajo, mediante un único proceso de autenticación. Esto permite decantar a las organizaciones a obtener el plan "Enterprise".
 - **Disponibilidad (tasa de acierto):** permite señalar el número de peticiones correctas que se pueden realizar en un mes. Dado que el despliegue en *AWS Lambda* u otros proveedores similares (*Google* y *Azure*) garantiza una disponibilidad de un 99%, y en mi opinión considero que aumentar en 1% este margen permite reducir problemas y cumplir con los objetivos pactados. En el caso del plan "Premium", este índice será más bajo con un 95%, dando prioridad así al plan "Enterprise".
 - **En caso de incumplir el SLA:** en los planes de pago no se cobrará el siguiente mes, con el objetivo de indicar seriedad al contratante y un compromiso por parte del proveedor del producto, sin suponer coste ninguno al proveedor y, por tanto, reduciendo costes. Además, en caso de tener el plan "Enterprise", se recibirá una compensación económica en función de la gravedad de la situación:
 - Grado 1 (x10 veces el pago mensual): sufre un incidente de seguridad grave (robo de cuentas de usuarios, correos, proyectos...) o se produce una caída del servicio no relativa a mantenimiento (previo aviso) durante más de 24 horas.
 - Grado 2 (x2 veces el pago mensual): sufre un incidente de seguridad leve o se produce una caída del servicio no relativa a mantenimiento de usuario (previo aviso) de más de 8 horas.
 - Grado 3 (x1 vez el pago mensual): no se da soporte al tiempo de respuesta.
 - **Características:** solamente los planes de pago tendrán acceso a todas las funcionalidades de la aplicación. Esto permite diferenciar
- **OpEX**

En este apartado, se pretende analizar el coste de operación por cada plan asociado a un usuario.

Primero, se ha analizado el número de peticiones que se realizan a los respectivos proveedores de servicios en la nube para cada solicitud o petición del usuario. Para generalizar el caso de

estudio, se ha usado el caso más simple de todos, en el que se toman las siguientes consideraciones:

- **1 consulta a BBDD desde el endpoint:** dado que existen endpoints que consultan 1 vez (peticiones *get*, *delete* y *create*), 2 veces (peticiones *update*) o múltiples veces (peticiones anidadas mediante *FetchType.Lazy* en *Hibernate* [33]); se pretende estandarizar y homogeneizar por el caso más simple, en este caso, 1.
- **Microservicio de proyecto:** aunque existan múltiples microservicios, se va a tomar como referente el microservicio de proyectos por la utilización de BBDD de tipo SQL.
- **Servicios desplegados en AWS Gateway:** las aplicaciones de *Spring* se han transformado a expresiones *lambda* mediante *Spring Cloud Function Adapter*.

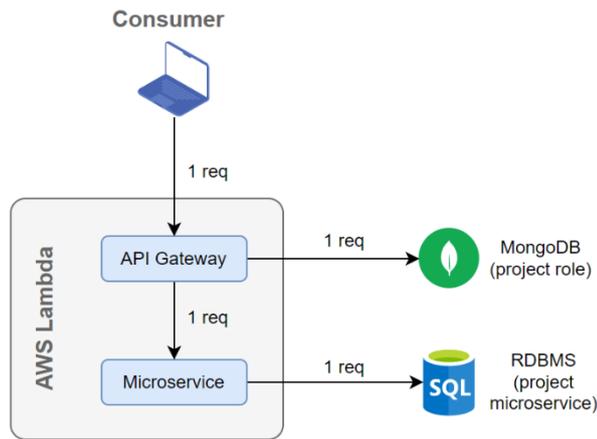


Imagen 44: Peticiones a los proveedores

De este gráfico, obtenemos que se realizan 2 peticiones a *AWS Lambda*, 1 petición a *MongoDB* y 1 petición a *RDBMS*. Para estos casos, se va a analizar el coste (en dólares, \$) por cada millón de peticiones que realiza un usuario.

Proveedor	Plan	Peticiones / solicitud	Precio (\$) / millón de sol.	Total (\$) / millón de sol.
AWS Lambda	AWS Lambda x86	2	0,2	0,4
MongoDB (Atlas)	MongoDB Atlas Serverless	1	0,1	0,1
RDBMS (AWS)	Amazon Aurora Postgres Serverless v2	1	0,2	0,2

Total	0,7 \$ / M. sol.
--------------	-------------------------

Tabla 3: Análisis del OpEX por millón de peticiones

Como puede apreciarse en la tabla anterior, se ha determinado que, por cada **millón de peticiones** entrantes de los diferentes usuarios **se gasta un total de 0,70\$**.

• Beneficios por plan

En este apartado, se va a analizar los beneficios que se obtiene por cada tipo de plan.

Antes de nada, deberá obtenerse precio por cada plan, *rates* por cada plan y precio por millón de peticiones. Estas fueron calculadas en los apartados anteriores.

Posteriormente, hay que analizar cuánto tiempo va a estar conectado un usuario a la aplicación, por lo que se han tomado las siguientes consideraciones en función del plan:

- **Free:** se orienta a un usuario que crea proyectos por su cuenta o autónomo que no requiere coordinación con otras personas, por lo que su tiempo de trabajo diario será de 3 horas diarias y un tiempo de dedicación a labores de gestión de un 5%.
- **Premium / Enterprise:** se orienta a un usuario profesional que trabaja 8 horas, de las cuales destinará un 10% a organizar sus tareas y coordinarse con el resto de sus compañeros.

A continuación, se calculará el beneficio mensual (20 días de trabajo) por cada uno de los planes de precio:

Plan	Horas / día	% dedicación	Mill. Peticiones / mes	Gasto (\$) / usuario mes	Beneficios (\$) / mes
Free	3	5%	0,0540	0,03780	-0,0378
Premium	8	10%	0,5760	0,40320	5,5968
Enterprise	8	10%	2,8800	2,01600	15,9840

Tabla 4: Beneficios por cada plan

Como puede indicarse en la tabla anterior, tanto los gastos como los beneficios dependerán del plan que tengan estos asociados. En el caso del plan *Free*, se pierde dinero y es necesario que se vea compensado del resto de planes de pago.

• Simulación

En esta sección, se simulará cómo afecta el beneficio en función de los usuarios por plan.

Para ello, se va a realizar una simulación con las siguientes características:

- **1000 usuarios en "Free":** un gran número de nuevos candidatos que disfrutan del servicio gratuito. Esto será usado también por universidades y otras entidades de enseñanza.
- **100 usuarios en "Premium":** usuarios independientes adquieren este servicio para organizar y extraer informes de su trabajo. Su número no será muy elevado porque no es el objetivo de ventas.
- **200 usuarios en "Enterprise":** gran número que disfrutan de todos los privilegios, debido a una adquisición por parte de sus empresas.

A continuación, se va a realizar la simulación teniendo en cuenta los gastos por usuario al mes y el precio del plan calculados previamente:

Plan	Nº usuarios	Gastos (\$) / mes	Ingresos / mes	Beneficios (\$) / mes
Free	1000	37,80	0	-37,80
Premium	100	40,32	600	559,68
Enterprise	200	403,20	3600	3196,80
Total	1300	481,32	4200	3718,68

Tabla 5: Simulación de planes de precio por usuarios

Como puede apreciarse, en esta simulación se obtendría un **beneficio mensual de 3.718,18\$** sin tener en cuenta la mano de obra y otros gastos. Y, por otra parte, se obtendría una gran visibilidad a cambio de reducir las ganancias ligeramente en el plan "Free".

En conclusión, se han establecidos 3 planes económicamente viables, que permiten obtener beneficios y, a su vez, obtener visibilidad dentro del sector mediante un plan gratuito.

14. Validación

En este apartado, se va a verificar que el funcionamiento del sistema es el esperado.

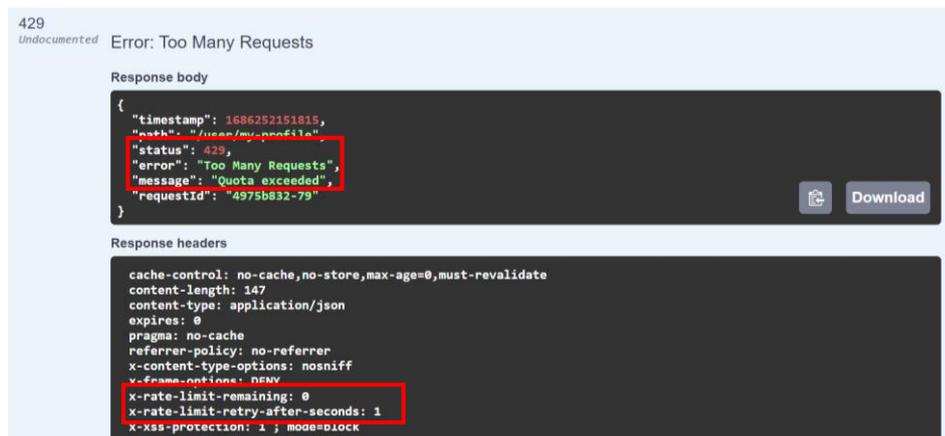
En primera instancia, se intentó realizar pruebas unitarias desde *API Gateway*, pero no pude llegarlo a configurar bien por un problema relativo a la base de datos *Mongo Atlas*. Es por ello, que se va a validar manualmente cada una de las casuísticas.

14.1. Validación de rates y cuotas

Se va a verificar que el algoritmo de *rates* y cuotas funciona correctamente. Para ello se va a alterar el fichero *plans.yaml* en el que se va a aumentar el tiempo de *rates* a 10 segundos, con el objetivo de poder realizar las pruebas.

Vamos a probar con las siguientes casuísticas:

- **Exceso de peticiones** (5 o más peticiones en 10 segundos): indica un error 429 *“Too Many Requests”* con el mensaje de *“Quota exceeded”*. Además, en la cabecera *“x-rate-limit-retry-after-seconds”* indica que reintente dentro de 1 segundo.



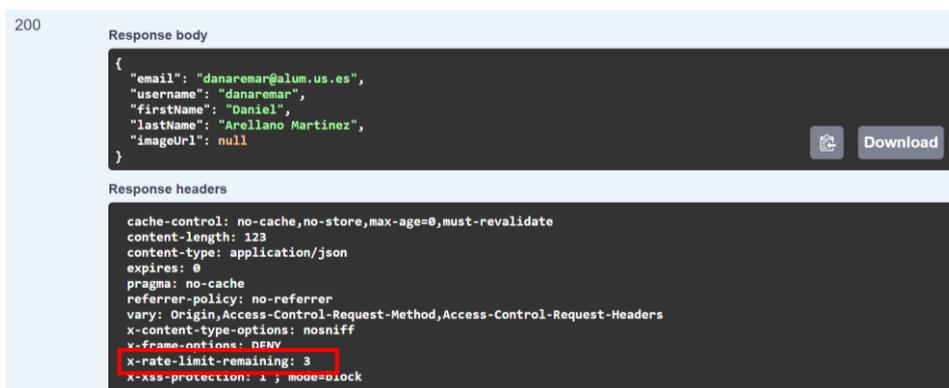
```
429
Undocumented Error: Too Many Requests

Response body
{
  "timestamp": 1686252151815,
  "path": "/user/my-profile",
  "status": 429,
  "error": "Too Many Requests",
  "message": "Quota exceeded",
  "requestId": "49750832-79"
}

Response headers
cache-control: no-cache,no-store,max-age=0,must-revalidate
content-length: 147
content-type: application/json
expires: 0
pragma: no-cache
referrer-policy: no-referrer
x-content-type-options: nosniff
x-frame-options: DENY
x-rate-limit-remaining: 0
x-rate-limit-retry-after-seconds: 1
x-xss-protection: 1 ; mode=block
```

Imagen 45: Validación de exceso de peticiones en el rate

- **Defecto de peticiones** (menos de 5 peticiones en 10 segundos): indica un código de respuesta 200 *“OK”* con la información que se ha solicitado. En la cabecera *“x-rate-limit-retry-after-seconds”* indica que le quedan 3 peticiones restantes para saturar el *rate*.



```
200

Response body
{
  "email": "danaremar@alum.us.es",
  "username": "danaremar",
  "firstName": "Daniel",
  "lastName": "Arellano Martinez",
  "imageUrl": null
}

Response headers
cache-control: no-cache,no-store,max-age=0,must-revalidate
content-length: 123
content-type: application/json
expires: 0
pragma: no-cache
referrer-policy: no-referrer
vary: Origin,Access-Control-Request-Method,Access-Control-Request-Headers
x-content-type-options: nosniff
x-frame-options: DENY
x-rate-limit-remaining: 3
x-xss-protection: 1 ; mode=block
```

Imagen 46: Validación de defecto de peticiones en el rate

- **Cambio de plan** (cambio a “premium”): se cambia la suscripción del usuario al plan “premium”, incrementando el *rate* del usuario. Como el *rate* es de 10 peticiones cada 10 segundos, puede observarse que tras consumir 1 petición indica que quedan pendiente 9 peticiones en la cabecera “*x-rate-limit-retry-after-seconds*”.

```

200
Response body
{
  "email": "danaremar@alum.us.es",
  "username": "danaremar",
  "firstName": "Daniel",
  "lastName": "Arellano Martinez",
  "imageUrl": null
}
Response headers
cache-control: no-cache, no-store, max-age=0, must-revalidate
content-length: 123
content-type: application/json
expires: 0
pragma: no-cache
referrer-policy: no-referrer
vary: Origin, Access-Control-Request-Method, Access-Control-Request-Headers
x-content-type-options: nosniff
x-frame-options: deny
x-rate-limit-remaining: 9
x-xss-protection: 1; mode=block

```

Imagen 47: Validación de un cambio de plan en el rate

14.2. Validación de permisos

Se va a verificar que el funcionamiento de roles y permisos es el esperado en la aplicación para un usuario dado.

Primero, se creará un usuario y se le asignará a un proyecto con el rol “member”. A continuación, se va a verificar que solamente puede acceder a los *endpoints* o recursos a los que tiene acceso para el proyecto al que ha sido asignado:

- **Endpoint “owner”**: al no tener los suficientes privilegios, deniega la petición. Devuelve un estado 403 indicando que el acceso está denegado.

```

POST /project/{projectId}/projects/testOwner
Parameters
Name Description
projectId * required 641e0618ead46d1d43fbab49
string (path)
Execute Clear
Responses
Curl
curl -X 'POST' \
  -H 'accept: */*' \
  -H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpzZW50L3VzZXQ6IiwiaWF0IjoiY2M2VnNk...' \
  -d ''
Request URL
http://localhost:8080/project/641e0618ead46d1d43fbab49/projects/testOwner
Server response
Code Details
403 Undocumented Error: Forbidden
Response body
{
  "timestamp": "2023-06-14T11:49:07.537+00:00",
  "status": "403",
  "error": "Forbidden",
  "trace": "org.springframework.security.access.AccessDeniedException: Acceso denegado\n\tat org.springframework.security.access.vote.AffirmativeBased.decide(AffirmativeBased.java:75)\n\tat org.springframework.security.access.intercept.AbstractSecurityInterceptor.attemptAuthorization(AbstractSecurityInterceptor.java:231)\n\tat org.sp..."
}

```

Imagen 48: Verificar acceso fallido a un recurso que requiere el rol Owner

- **Endpoint “admin”**: al no tener los suficientes privilegios, deniega la petición al igual que en el caso anterior.
- **Endpoint “member”**: dispone de privilegios y el servidor le responderá de forma correcta. Devuelve un estado 200 indicando que todo está correcto.

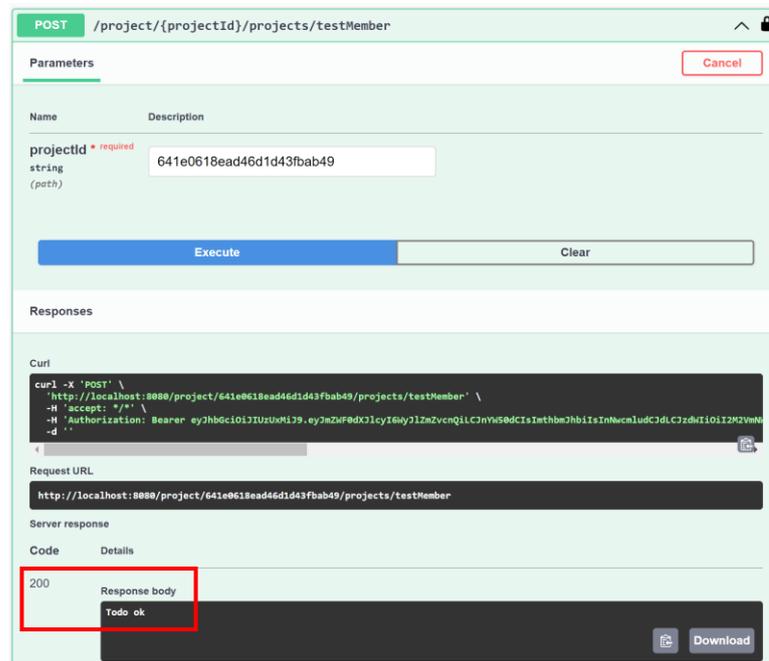


Imagen 49: Verificar acceso satisfactorio que requiere el rol Member

- **Endpoint “visitor”**: dispone de privilegios y el servidor le responderá de forma correcta, al igual que se indica en el caso anterior.

PARTE IV: CONCLUSIONES

15. Cumplimiento de objetivos

En este punto, se va a analizar si se han cumplido los objetivos propuestos al comienzo del proyecto, para ello se crea una tabla, en la cual se establezcan cuáles se han cumplido y cuales no, y una descripción de la razón por la que se ha cumplido, si procede:

OBJETIVO	Cumplimiento
OBJ-01: Enfocar producto comercialmente Se ha creado un <i>pricing</i> con 3 planes que contienen precios, funcionalidades (feature toggles) y limitaciones (<i>rates</i> y cuotas) asociadas a cada uno de ellos. Además, se ha intentado obtener la infraestructura más rentable económicamente.	✓
OBJ-02: Analizar el coste de operación Se ha analizado el OpEX teórico en base a la arquitectura a desplegar y se han simulado ganancias para un cierto número de licencias en cada uno de los diferentes planes de precio.	✓

Como podrá observarse en la tabla anterior, se han cumplido todos los objetivos propuestos desde el principio del proyecto, indicándonos que se han cumplido todas las expectativas del principio.

16. Esfuerzo y costes totales

16.1. Esfuerzo empleado

En esta sección, se va a tratar sobre el esfuerzo empleado finalmente en el desarrollo del proyecto desde el comienzo hasta el final de éste.

Para la imputación del tiempo se ha usado la aplicación [Toggl](#) mediante el plan gratuito. El cómputo desglosado del tiempo se adjuntará como anexo, tal y como se indica al final de este documento. A modo de resumen, se puede decir que se han empleado 290 horas en total en el proyecto, siendo el resumen del informe del cómputo del tiempo:



Imagen 50: Reporte de tiempo imputado en Toggl

A continuación, se va a analizar la similitud entre el esfuerzo planificado previamente y el esfuerzo empleado en la construcción de la solución. Para ello, se ha representado una gráfica de barras que indicará el número de horas de dedicación por cada una de las áreas.

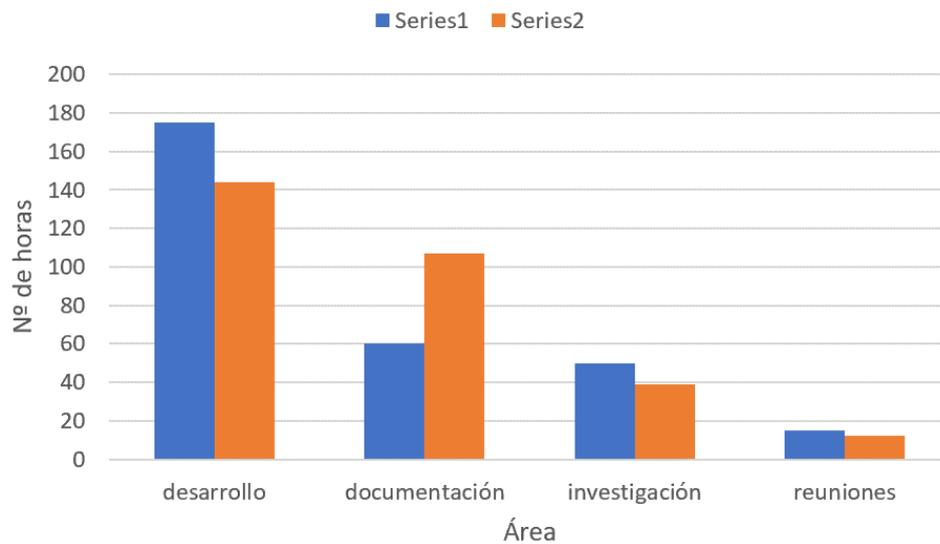


Imagen 51: Comparación tiempo planificado y tiempo empleado

Como puede apreciarse, la desviación más grande es provocada por la documentación, debido a un refinamiento de la memoria durante las últimas semanas. En el resto de aspectos, puede observarse que es similar a lo planificado.

16.2. Costes totales

En esta sección, se va a analizar el coste total del proyecto realizado.

Para su cálculo, se multiplicará el precio por hora (establecido en el apartado 2.4 con un valor de 22,21€/hora) por el número total de horas empleadas (305 horas), resultando un coste sin impuestos de 6.774,05€ y un **coste total con impuestos de 8196,60€**.

Como podrá apreciarse, el coste real es ligeramente superior al coste planificado, siendo una aproximación con un margen de error de 1,17% al alza, lo cual indica que la estimación de tiempo ha sido bastante buena y no han surgido problemas durante el desarrollo de este proyecto.

17. Mis conclusiones

En este apartado, quería hacer una reflexión general sobre el proyecto desarrollado durante esta última etapa universitaria.

Para comenzar, quiero mencionar que el proyecto tenía 2 objetivos (enfocar un producto comercialmente y analizar su coste de operación) y ambos se han cumplido. Esto me ha permitido aprender sobre las áreas de análisis de costes y la viabilidad de un producto software, profundizando así los conceptos que se imparten en la rama de *cloud* del máster. Por otro lado, aunque no haya dado tiempo a la construcción e implantación, quiero seguir su desarrollo y comercializarlo en algún tiempo cercano (sobre 3 meses según mi planteamiento, entre las vacaciones laborales y la no asistencia a clases).

Con respecto su evolución, al principio fue algo lenta y desmotivadora porque no llegaba a implementar una solución, ya que el mayor esfuerzo se destinaba a la investigación. Posteriormente, me sentí más motivado y desempeñé mejor mi rendimiento, ya que destinaba más esfuerzo al desarrollo, y se manteniéndose el ritmo hasta el final (salvo la excepción de abril por la cantidad de festivos).

En resumen, me ha parecido una muy buena experiencia y me ha ayudado a comprender en mayor profundidad conceptos como *pricing*, *OpEX*, *rate limits*, *feature toggles* y arquitectura de microservicios basadas en *API Gateways*.

Considero que sentará la base para una futura versión que pueda comercializar, quedando pendiente una implementación de un rediseño de la interfaz gráfica y definir estrategias de marketing y empresariales para el lanzamiento del producto *SaaS*.

Quiero agradeceros la lectura de este documento, y espero que lo hayáis disfrutado en mayor medida de lo posible. Ante cualquier sugerencia o duda, no dudéis en poneros en contacto conmigo: daniellarellano99@gmail.com / danaremar@alum.us.es .

PARTE V: BIBLIOGRAFÍA CONSULTADA

18. Bibliografía

- [1] [En línea]. Available: https://www.glassdoor.es/Sueldos/cloud-architect-sueldo-SRCH_KOO,15.htm#:~:text=%C2%BFCu%C3%A1nto%20gana%20un%20Cloud%20Architect,Architect%20es%20de%20%E2%82%AC49.721%20..
- [2] M. Rehkopf, «Atlassian,» [En línea]. Available: <https://www.atlassian.com/es/agile/scrum/sprints#:~:text=son%20los%20sprints%3F-,Un%20sprint%20es%20un%20per%C3%ADodo%20breve%20de%20tiempo%20fijo%20en,c on%20menos%20quebraderos%20de%20cabeza..> [Último acceso: 18 Abril 2023].
- [3] M. Rehkopf, «Atlassian,» [En línea]. Available: <https://www.atlassian.com/agile/project-management/epics-stories-themes>. [Último acceso: 18 Abril 2023].
- [4] M. Rehkopf, «Atlassian,» [En línea]. Available: <https://www.atlassian.com/es/agile/project-management/user-stories>. [Último acceso: 18 Abril 2023].
- [5] «Wrike,» [En línea]. Available: <https://www.wrike.com/agile-guide/faq/what-is-a-milestone-in-agile/>. [Último acceso: 2023 Abril 18].
- [6] [En línea]. Available: <https://www.sap.com/insights/what-is-product-lifecycle-management.html>.
- [7] R. F. Aranda, «Sistedes,» [En línea]. Available: <https://biblioteca.sistedes.es/submissions/descargas/2022/JCIS/2022-JCIS-032.pdf>. [Último acceso: 2023 Abril 19].
- [8] P. F. A. D. A. R.-. C. Rafael Fresno-Aranda, de *Semi-automated capacity analysis of limitation-aware microservices architectures*.
- [9] [En línea]. Available: <https://www.predicagroup.com/blog/what-are-feature-flags-and-how-to-use-them/#experimental>.
- [10] P. Hodgson. [En línea]. Available: <https://martinfowler.com/articles/feature-toggles.html>.
- [11] VMware, «VMware,» [En línea]. Available: <https://www.vmware.com/es/topics/glossary/content/microservices.html#:~:text=Los%20microservicios%20son%20los%20miles,del%20desarrollo%20de%20software%20contempor%C3%A1neo..>
- [12] Microsoft. [En línea]. Available: <https://learn.microsoft.com/es-es/azure/architecture/microservices/model/tactical-ddd>.
- [13] «Wallarm,» [En línea]. Available: <https://www.wallarm.com/what/the-concept-of-an-api-gateway>. [Último acceso: 18 Abril 2023].
- [14] D. B. Ospina, «Medium,» 28 Octubre 2020. [En línea]. Available: <https://medium.com/bancolombia-tech/potenciando-la-arquitectura-de-microservicios-reactivos-88fc84ae0b7d>.

- [15] Canonical, «Spring,» [En línea]. Available: <https://spring.io/projects/spring-cloud>.
- [16] «Angular,» [En línea]. Available: <https://material.angular.io/components/categories>. [Último acceso: 21 Abril 2023].
- [17] Microsoft, «Microsoft,» 23 Septiembre 2022. [En línea]. Available: <https://learn.microsoft.com/es-es/windows/win32/cosssdk/acid-properties>.
- [18] I. C. Foundation, «IBM,» 14 Noviembre 2019. [En línea]. Available: <https://www.ibm.com/es-es/cloud/learn/cap-theorem>.
- [19] maoyunfei, «Github,» 25 Abril 2018. [En línea]. Available: <https://github.com/spring-cloud/spring-cloud-gateway/issues/301>. [Último acceso: 20 Abril 2023].
- [20] «AWS,» [En línea]. Available: <https://aws.amazon.com/es/lambda/pricing/>. [Último acceso: 20 Abril 2023].
- [21] Alexander, «dev.to,» 20 Diciembre 2022. [En línea]. Available: <https://dev.to/aleksk1ng/spring-webflux-and-grpc-319l>. [Último acceso: 20 Abril 2023].
- [22] «Atlassian,» [En línea]. Available: <https://www.atlassian.com/es/itsm/it-asset-management/cmdb>. [Último acceso: 2023 Abril 20].
- [23] P. F. A. D. A. R.-C. Rafael Fresno-Aranda, de *Semi-automated Capacity Analysis of Limitation-Aware Microservices Architectures*.
- [24] «Spring,» [En línea]. Available: https://cloud.spring.io/spring-cloud-gateway/2.1.x/multi/multi__configuration.html. [Último acceso: 21 Abril 2023].
- [25] «Spring,» [En línea]. Available: https://cloud.spring.io/spring-cloud-gateway/multi/multi__global_filters.html. [Último acceso: 21 Abril 2023].
- [26] [En línea]. Available: <https://www.npmjs.com/package/openapi-merge>.
- [27] baeldung, «baeldung,» 20 Mayo 2022. [En línea]. Available: <https://www.baeldung.com/spring-security-method-security>. [Último acceso: 21 Abril 2023].
- [28] «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Token_bucket. [Último acceso: 2023 Abril 22].
- [29] «Okta,» [En línea]. Available: <https://developer.okta.com/docs/reference/rl-best-practices/#example-rate-limit-header-with-org-wide-rate-limit>. [Último acceso: 2023 Abril 23].
- [30] «IBM,» 24 Octubre 2022. [En línea]. Available: <https://www.ibm.com/docs/en/datapower-gateway/10.0.1?topic=processing-api-rate-limit-action>. [Último acceso: 23 Abril 2022].
- [31] «Microsoft,» [En línea]. Available: <https://www.microsoft.com/es-es/microsoft-365/try>. [Último acceso: 2023 Mayo 22].

- [32] «Toggl,» [En línea]. Available: <https://toggl.com/plan/pricing>. [Último acceso: 2023 Mayo 22].
- [33] baeldung, «baeldung,» 2023 Mayo 21. [En línea]. Available: <https://www.baeldung.com/hibernate-lazy-eager-loading>.
- [34] «Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/GNU_General_Public_License.
- [35] [En línea]. Available: https://upload.wikimedia.org/wikipedia/commons/c/ca/MariaDB_colour_logo.svg.
- [36] [En línea]. Available: https://es.wikipedia.org/wiki/Sistema_de_gesti%C3%B3n_de_bases_de_datos_relacionales.
- [37] [En línea]. Available: <https://mariadb.org/>.
- [38] [En línea]. Available: https://upload.wikimedia.org/wikipedia/commons/b/b5/DBeaver_logo.svg.
- [39] [En línea]. Available: <https://maven.apache.org/>.
- [40] [En línea]. Available: <https://spring.io/projects/spring-framework>.
- [41] [En línea]. Available: <https://www.campusmvp.es/recursos/post/la-api-de-persistencia-de-java-que-es-jpa-jpa-vs-hibernate-vs-eclipselink-vs-spring-jpa.aspx>.
- [42] [En línea]. Available: <https://www.redhat.com/es/devops/what-is-agile-methodology>.
- [43] [En línea]. Available: <https://es.wikipedia.org/wiki/Servidor#:~:text=Un%20servidor%20es%20un%20conjunto,in%20dividualmente%20como%20C2%ABel%20servidor%20BB..>
- [44] [En línea]. Available: <https://www.bbvaapimarket.com/es/mundo-api/api-rest-que-es-y-cuales-son-sus-ventajas-en-el-desarrollo-de-proyectos/>.
- [45] [En línea]. Available: https://es.wikipedia.org/wiki/Entorno_de_desarrollo_integrado.
- [46] [En línea]. Available: <https://www.java.com/es/download/help/develop.html>.
- [47] [En línea]. Available: https://es.wikipedia.org/wiki/Java_Persistence_API.
- [48] [En línea]. Available: <https://brilliant.org/wiki/secure-hashing-algorithms/#:~:text=Secure%20Hash%20Algorithms%2C%20also%20known,modular%20aditions%2C%20and%20compression%20functions..>
- [49] [En línea]. Available: <https://nodejs.org/es/>.
- [50] [En línea]. Available: <https://angular.io/docs>.
- [51] G. Inc.. [En línea]. Available: <https://www.electronjs.org/docs/tutorial/introduction>.

- [52] [En línea]. Available: <https://agilemanifesto.org/iso/es/manifesto.html>.
- [53] [En línea]. Available: <https://www.bbva.com/es/metodologia-agile-la-revolucion-las-formas-trabajo/>.
- [54] SAP, «SAP,» [En línea]. Available: <https://www.sap.com/insights/what-is-product-lifecycle-management.html>.

19. Imágenes

Imagen 1: Proceso metodológico en el TFM	13
Imagen 2: Planificación del proyecto mediante RUP.....	14
Imagen 3: Iniciativas, épicas e historias de usuario según Atlassian [3] [4]	19
Imagen 4: Propuesta PLM de SAP [6].....	20
Imagen 5: Ejemplo de LAMA propuesto por Rafael Fresno [7]	22
Imagen 6: Tipos de feature toggles [10]	23
Imagen 7: Ejemplo de segmentación en microservicios [12]	24
Imagen 8: Pasos en Drone Delivery [12].....	24
Imagen 9: API Gateway [13].....	25
Imagen 10: Modelo reactivo vs modelo bloqueante [14]	26
Imagen 11: Logotipo de Angular.....	27
Imagen 12: Algunos componentes de Material Angular [16].....	27
Imagen 13: UML usuarios y proyectos.....	34
Imagen 14: UML empresa.....	34
Imagen 15: UML suscripción.....	35
Imagen 16: UML sprint.....	35
Imagen 17: UML kanban	36
Imagen 18: UML efforts	36
Imagen 19: Mockup de acceso	37
Imagen 20: Mockup listado de proyectos.....	37
Imagen 21: Mockup actualizar proyecto	38
Imagen 22: Mockup listado de sprints.....	38
Imagen 23: Actualizar sprint	38
Imagen 24: Mockup mostrar kanban.....	39
Imagen 25: Mockup actualizar columna.....	39
Imagen 26: Mockup actualizar tarea	39
Imagen 27: Mockup listado de esfuerzos	40
Imagen 28: Mockup actualización de esfuerzo.....	40
Imagen 29: Arquitectura y tecnologías para la construcción	41
Imagen 30: Diagrama de funcionalidad de microservicios.....	43
Imagen 31: Evolución del coste en función de las peticiones del cliente.....	44
Imagen 32: Repositorio en Github	45
Imagen 33: Github for Windows.....	46
Imagen 34: Canal TFM en Microsoft Teams	46
Imagen 35: Reunión de seguimiento en Teams.....	47
Imagen 36: Carpeta TFM en OneDrive.....	47
Imagen 37: Mongo Atlas.....	48
Imagen 38: DBeaver	49
Imagen 39: Visual Studio Code	50
Imagen 40: Petición de prueba de redireccionamiento	57
Imagen 41: Documentación REST del microservicio de proyectos.....	59
Imagen 42: Documentación de todos los servicios en OpenAPI	61
Imagen 43: Token JWT generado en la aplicación.....	64
Imagen 44: Peticiones a los proveedores	72
Imagen 45: Validación de exceso de peticiones en el rate	75
Imagen 46: Validación de defecto de peticiones en el rate.....	75

Imagen 47: Validación de un cambio de plan en el rate.....	76
Imagen 48: Verificar acceso fallido a un recurso que requiere el rol Owner	76
Imagen 49: Verificar acceso satisfactorio que requiere el rol Member	77
Imagen 50: Reporte de tiempo imputado en Toggl.....	80
Imagen 51: Comparación tiempo planificado y tiempo empleado	81

20. Código

Código 1: JSON (archivo de configuración) de la herramienta de SmartLAMA Rafael Fresno [7]	51
Código 2: plans.yaml	51
Código 3: settings.gradle padre	53
Código 4: build.gradle padre.....	53
Código 5: Spring Initializr para generar un microservicio	54
Código 6: build.gradle en manapi-project	54
Código 7: build.gradle en manapi-gateway	55
Código 8: application.yml en manapi-gateway.....	56
Código 9: GlobalPreFiltering en manapi-gateway	56
Código 10: SecurityFilter en manapi-project	57
Código 11: Permiso en la petición del microservicio consultado	57
Código 12: Configuración de OpenAPI en el microservicio de proyectos	58
Código 13: Anotaciones de OpenAPI en el controlador de Sprints	58
Código 14: Método para extraer especificación de OpenAPI.....	60
Código 15: Método para combinar especificaciones de OpenAPI.....	60
Código 16: Método para unificar todas las especificaciones de OpenAPI.....	61
Código 17: Modelo de seguridad a capas propuesto	62
Código 18: Funciones para generar tokens JWT en JwtService.....	63
Código 19: Configuración JWT en WebSecurityConfig	64
Código 20: Configuración JWT en JwtAuthenticationManager	64
Código 21: Función validateToken en JwtService	65
Código 22: Limitar rates y cuotas en RateFilter	65
Código 23: Nuevo bucket en RateFilter	66
Código 24: Filtrar en función del bucket en RateFilter	67
Código 25: Configuración de rutas en application.yml en API Gateway	68
Código 26: Función apply() en FeatureGroupGatewayFilterFactory	68

ANEXOS

- **Desglose de horas imputadas**

En el documento llamado “*TFM danaremar - efforts.pdf*” se indicará un desglose más detallado del tiempo imputado en el proyecto.

- **Código fuente**

En el documento llamado “*TFM danaremar - sources.zip*” se indicará un fichero zip con todos los códigos fuentes de la aplicación. Además, podrá ser accesible desde:

<https://github.com/danaremar/manapi>