Grado en Ingeniería Informática

Trabajo de Final de Grado

Motor de recomendación parametrizable

Autor:
Jaume Barrios Forner

Supervisor:
Sergio Aguado González
Tutor académico:
Oscar Belmonte Fernández

Fecha de lectura: 13 de Julio de 2021
Curso académico 2020/2021

Resumen

Memoria técnica del trabajo de fin de grado realizado por Jaume Barrios Forner en la que se describe el proyecto de su estancia de prácticas en la empresa tecnológica Cuatroochenta.

El proyecto realizado consiste en la implementación de un motor de recomendación con una API (*Application Programming Interface*) fundamentada en los principios REST (*Representational State Transfer*). El motor ofrece herramientas para poder entrenar distintos algoritmos de generación de recomendaciones apoyados en el filtrado colaborativo y filtrado basado en el contenido.

El desarrollo del proyecto se ha realizado siguiendo la metodología ágil *Scrum* y se ha desarrollado a lo largo de 7 *sprints* de una duración aproximada de dos semanas.

Palabras clave

API REST, motor de recomendación, filtrado colaborativo, filtrado basado en contenido, Scrum, metodologías ágiles.

Keywords

REST API, recommendation engine, collaborative filtering, content-based filtering, Scrum, agile development.

Índice general

Índice de cuadros

Índice de figuras

Capítulo 1

Introducción

1.1. Contexto y motivación del proyecto

1.1.1. Descripción de la empresa

Cuatroochenta es una empresa tecnológica con distintas oficinas distribuidas por España y América, cuya sede principal está ubicada en el edificio Espaitec 2 dentro del campus de la universidad Jaume I de Castelló de la Plana.

Cuatroochenta, en sus orígenes, era una empresa especializada en el desarrollo de aplicaciones para teléfonos móviles, pero, a día de hoy, ha tenido un enorme crecimiento y abarca desde el desarrollo de software a nivel empresarial hasta la realización de auditorías de ciberseguridad.

1.1.2. Estado del arte

Actualmente muchas empresas utilizan motores de recomendación para mejorar la experiencia de sus clientes y así aumentar las ventas. Dependiendo del tamaño de la empresa, implementan su propio sistema de recomendación enfocado a sus necesidades y al tipo de datos de los que disponen. Sin embargo, muchas de ellas no cuentan con los recursos necesarios para tener esta tecnología a su alcance.

Dada esta necesidad, han surgido numerosas soluciones, como por ejemplo, *recombee*. Estas soluciones, al igual que se pretende hacer en este proyecto, ofrecen implementaciones propias de motores de recomendación adaptadas a diversos tipos de datos. En cuanto a la configuración de los sistemas de recomendación, la mayoría de estas ofrecen la posibilidad de configurarlos desde plataformas web y API REST.

Aunque existan alternativas similares en el mercado, la solución que se pretende ofrecer en este proyecto se diferenciará en su capacidad de personalización de los algoritmos utilizados.

1.1.3. Motivación y utilidad

Este proyecto será utilizado de manera interna en los distintos servicios de Cuatroochenta para ofrecer recomendaciones de sus productos. Además, la empresa Easygoband, especialista en tecnologías NFC (*Near-Field Communication*) aplicadas al control de acceso y pago de servicios en eventos y cadenas hoteleras, pretende contratar este servicio para incorporar el motor de recomendación en uno de los hoteles de la famosa cadena hotelera Hilton, para así mejorar el ticket medio de sus clientes.

1.2. Objetivos del proyecto

El principal objetivo de este proyecto es la creación de un sistema de recomendación parametrizable de calidad y fácil de utilizar.

Dicho objetivo se puede desglosar en los siguientes subobjetivos:

- Ofrecer recomendaciones de distintos tipos de productos basándose en el perfil del usuario.
- Facilitar diversos algoritmos de recomendación.
- Hacer que se pueda integrar fácilmente en cualquier aplicación.
- Abstraer los detalles técnicos de la implantación del sistema de recomendación.
- Ofrecer la posibilidad de configurar todos los detalles del sistema.
- Aumentar el *ticket* medio[1].
- Generar las recomendaciones de manera eficiente.

En cuanto al alcance funcional del proyecto, las funcionalidades que se van a incluir en él son:

- Acceso al sistema de recomendación a través de una API basada en una arquitectura REST.
- Documentación interactiva basada en el estándar *OpenAPI*.
- Posibilidad de seleccionar el algoritmo de recomendación que se va a utilizar entre distintas familias de algoritmos.
- Aprendizaje a partir de los intereses de los usuarios.
- Realización de operaciones CRUD (Crear, Leer, Actualizar y Borrar) sobre los datos de los usuarios.

[1] El *ticket* medio es el dinero promedio gastado por un cliente

- Entreno del algoritmo de recomendación con los datos actualizados.
- Generación de recomendaciones a partir de un modelo indicado.
- Gestión los distintos modelos entrenados por el usuario.
- Control del acceso mediante una *API key*.
- Control del proceso de entrenamiento mediante tokens.

No entra dentro del alcance funcional:

- La generación de informes de análisis del negocio en base a los datos usados para el entrenamiento de los algoritmos de recomendación.
- Integración en el sistema de Easygoband.

En cuanto al alcance organizativo, la API será utilizada de manera interna por los encargados de integración los cuales la van a configurar para que esta ofrezca recomendaciones en las aplicaciones de la empresa. También será usada por la startup Easygoband que integrará la API con sus servicios.

Finalmente, en cuanto al alcance informático, la API deberá de interactuar con las aplicaciones que la implementan para ofrecer sus servicios, además, también interactuará con las bases de datos de estos para poder entrenar sus algoritmos de recomendación de manera periódica con datos actualizados.

1.3. Descripción del proyecto

El proyecto consiste en la realización de un motor de recomendación parametrizable el cual tiene como objetivo ser de uso general, para así, poder adaptarse a las necesidades del cliente a la vez que le facilita la integración en sus productos. De este modo, se abstraen los detalles del funcionamiento de los algoritmos y solo se tienen que centrar en ofrecer los datos requeridos por el motor.

1.4. Estructura de la memoria

En el capitulo 2, se muestra la metodología seguida en el proyecto, los recursos tecnológicos que se pretenden usar, la estimación de los recursos y los costes requeridos, así como el seguimiento este. En el capítulo 3, se realiza el análisis del sistema donde se precisan los requisitos, los casos de uso y la arquitectura del proyecto. En el capítulo 4, se plasman los detalles de la implementación de los distintos algoritmos de recomendación y se enumeran las distintas pruebas de verificación y validación que se han realizado sobre el sistema. En el capítulo 5, se muestran las conclusiones del proyecto.

Capítulo 2

Planificación del proyecto

2.1. Metodología

Este proyecto utilizará la metodología ágil Scrum porque los requisitos exigidos pueden variar debido a que existe un alto componente de aprendizaje e investigación, tanto de tecnologías como del estado del arte de los sistemas de recomendación.

Dentro de los roles de la metodología Scrum se distinguen tres: product owner, Scrum master y equipo de desarrollo. Lo habitual es que el Scrum master sea uno de los miembros del equipo de desarrollo, aunque con menos frecuencia, puede también ser el propio product owner. En este caso, es Sergio Aguado quien se encarga de liderar las reuniones de seguimiento del sprint y los dayli Scrum a parte de ser el principal interesado en el proyecto y el encargado en la toma de decisiones relevantes.

En Scrum, las tareas a realizar se dividen en sprints, los cuales en el caso de este proyecto han sido de dos semanas laborales. En cuanto a las reuniones de inicio de sprint, se han realizado todos los lunes. En ellas, se decidía cuales iban a ser las tareas a realizar durante las próximas dos semanas. Por otro lado, en las reuniones de cierre de sprint se mostraba una demo del progreso del proyecto al product owner y se comentaban las posibles mejoras y sugerencias de cambio.

A parte de las reuniones de inicio y fin de sprint, en Scrum también se suelen realizar reuniones diarias breves de alrededor de 5 minutos, conocidas como *dayli Scrum*. En este proyecto, al ser solamente una persona la encargada del desarrollo, estas se hacían únicamente los lunes juntamente con otros miembros del departamento de la empresa.

2.2. Planificación

En esta sección se detallará la primera versión de la pila del producto, en la que se enumerarán las historias de usuario ordenadas por prioridad y estimadas mediante puntos de historia.

Para la estimación de los puntos de historia, se ha utilizado como criterio principal el esfuerzo en días de trabajo necesarios para poder finalizarla.

En la tabla 2.1 se encuentra la pila inicial del proyecto donde se detallan todas las historias de usuario y su peso en puntos de historia. En esta, las distintas historias de usuario se han organizado en función de la importancia y de su peso en puntos de historia. Por ejemplo, si una historia de usuario es considerada importante por el cliente y tiene asociados una gran cantidad de puntos de historia, esta irá por delante de las demás.

A parte de los criterios anteriormente mencionados, la pila inicial del proyecto ha sido revisada por el *product owner* para asegurar que todas las historias presentes y su orden están concordes con sus necesidades.

ID	Descripción	Puntos de historia
HU01	Como *product owner* quiero que la API ofrezca algoritmos de recomendación por filtrado colaborativo item-item para que los sistemas que la integren puedan hacer recomendaciones en función de las valoraciones de los productos consumidos.	**13PH**
HU02	Como *product owner* quiero que la API ofrezca algoritmos de recomendación por filtrado de contenido para que los sistemas que la integren puedan hacer recomendaciones en función de las características de los productos.	**13PH**
HU03	Como *product owner* quiero que cada cliente solo pueda ver los datos y modelos suyos para evitar problemas de privacidad.	**3PH**
HU04	Como *product owner* quiero que para acceder a la API se requiera de una clave de autenticación única asociada a cada cliente para garantizar la seguridad de sus datos.	**2PH**
HU05	Como encargado de integración de la API quiero que esta me permita acceder a una documentación basada en el estándar OpenApi para poder documentarme sobre su uso.	**8PH**
HU06	Como encargado de integración de la API quiero que esta me permita crear distintos modelos de recomendación en base a los algoritmos ofrecidos para poder usarlos en distintos contextos.	**4PH**
HU07	Como encargado de integración de la API quiero poder enviar los datos requeridos por los usuarios en un JSON para que el modelo de recomendación se mantenga actualizado.	**5PH**
HU08	Como encargado de integración de la API quiero poder configurar los parámetros que se van a utilizar durante el entrenamiento del algoritmo de recomendación para que este funcione de la manera más óptima.	**2PH**
HU09	Como encargado de integración de la API quiero poder indicar que se entrene el algoritmo de recomendación con los datos actualizados para que este ofrezca mejores recomendaciones.	**2PH**
HU10	Como encargado de integración de la API quiero poder indicar que se elimine un determinado modelo juntamente a los datos almacenados para liberar espacio en memoria.	**1PH**

HU11	Como usuario de una de las aplicaciones que implementan la API quiero recibir recomendaciones que me sean relevantes para mejorar mi experiencia de uso.	**3PH**
HU12	Como encargado de integración de la API quiero poder acceder a informes en los que se indiquen estadísticas del nivel de precisión del modelo entrenado para poder comprobar si el algoritmo rendirá como es de esperar.	**5PH**
HU13	Como propietario de una de las aplicaciones que implementan la API quiero poder acceder a informes en los que se muestren estadísticas sobre las recomendaciones generadas por la API para así visualizar cuáles son los gustos de mis clientes.	**13PH**

Cuadro 2.1: Pila inicial del proyecto

2.3. Identificación y gestión de riesgos

Una parte fundamental a la hora de planificar un proyecto es la identificación y gestión de los riesgos que pueden surgir durante el desarrollo de este. Detectar los riesgos más probables facilita su prevención y agiliza la gestión de estos, minimizando así el impacto que pueden causar tanto a la duración del proyecto como a su coste, entre otras cosas.

En la tabla 2.2 se identifican los posibles riesgos del proyecto y su gestión. En esta podemos identificar los siguientes campos:

- **ID**: Identificador único del riesgo
- **Descripción**: Breve descripción en la que se detalla en qué consiste el riesgo
- **Impacto**: El impacto que se estima que podría tener sobre el proyecto en el caso de que suceda el riesgo. Este es cuantificado en tres niveles:
  - **Alto**: En el caso de que suceda puede afectar seriamente al proyecto pudiendo desencadenar en el fracaso de este.
  - **Medio**: En el caso de que suceda puede hacer que no se logre alguno de los objetivos del proyecto o que este se retrase ligeramente.
  - **Bajo**: En el caso de que suceda el impacto sobre el proyecto es mínimo causando un posible retraso de pocos días.
- **Prevención**: Plan para prevenir que el riesgo suceda
- **Contención**: Plan para que en el caso de que uno de los riesgos identificados suceda este tenga un bajo impacto sobre el proyecto

Id	Riesgo	Impacto	Prevención	Contención
R01	Falta de experiencia en las tecnologías utilizadas.	Alto	Realizar cursillos de formación en las tecnologías que se utilizarán en el proyecto.	Se buscará a un empleado que tenga más experiencia en la tecnología para solventar el problema.
R02	Problemas de compatibilidad entre tecnologías.	Alto	Realizar un análisis previo de las tecnologías en el que se pruebe su correcto funcionamiento al usarse de manera conjunta.	Analizar posibles alternativas a la tecnología que causa problemas para reemplazarla.
R03	Ausencia de un miembro del equipo por problemas de salud.	Medio	Poner dentro de la duración total del proyecto un margen de varios días para recuperar el tiempo perdido.	Recortar funcionalidades no esenciales para poder llegar a la fecha de entrega.
R04	Falta de *feedback* por parte de los clientes.	Medio	Establecer reuniones de seguimiento de manera periódica para así recibir *feedback*.	Concertar una reunión para poder recibir *feedback* del estado del proyecto.
R05	Falta de conocimiento teórico sobre el funcionamiento de los motores de recomendación.	Bajo	Lectura de libros técnicos introductorios a los algoritmos utilizados por los motores de recomendación y la revisión del estado del arte.	Ponerse en contacto con expertos en el tema para resolver las dudas surgidas.

Cuadro 2.2: Identificación y gestión de los riesgos del proyecto.

2.4. Tecnologías

En esta sección se enumerarán las tecnologías que se tiene previsto utilizar en este proyecto y se justificará su uso. En concreto, se utilizarán únicamente tecnologías gratuitas de libre distribución para abaratar costes.

Las tecnologías y herramientas que se utilizarán son:

1. **Visual Studio Code[13]**. Es un editor de código desarrollado por Microsoft el cual es de libre uso y cuenta con una gran comunidad de desarrolladores que ha creado *pluguins* para trabajar con la mayoría de lenguajes de programación existentes en la actualidad. Por este motivo, se utilizará este editor ya que al tener acceso a tantas funcionalidades desde una única herramienta se agiliza el proceso de desarrollo enormemente.

2. **Python[18]**. Es un lenguaje de programación que se caracteriza por tener una sintaxis simple y leíble. Además, es el lenguaje más usado dentro del campo de la ciencia de datos

y la IA (Inteligencia Artificial), por lo que dispone de una gran cantidad de librerías y documentación. Por este motivo, dada la naturaleza del proyecto, este es el lenguaje de programación que mejor se adapta a las necesidades.

3. **FastAPI[16]**. Es un *framework* de desarrollo de API REST el cual se caracteriza por ser muy eficiente y fácil de aprender. Además, dispone de numerosas utilidades como la validación de datos o la generación automática de documentación OpenAPI a partir de la propia base de código. Por lo tanto, este *framework* agilizará el proceso de desarrollo de la API del proyecto.

4. **Pytest[15]**. Es un *framework* de creación de pruebas para *Python* el cual es recomendado por FastAPI ya que este dispone de herramientas compatibles con este framework que facilitan la realización de estas.

5. **MongoDB[10]**. Es una base de datos *NoSQL* la cual brinda una gran flexibilidad a la hora de estructurar la información, además dispone de un gestor de ficheros interno llamado *GridFS* el cual permite acceder a los ficheros de manera eficiente. Por este motivo se utilizará esta base de datos tanto para almacenar los datos requeridos para entrenar los modelos de recomendación como para almacenar los propios modelos.

6. **Docker[9]**. Es un software que permite desplegar aplicaciones dentro de contenedores donde se simula el entorno ideal para que estas funcionen correctamente. Este software se utilizará para agilizar el despliegue de la aplicación en la web.

7. **Celery[1]**. Es una librería de *Python* que permite crear colas de distribución de tareas entre múltiples procesos. Esta librería se utilizará para paralelizar el entrenamiento de modelos de recomendación ya que este es un proceso costoso.

8. **RabbitMQ[17]**. Es un software de envío de mensajes de altas prestaciones, este se usará para gestionar los procesos de entrenamiento de modelos de recomendación.

9. **Flutter[7]**. Es un SDK escrito en el lenguaje de programación *Dart*. Este permite la realización de aplicaciones gráficas multiplataforma de manera sencilla y eficiente. Debido a su sencillez de uso y aprendizaje es ideal para la realización de la aplicación de demostración.

10. **Surprise[8]**. Es un SDK para *Python* enfocado en la creación de sistemas de recomendación que utilizan datos explícitos. Este SDK, será de gran utilidad ya que ofrece una gran cantidad de implementaciones optimizadas de algoritmos típicamente utilizados en la elaboración de sistemas de recomendación basados en filtrado colaborativo.

11. **LightFM[12]**. Es una librería para *Python* que ofrece una implementación altamente personalizable de un sistema de recomendación híbrido que combina el filtrado colaborativo con el filtrado basado en características y puede utilizar tanto datos explícitos como implícitos. Al ser una implementación muy optimizada y que ofrece mucha personalizaron esta se adaptará en el motor para así ofrecer mayor flexibilidad con el tipo de datos que los usuario podrán utilizar.

2.5. Calculo de costes

En esta sección se calcularán los costes asociados al desarrollo del proyecto. Este coste se obtendrá a partir de la suma de los costes en personal, la amortización del hardware utilizado

y los costes de las licencias del software empleado en el desarrollo.

2.5.1. Costes humanos

En el desarrollo del proyecto han participado dos personas, el alumno en cuestión y el supervisor de la empresa. El alumno ha dedicado 300h que corresponden a la duración de la estancia mientras que el supervisor ha dedicado alrededor de 12h.

Teniendo en cuenta que a la empresa le cuesta 25€ la hora de trabajo y que el salario del supervisor ronda los 40€/hora, como se puede visualizar en la tabla 2.3, hacen un total de 4.452,00 €, donde se ha añadido un 20 % de coste asociado a la contratación más un 20 % de costes indirectos.

Rango	Horas	Salario/hora	Coste	Costes indirectos	Costes contratación	Total
Desarrollador	300	25,00 €	7.500,00 €	1.500,00 €	1.500,00 €	10.500,00 €
Supervisor	12	40,00 €	480,00 €	96,00 €	96,00 €	672,00 €
						11.172,00 €

Cuadro 2.3: Costes humanos asociados al desarrollo del proyecto

2.5.2. Costes tecnológicos

Los cotes tecnológicos engloban todos los costes asociados tanto al software como a la amortización del hardware utilizado a lo largo del desarrollo del proyecto.

Con relación a los costes asociados al software, solo se han utilizado tecnologías y herramientas gratuitas, por lo que el coste total asociado al software se reduce a 0€.

Respecto al coste del asociado al hardware, se ha utilizando un equipo personal valorado en 1.900€, que engloba el coste de los componentes y periféricos. Se pretende amortizar el equipo a lo largo de 5 años, por lo que el coste de su uso, como se puede ver en la fórmula 2.1, es de 0,19€ la hora. Puesto que se utiliza 8h diarias todos los días laborales (aproximadamente 250 días al año).

$$PH = \frac{\text{coste}}{\text{años} \times \text{días/año} \times \text{horas/día}} = \frac{1.900}{5 \times 250 \times 8} = 0,19 \text{ €/h} \tag{2.1}$$

Esto hace que, como se ve en la ecuación 2.2, el coste relativo al hardware a lo largo de las 300 horas de la estancia en prácticas, ascienda a unos 57€.

$$CH = horas \times \text{coste/h} = 300\text{h} \times 0,19\text{€/h} = 57 \text{ €} \tag{2.2}$$

2.5.3. Coste total

Habiendo calculado tanto los costes tecnológicos como los costes humanos, como se puede ver en la tabla 2.4, el coste total de la realización del proyecto realizado durante las 300h que ha durado la estancia en prácticas asciende a 11.229,00€.

Descripción	Coste
Costes humanos	11.172,00€
Costes tecnológico	57,00 €
	11.229,00€

Cuadro 2.4: Coste total del proyecto

2.6. Seguimiento del proyecto

En este apartado se describirán los detalles de lo acordado en las reuniones de inicio y fin a lo largo de los 6 sprints en los que se ha dividido el proyecto. Además, también se indicaran las historias de usuario y tareas realizadas en cada sprint.

2.6.1. Sprint 1

En la reunión de inicio del sprint se decidió que el principal objetivo de este iba a ser el investigar el estado del arte con respecto a los motores de recomendación. A lo largo de toda la quincena se centró en buscar información sobre motores de recomendación y a la lectura del libro *Practical Recommender Systems by Kim Falk[2]* el cual habla de como implementar y optimizar sistemas de recomendación en Python.

En la reunión de cierre del sprint se mostraron los resultados del aprendizaje y se valoró el uso de la librería *sckit-surprise* para la creación de los distintos algoritmos.

2.6.2. Sprint 2

En la reunión inicial, al tener ya un cierto conocimiento del funcionamiento de los sistemas de recomendación, se decidió que el principal objetivo del sprint iba a ser el implementar varios algoritmos de recomendación.

Las tareas y historias de usuario que se tenía planeado realizar durante el sprint son:

- **HU01**.- Como *product owner* quiero que la API ofrezca algoritmos recomendación por filtrado colaborativo *item-item* para que los sistemas que la integren puedan hacer recomendaciones en función de las valoraciones de los productos consumidos.**13PH**

- **HU02**.- Como *product owner* quiero que la API ofrezca algoritmos de recomendación por filtrado de contenido para que los sistemas que la integren puedan hacer recomendaciones en función de las características de los productos.
- **HU03**.- Como *product owner* quiero que cada cliente solo pueda ver los datos y modelos suyos para evitar problemas de privacidad.**3PH**

En la reunión de cierre del sprint se realizó una demostración del funcionamiento de los algoritmos siendo usados desde la API.

2.6.3. Sprint 3

En la reunión de inicio, se decidió que el principal objetivo sería el implementar las funcionalidades básicas para poder entrenar los distintos modelos de recomendación desde la propia API de manera segura. Además, el *product owner* mostró interés en que todas las operaciones de subida, actualización o eliminación de datos se pudieran realizar desde un único *endpoint*, por lo que esta nueva necesidad se definió como la historia de usuario 14.

Las tareas y historias de usuario que se tenía planeado realizar durante el sprint son:

- **HU07**.- Como encargado de integración de la API quiero poder enviarlos datos requeridos por los usuarios en un JSON para que el algoritmo de recomendación se mantenga actualizado.**5PH**
- **HU09**.- Como encargado de integración de la API quiero poder indicar que se entrene el algoritmo de recomendación con los datos actualizados para que este ofrezca mejores recomendaciones. **2PH**
- **HU04**.- Como *product owner* quiero que para acceder a la API se requiera de una clave de autenticación única asociada a cada cliente para garantizar la seguridad de sus datos. **2PH**
- **HU10**.- Como encargado de integración de la API quiero poder indicar que se elimine un determinado modelo juntamente a los datos almacenados para liberar espacio en memoria. **1PH**
- **HU14**.- Como encargado de integración de la API quiero poder realizar operaciones de subida, actualización y eliminación sobre los datos de un determinado algoritmo en una única operación para poder facilitar la integración de la API con la base de datos de mi aplicación. **2PH**
- Adaptar los algoritmos previamente implementados para que obtengan los datos directamente de la base de datos.

Durante las dos semanas se completaron si problemas todas las tareas asignadas.

En la reunión de cierre, se realizó una demo para mostrar el funcionamiento del mecanismo de subida de datos así como del acceso mediante *API Key*. Tras realizar la demo, el *product*

owner solicitó que la API incluyera la opción de obtener todos los productos valorados por un usuario y todos los usuarios que han valorado un determinado producto, lo que se tradujo en añadir una nueva historia de usuario al *backlog* del proyecto:

- **HU15**.- Como *product owner* quiero que la API ofrezca la posibilidad de ver todas las interacciones producto-usuario y usuario-producto para poder comprender mejor las recomendaciones que se me ofrecen. **3PH**

2.6.4. Sprint 4

En la reunión de inicio del cuarto sprint, se decidió que este se dedicaría a desarrollar una documentación detallada sobre la API. Además, se comentó que sería interesante añadir a la documentación una guía con ejemplos reales de uso juntamente con enlaces a *datasets* para poder hacer pruebas con distintos tipos de datos. También se habló de que sería interesante meter el proyecto dentro de un contenedor de *Docker* para facilitar su futuro despliegue.

Las tareas y historias de usuario que se tenía planeado realizar durante el sprint son:

- **HU05**.- Como encargado de integración de la API quiero que esta me permita acceder a una documentación basada en el estándar *OpenApi* para poder documentarme sobre su uso. **4PH**

- **HU15**.- Como *product owner* quiero que la API ofrezca la posibilidad de ver todas las interacciones producto-usuario y usuario-producto para poder comprender mejor las recomendaciones que se me ofrecen. **3PH**

- **HU16**.- Como encargado de integración de la API quiero disponer de una pequeña guía de uso en la que se explique el uso básico de la API para poder saber su uso básico. **1PH**

- **HU17**.- Como *product owner* quiero que en la documentación de la API se ofrezcan enlaces a distintas bases de datos de ejemplo para poder poner a prueba los distintos algoritmos y ver las recomendaciones que ofrecen. **1PH**

- **HU18**.- Como *product owner* quiero que la API se despliegue mediante contenedores de *Docker* para poder acceder a sus funcionalidades desde internet . **5PH**

- Aprendizaje de uso de *Docker*.

En este sprint se dispuso de dos días menos de lo habitual por haber coincidido en días festivos, por lo que no dio tiempo a terminar la *HU18* y la tarea de aprender a utilizar *Docker*. Además, no se pudo realizar la reunión de cierre del sprint debido a que el supervisor no estaba disponible.

2.6.5. Sprint 5

En la reunión de inicio, se comentaron los resultados del cierre del cuarto sprint al supervisor, ya que no pudo asistir a la reunión y se decidió que el objetivo de este sería el realizar pruebas de integración sobre la API y la implementación de un *endpoint* para poder ver la precisión de los distintos modelos entrenados.

Las tareas y historias de usuario que se tenían planeadas realizar durante el sprint son:

- **HU18**.- Como *product owner* quiero que la API se despliegue mediante contenedores de *Docker* para poder acceder a sus funcionalidades desde internet . **5PH**
- **HU11**.- Como usuario de una de las aplicaciones que implementan la API quiero recibir recomendaciones que me sean relevantes para mejorar mi experiencia de uso. **3PH**
- **HU12**.- Como encargado de integración de la API quiero poder acceder a informes en los que se indiquen estadísticas del nivel de precisión del modelo entrenado para poder comprobar si el algoritmo rinde correctamente.**5PH**
- Implementación de pruebas de integración.
- Aprendizaje de uso de *Docker*.

En la reunión de cierre del sprint se indicó que se habían realizado todas las tareas e historias previstas y se hizo una demostración del trabajo. Tras la demostración el *product owner* dijo que se descartaría la *HU13* , ya que tenia previsto integrarla en un futuro con el software de gestión empresarial *PowerBI*.

2.6.6. Sprint 6

En la reunión de inicio del sprint se decidió que el objetivo de este sería la realización de una revisión a fondo de su documentación y funcionalidad con la finalidad de pulir el producto para su posterior puesta en producción, por lo que se decidió que sería necesario el despliegue del proyecto en los servidores de la empresa.

Las tareas y historias de usuario que se tenía planeado realizar durante el sprint son:

- Desplegar el proyecto mediante *Docker*.
- Revisar funcionalidad y aplicar mejoras.

En la reunión de cierre del sprint el propio *product owner* realizó una prueba de uso de la API en su propio ordenador y concluyó que el producto estaba completo. Como sobraba tiempo, indicó que sería interesante la realización de una pequeña aplicación móvil en la que se pudieran visualizar parte de las funcionalidades del motor de recomendación, para así poder enseñarlo a los clientes a modo de demostración.

2.6.7. Sprint 7

En la reunión de inicio del sprint se decidió que el objetivo de este sería la realización de mejoras a la API REST. En la reunión se habló entre otras cosas de los resultados de las pruebas realizadas durante el anterior sprint y se propusieron mejoras y funcionalidades extra. Entre las más relevantes, se habló de reorganizar la estructura de los *endpoint* para hacerlos más intuitivos y se planteó mejorar el proceso de subida de datos para que se ejecuten en segundo plano.

Las tareas e historias de usuario que se tenían planeadas realizar durante el sprint son:

- Cambiar organización *endpoints.*
- Mejorar el proceso de operaciones de datos haciéndolo en segundo plano y monitorizarlo con un *token* desde un *endpoint* nuevo.
- Realizar aplicación de demostración de uso del motor de recomendación.

En la reunión de cierre del sprint se mostraron los nuevos *endpoint* reorganizados, la nueva forma de realizar operaciones sobre datos y la aplicación en funcionamiento. Tras las demostraciones, se comentó que el estado del proyecto era lo bastante bueno como para ponerse en producción, por lo tanto, se decidió que se le daría acceso a Easygoband para que lo pruebe y proponga mejoras.

Capítulo 3

Análisis y diseño del sistema

3.1. Análisis del sistema

En el análisis del sistema se pretende la elaboración de una descripción detallada de las funcionalidades que este deberá de ofrecer, ajustándose a las necesidades del cliente.

3.1.1. Definición de requisitos

En este apartado se definirán los requisitos del producto. Para obtenerlos se utilizará un diagrama de casos de uso, ya que este detalla de manera esquemática todas las funcionalidades e interacciones que tendrá el sistema a desarrollar.

Como se puede ver en la figura 3.1, se han identificado cuatro actores distintos y 10 casos de uso.

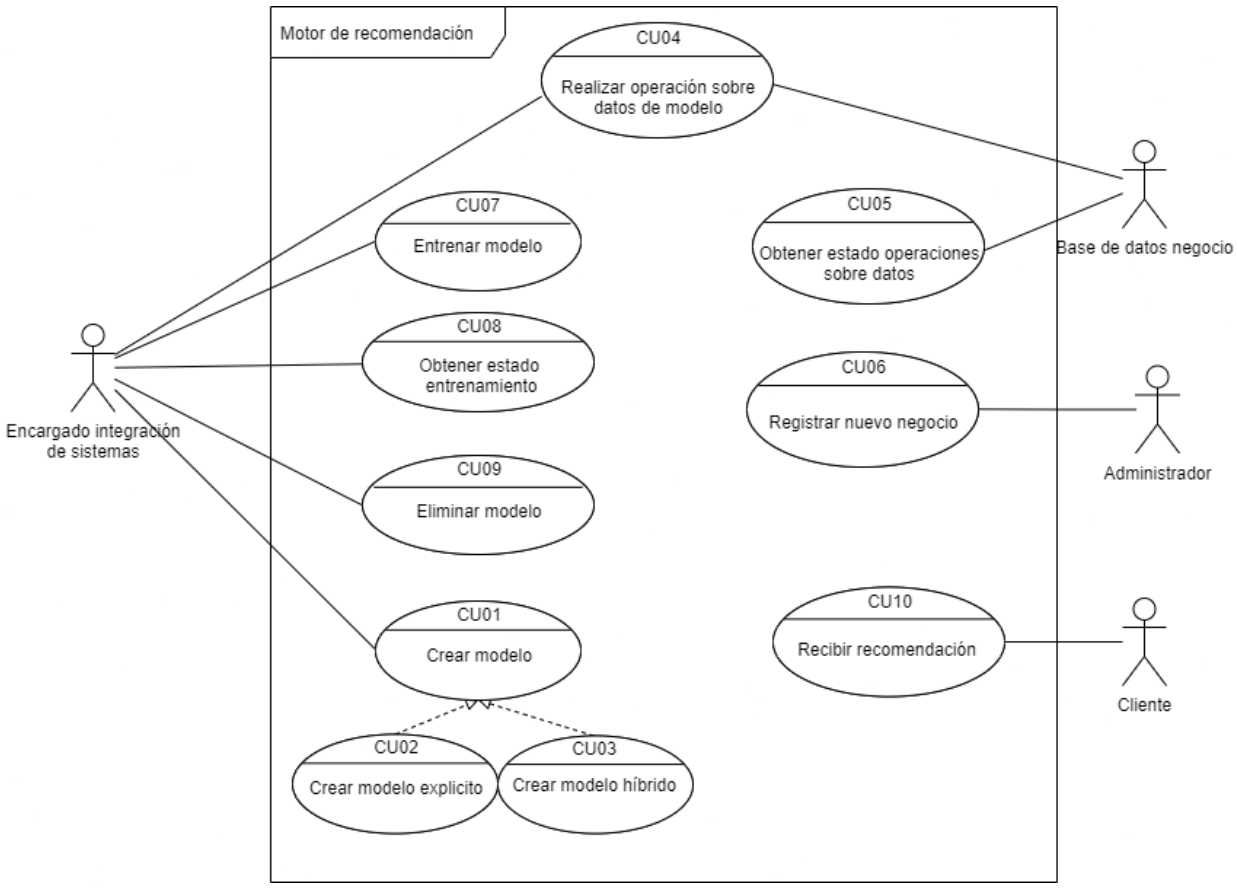

Figura 3.1: Diagrama de casos de uso de motor de recomendación.

Especificación actores

En cuanto a los distintos actores presentes en el sistema, se definen como:

- **Encargado de integración de sistemas**: Es el empleado de la empresa encargado de todas las tareas relativas a la integración del motor de recomendación en otros productos. También es el encargado de realizar las acciones de mantenimiento de los modelos de recomendación.

- **Base de datos negocio**: Es cualquier sistema de almacenamiento externo perteneciente al negocio que integra el motor de recomendación. Este tiene la responsabilidad de mantener actualizados los datos de motor de recomendación.

- **Administrador**: Es el responsable de gestionar las claves de acceso a la API REST del motor de recomendación.

- **Cliente**: Es el interesado en recibir recomendaciones en base a sus gustos.

Especificación casos de uso

A continuación se muestran las tablas (3.1-3.10) en las que se especifican con mayor detalle todos los casos de uso presentes en la figura 3.1.

Especificación CU01	
Identificador	CU01
Nombre	Crear modelo
Versión	1.0
Autores	Jaume Barrios Forner
Fuentes	Sergio Aguado
Descripción	Se debe de permitir crear modelos de recomendación a partir de un algoritmo de recomendación
Alcance	El sistema debe de permitir crear modelos a partir de uno de los algoritmos disponibles y asignarle un nombre a este. En caso de ya existir no debe permitir que un mismo usuario tenga modelos con el mismo nombre.
Nivel	Tarea principal
Actor principal	Encargado de integración de sistemas
Actores secundarios	N/A
Relaciones	DR01: Encargado de integración de sistemas
Precondición	El encargado debe de tener una clave de identificación
Condición fin éxito	Se crea un modelo correctamente
Condición fin fracaso	No se crea el modelo y se indica el motivo.
Trigger	El encargado va a integrar la API en un nuevo sistema.
Secuencia normal	**Acción**
1	El encargado construye una petición a la API con todos los parámetros requeridos
2	El encargado manda la petición al sistema.
3	El sistema valida los datos.
4	El sistema crea el nuevo modelo.
5	El sistema almacena el modelo en la base de datos.
6	El sistema responde con un mensaje con código 200 indicando que se ha realizado la operación.
Excepción 1	**Excepción nombre en uso**
4	El sistema detecta que el nombre asignado al modelo está en uso
5	El sistema responde con un mensaje con código 400 e indica que el nombre está en uso.
Frecuencia esperada	Cada vez que se integre el motor de recomendación en un nuevo negocio.
Importancia	Necesario
Prioridad	Termino corto
Comentarios	N/A

Cuadro 3.1: Especificación CU01 - Crear Modelo.

Especificación CU02	
Identificador	CU02
Nombre	Crear modelo explícito
Versión	1.0
Autores	Jaume Barrios Forner
Fuentes	Sergio Aguado
Descripción	Se debe de permitir crear modelos de recomendación a partir de datos explícitos.
Alcance	El sistema debe de permitir crear modelos a partir de un algoritmo de filtrado colaborativo de tipo knn (vecino más próximo) para que a partir de valoraciones explícitas genere recomendaciones.
Nivel	Tarea principal
Actor principal	Encargado de integración de sistemas
Actores secundarios	N/A
Relaciones	DR01: Encargado de integración de sistemas, CU01: Crear Modelo
Precondición	El encargado debe de tener una clave de identificación
Condición fin éxito	Se crea un modelo correctamente
Condición fin fracaso	No se crea el modelo y se indica el motivo.
Trigger	El encargado va a integrar la API en un nuevo sistema.
Secuencia normal	**Acción**
1	El encargado construye una petición a la API con todos los parámetros requeridos.
2	El encargado manda la petición al sistema.
3	El sistema valida los datos.
4	El sistema crea el nuevo modelo a partir de la configuración indicada.
5	El sistema almacena el modelo en la base de datos.
6	El sistema responde con un mensaje con código 200 indicando que se ha realizado la operación.
Excepción 1	**Excepción número de vecinos fuera de rango**
4	El sistema detecta que el mínimo número de vecinos indicado es superior al máximo
5	El sistema responde con un mensaje con código 400 e indica que el rango es incorrecto.
Frecuencia esperada	Cada vez que se integre el motor de recomendación en un nuevo negocio.
Importancia	Necesario
Prioridad	Termino corto
Comentarios	N/A

Cuadro 3.2: Especificación CU02 - Crear modelo explícito.

Especificación CU03	
Identificador	CU03
Nombre	Crear modelo híbrido
Versión	1.0
Autores	Jaume Barrios Forner
Fuentes	Sergio Aguado
Descripción	Se debe de permitir crear modelos de recomendación a partir de un algoritmo de recomendación
Alcance	El sistema debe de permitir crear modelos a partir de algoritmos híbridos que permitan generar recomendaciones a partir de valoraciones de los usuarios y de las características tanto de otros clientes como de los productos que les gustan.
Nivel	Tarea principal
Actor principal	Encargado de integración de sistemas
Actores secundarios	N/A
Relaciones	DR01: Encargado de integración de sistemas,CU01: Crear Modelo
Precondición	El encargado debe de tener una clave de identificación
Condición fin éxito	Se crea un modelo correctamente
Condición fin fracaso	No se crea el modelo y se indica el motivo.
Trigger	El encargado va a integrar la API en un nuevo sistema.
Secuencia normal	**Acción**
1	El encargado construye una petición a la API con todos los parámetros requeridos
2	El encargado manda la petición al sistema.
3	El sistema valida los datos.
4	El sistema crea el nuevo modelo.
5	El sistema almacena el modelo en la base de datos.
6	El sistema responde con un mensaje con código 200 indicando que se ha realizado la operación.
Excepción 1	**Excepción valores incorrectos**
4	El sistema detecta el nombre del algoritmo de optimización no existe entre los que se ofrecen.
5	El sistema responde con un mensaje con código 422 e indica que un parámetro no es válido.
Frecuencia esperada	Cada vez que se integre el motor de recomendación en un nuevo negocio.
Importancia	Necesario
Prioridad	Término Medio
Comentarios	N/A

Cuadro 3.3: Especificación CU03 - Crear modelo híbrido.

Especificación CU04	
Identificador	CU04
Nombre	Realizar operación sobre datos de modelo
Versión	1.0
Autores	Jaume Barrios Forner
Fuentes	Sergio Aguado
Descripción	El sistema debe de permitir realizar operaciones de eliminación, inserción y actualización sobre los datos de un modelo.
Alcance	El sistema debe de permitir realizar operaciones de eliminación, inserción y actualización sobre los datos relativos a clientes, productos e interacciones cliente/producto siempre que estos no se repitan.
Nivel	Tarea principal
Actor principal	Encargado de integración de sistemas
Actores secundarios	Base de datos negocio
Relaciones	DR01: Encargado de integración de sistemas DR02: Base de datos negocio
Precondición	El encargado de integración debe de haber configurado correctamente la base de datos del negocio.
Condición fin éxito	Se realiza la/las operación/es sobre los datos asociados a un modelo de recomendación
Condición fin fracaso	Se cancela la operación y se descartan los cambios.
Trigger	El encargado va a integrar la API en un nuevo sistema o el sistema que tiene integrada la API quiere actualizar los datos.
Secuencia normal	**Acción**
1	Se manda una petición HTTP a la API del motor de recomendación con los datos a cambiar
2	El sistema recibe la petición
3	El sistema valida los datos.
4	Retorna un token que identifica la operación.
5	El sistema empieza a realizar las operaciones.
6	El sistema actualiza la información del estado de las operaciones asociado al token en la base de datos.
Excepción 1	**Excepción modelo no existente**
4	El sistema detecta el nombre del modelo de recomendación no existe.
5	El sistema responde con un mensaje con código 404 e indica que no se ha encontrado el modelo.
Frecuencia esperada	Diaria.
Importancia	Necesario
Prioridad	Termino Corto
Comentarios	N/A

Cuadro 3.4: Especificación CU04 - Realizar operación sobre datos de modelo.

Especificación CU05	
Identificador	CU05
Nombre	Obtener estado operaciones sobre datos
Versión	1.0
Autores	Jaume Barrios Forner
Fuentes	Sergio Aguado
Descripción	El sistema debe permitir consultar el estado de las operaciones que se han realizado sobre los datos.
Alcance	El sistema debe de permitir ver el estado de la operación sobre los datos asociados a un modelo. Este estado puede ser de en proceso, finalizado, en cola y error y además debe de indicar cuántas operaciones ha hecho y cuántas faltan.
Nivel	Tarea principal
Actor principal	Base de datos negocio
Actores secundarios	N/A
Relaciones	DR02: Base de datos negocio
Precondición	Se debe de disponer de un token de operación generado por la operación de operación sobre datos asociados a un modelo.
Condición fin éxito	Se muestra el estado de la operación
Condición fin fracaso	Se produce una excepción.
Trigger	Se ha iniciado una operación sobre los datos asociados a un modelo de recomendación.
Secuencia normal	**Acción**
1	Se manda una petición HTTP al *endpoint* de estado de operación con un token alfanumérico.
2	El sistema valida los datos.
3	El sistema consulta en la base de datos el estado.
5	El sistema retorna un mensaje que contiene el estado de la operación
Excepción 1	**Excepción token no existente**
4	La base de datos no retorna ningún dato asociado al token.
5	El sistema responde con un mensaje con código 404 e indica que no se ha encontrado el token.
Frecuencia esperada	Cada minuto tras empezar una operación.
Importancia	Media
Prioridad	Término Medio.
Importancia	Necesario
Prioridad	Termino Corto
Comentarios	N/A

Cuadro 3.5: Especificación CU05 - Obtener estado operaciones sobre datos.

Especificación CU06	
Identificador	CU06
Nombre	Registrar nuevo negocio
Versión	1.0
Autores	Jaume Barrios Forner
Fuentes	Sergio Aguado
Descripción	El sistema debe permitir generar *API Key* únicas para identificar los distintos negocios que implementan el sistema.
Alcance	Únicamente debe de generar un token encriptado que se asocie a todos los datos de un negocio para separar los datos de los clientes y añadir seguridad.
Nivel	Tarea secundaria
Actor principal	Administrador
Actores secundarios	N/A
Relaciones	DR03: Administrador
Precondición	El negocio no debe de estar registrado en el sistema
Condición fin éxito	Se genera una *API key* única.
Condición fin fracaso	Se produce una excepción.
Trigger	Un nuevo cliente quiere integrar el motor de recomendación en su negocio.
Secuencia normal	**Acción**
1	El administrador ejecuta el *script* de creación de claves.
2	El administrador introduce los datos del negocio.
3	El sistema valida los datos.
5	El sistema genera una *API key* nueva.
6	El sistema retorna la *API key* por consola.
Excepción 1	**Excepción negocio existente**
4	El sistema detecta que el nombre de negocio está en uso.
5	El sistema muestra por consola un mensaje indicando que el nombre está en uso.
Frecuencia esperada	Baja.
Importancia	Baja.
Prioridad	Término Largo.
Comentarios	N/A
Comentarios	N/A

Cuadro 3.6: Especificación CU06 - Registrar nuevo negocio.

Especificación CU07	
Identificador	CU07
Nombre	Entrenar modelo
Versión	1.0
Autores	Jaume Barrios Forner
Fuentes	Sergio Aguado
Descripción	El sistema debe de permitir entrenar los modelos para que "aprendan" a realizar recomendaciones en base a los datos que se le han ofrecido.
Alcance	Este caso de uso se limita a iniciar el proceso de entrenamiento y a generar un token para comprobar el estado del proceso.
Nivel	Tarea principal
Actor principal	Encargado de integración de sistemas
Actores secundarios	N/A
Relaciones	DR01: Encargado de integración de sistemas
Precondición	El modelo a entrenar debe de existir y debe de tener datos asociados.
Condición fin éxito	Se genera un token de control de proceso de entrenamiento.
Condición fin fracaso	Se produce una excepción.
Trigger	El encargado de integración quiere entrenar el motor de recomendación de un negocio.
Secuencia normal	**Acción**
1	El encargado de integración manda una petición HTTP a la API donde indica el modelo a entrenar y la configuración del entrenamiento
2	El sistema valida si el modelo es apto para ser entrenado y la configuración.
3	El sistema genera un token de seguimiento de entrenamiento.
4	El sistema encola el modelo en la cola de entrenamiento de modelos.
5	El sistema retorna un mensaje HTTP con el token.
Excepción 1	**Excepción modelo no existe**
4	El sistema detecta que el nombre del modelo no existe en la base de datos.
5	El sistema retorna un mensaje HTTP con código 404 indicando que no existe el modelo solicitado.
Frecuencia esperada	Semanal.
Importancia	Alta..
Prioridad	Término corto.
Comentarios	N/A
Comentarios	N/A

Cuadro 3.7: Especificación CU07 - Entrenar modelo.

Especificación CU08	
Identificador	CU08
Nombre	Obtener estado entrenamiento
Versión	1.0
Autores	Jaume Barrios Forner
Fuentes	Sergio Aguado
Descripción	El sistema debe de permitir controlar el proceso de entrenamiento de un modelo.
Alcance	Debe de mostrar el estado del entrenamiento de un modelo y si ha finalizado de manera satisfactoria el entrenamiento, debe de mostrar las puntuaciones de precisión alcanzada por el modelo tras el entrenamiento.
Nivel	Tarea principal
Actor principal	Encargado de integración de sistemas
Actores secundarios	N/A
Relaciones	DR01: Encargado de integración de sistemas
Precondición	Se debe de disponer de un token de entrenamiento asociado a uno de los modelos en propiedad del negocio.
Condición fin éxito	Se retorna el estado del entrenamiento.
Condición fin fracaso	Se produce una excepción.
Trigger	El encargado de integración quiere saber cómo va el proceso de entrenamiento de un modelo.
Secuencia normal	**Acción**
1	El encargado de integración manda una petición HTTP al *endpoint* de control de entrenamiento.
2	El sistema comprueba el estado del proceso asociado al token.
3	El sistema retorna un mensaje HTTP con el estado del proceso de entrenamiento.
Excepción 1	**Excepción token no existe**
3	El sistema no encuentra ningún proceso de entrenamiento asociado al token.
5	El sistema retorna un mensaje HTTP con código 404 indicando que no existe el token.
Frecuencia esperada	Cada minuto tras empezar un proceso de entrenamiento.
Importancia	Alta.
Prioridad	Término corto.
Comentarios	N/A
Prioridad	Término corto.
Comentarios	N/A
Comentarios	N/A

Cuadro 3.8: Especificación CU08 - Obtener estado entrenamiento.

Especificación CU09	
Identificador	CU09
Nombre	Eliminar modelo
Versión	1.0
Autores	Jaume Barrios Forner
Fuentes	Sergio Aguado
Descripción	El sistema debe permitir eliminar los modelos para así liberar espacio de almacenamiento.
Alcance	Al eliminar un modelo se deberá de eliminar el propio modelo como todos los datos asociados a este.
Nivel	Tarea secundaria.
Actor principal	Encargado de integración de sistemas
Actores secundarios	N/A
Relaciones	DR01: Encargado de integración de sistemas
Precondición	Se debe de disponer de un modelo existente (no es necesario ni que tenga datos asociados ni que esté entrenado).
Condición fin éxito	Se elimina el modelo de la base de datos.
Condición fin fracaso	Se produce una excepción.
Trigger	El encargado de integración quiere eliminar un modelo del sistema.
Secuencia normal	**Acción**
1	El encargado de integración manda una petición HTTP al *endpoint* de eliminación de modelos indicando el nombre de este.
2	El sistema comprueba que el modelo existe.
3	El sistema elimina todos los datos asociados al modelo.
4	El sistema retorna un mensaje HTTP con código 200 e indica que se ha realizado la operación.
Excepción 1	**Excepción modelo no existe**
3	El sistema no encuentra ningún modelo con el nombre solicitado.
5	El sistema retorna un mensaje HTTP con código 404 indicando que no existe el modelo.
Frecuencia esperada	Anual
Importancia	Baja.
Prioridad	Término largo.
Comentarios	N/A
Comentarios	N/A
Comentarios	N/A

Cuadro 3.9: Especificación CU09 - Eliminar modelo.

Especificación CU10	
Identificador	CU10
Nombre	Recibir recomendación
Versión	1.0
Autores	Jaume Barrios Forner
Fuentes	Sergio Aguado
Descripción	El sistema debe ofrecer a los clientes de los negocios recomendaciones personalizadas.
Alcance	A partir de un modelo entrenado se deben de generar un determinado número de recomendaciones que dependiendo de la configuración deben de incluir productos ya valorados.
Nivel	Tarea principal.
Actor principal	Cliente
Actores secundarios	N/A
Relaciones	DR04: Cliente
Precondición	Se debe de disponer de un modelo existente entrenado y de una API key.
Condición fin éxito	Se generan recomendaciones.
Condición fin fracaso	Se produce una excepción.
Trigger	Un cliente accede al apartado de recomendaciones de una de las aplicaciones que integran el sistema de recomendación.
Secuencia normal	**Acción**
1	El cliente accede a un apartado donde se deben de mostrar recomendaciones.
2	La aplicación manda una solicitud a la API para obtener recomendaciones.
3	El sistema de recomendaciones valida los datos de la solicitud.
4	El sistema genera las recomendaciones mediante el modelo indicado.
5	El sistema retorna las recomendaciones generadas por el modelo.
6	La aplicación muestra los productos recomendados al cliente.
Excepción 1	**Excepción key no válida.**
3	El sistema detecta que la clave de acceso no es válida.
5	El sistema retorna un mensaje HTTP indicando que el acceso no está permitido.
Frecuencia esperada	Cada minuto dependiendo de la cantidad de clientes.
Importancia	Alta.
Prioridad	Término corto.
Comentarios	N/A

Cuadro 3.10: Especificación CU10 - Recibir recomendación.

Requisitos de datos

A continuación se muestran las tablas (3.11-3.14) en las que se identifican los distintos requisitos de datos identificados a partir de los casos de uso.

Requisito de datos RD01	
Identificador	RD01
Nombre	Negocio
Versión	1.0
Autores	Jaume Barrios Forner
Fuentes	Jaume Barrios Forner
Requisitos asociados	-
Datos específicos	Nombre, código secreto
Ocurrencias	2
Importancia	Media
Comentarios	Inicialmente solo habrán dos instancias paraa Easygoband y Cuatroochenta.

Cuadro 3.11: Requisito de datos RD01 - Negocio.

Requisito de datos RD02	
Identificador	RD02
Nombre	Cliente
Versión	1.0
Autores	Jaume Barrios Forner
Fuentes	Jaume Barrios Forner
Requisitos asociados	RD01
Datos específicos	Id, Nombre, listado de etiquetas
Ocurrencias	Tantas como el número de clientes del negocio
Importancia	Alta
Comentarios	El listado de etiquetas sirve para indicar las características distintivas del cliente

Cuadro 3.12: Requisito de datos RD02 - Cliente.

Requisito de datos RD03	
Identificador	RD03
Nombre	Producto
Versión	1.0
Autores	Jaume Barrios Forner
Fuentes	Jaume Barrios Forner
Requisitos asociados	RD01
Datos específicos	Id, Nombre, listado de etiquetas
Ocurrencias	Tantas como el número de productos del negocio
Importancia	Alta
Comentarios	Las etiquetas sirven para diferenciar el producto

Cuadro 3.13: Requisito de datos RD03 - Producto.

Requisito de datos RD04	
Identificador	RD04
Nombre	Interacción
Versión	1.0
Autores	Jaume Barrios Forner
Fuentes	Jaume Barrios Forner
Requisitos asociados	RD02, RD03
Datos específicos	id, peso de la interacción, id del usuario y id del producto
Ocurrencias	El número de ocurrencias suele ser de al menos la cantidad de clientes del negocio
Importancia	Alta
Comentarios	-

Cuadro 3.14: Requisito de datos RD04 - Interacción.

3.1.2. Análisis de requisitos

Con los requisitos identificados mediante el diagrama de casos de uso presente en la figura 3.1, en esta sección se procederá al análisis en detalle de estos.

Para la realización del análisis se pueden utilizar diagramas UML (*Unified Modeling Languaje*) de distintos tipos como diagramas de clases del análisis, diagramas de flujo o diagramas de actividades entre otros. Para este proyecto se utilizarán diagramas de actividades ya que estos muestran de manera clara y visual las actividades que se deberán de hacer para poder ofrecer las funcionalidades descritas en el diagrama de casos de uso.

Creación de modelo

Como se puede ver en la figura 3.2, en esta se muestran las actividades requeridas para ofrecer las funcionalidades presentes en los casos de uso CU01, CU02 y CU03 las cuales hacen referencia a la creación de modelos de recomendación. En cuanto a la secuencia de actividades, primeramente, antes de crear un modelo, se validan para que todos los datos sean correctos para luego crear el modelo y almacenarlo en la base de datos.

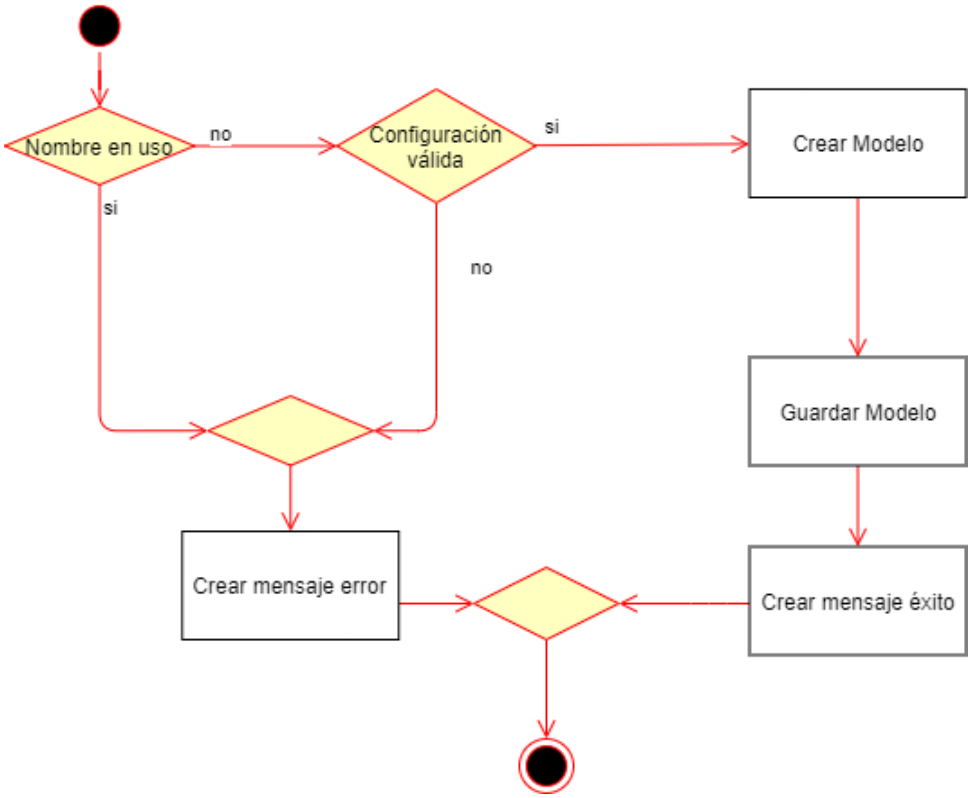

Figura 3.2: Diagrama de actividades creación modelo de recomendación.

Borrar modelo

Como se puede ver en la figura 3.3, en esta se muestran las actividades requeridas para ofrecer la funcionalidad presente en el caso CU09 (eliminar modelo). Cabe destacar que dentro del proceso de eliminación del modelo también se incluyen la eliminación de todos los datos de entrenamiento asociados al modelo.

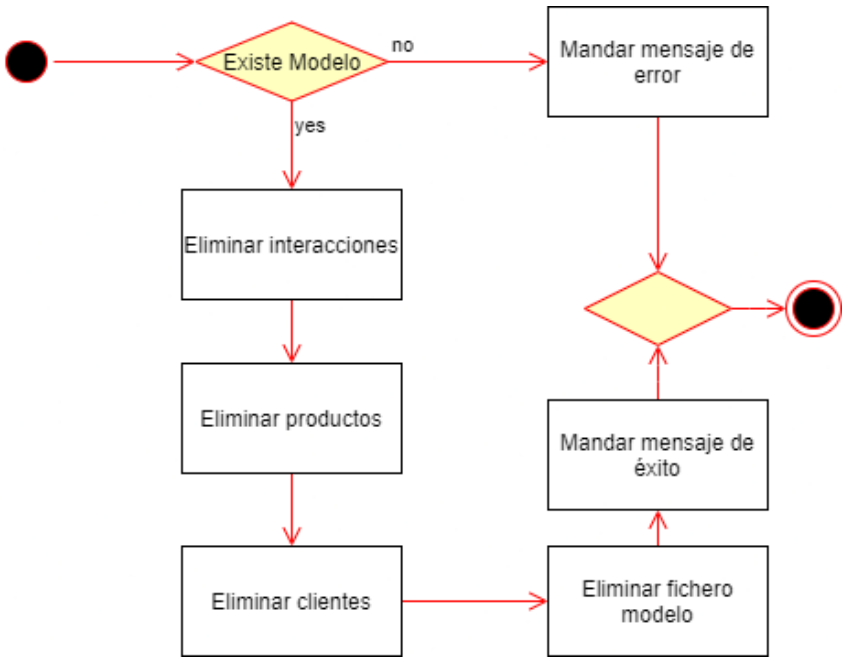

Figura 3.3: Diagrama de actividades eliminación modelo de recomendación.

Generar recomendación

Para ofrecer la funcionalidad presente en el CU10 (recibir recomendación) se ha elaborado el diagrama de actividades presente en la figura 3.4 en el que se identifica el cómo se deberían de generar recomendaciones personalizadas a partir de los datos del usuario y cual debería de ser la respuesta en el caso de que no exista el usuario en los datos de entrenamiento del modelo.

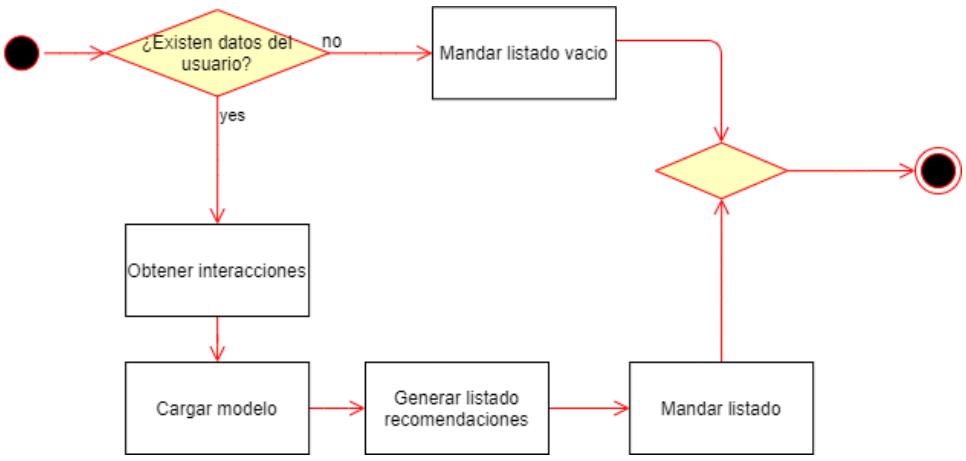

Figura 3.4: Diagrama de actividades generación de recomendaciones.

Operación sobre datos de modelo

Para ofrecer la funcionalidad presente en el CU04 (realizar operación sobre datos de modelo) se ha elaborado el diagrama de actividades presente en la figura 3.5 en el que se identifica únicamente el proceso donde se inicializa la carga de datos que está asociada a un token de seguimiento, el cual es mandado dentro de la respuesta.

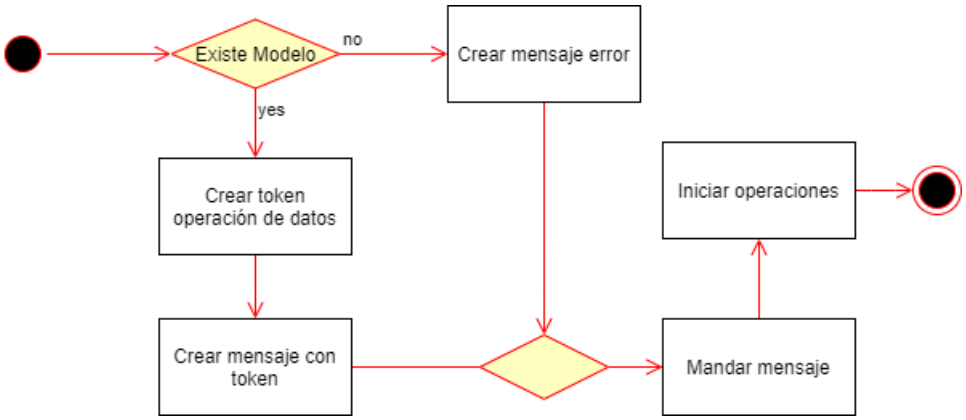

Figura 3.5: Diagrama de actividades operación sobre datos asociados a modelo.

Entrenar modelo

Para ofrecer la funcionalidad presente en el CU07 (entrenar modelo), se ha elaborado el diagrama de actividades presente en la figura 3.6 en el que se describe la secuencia de creación de un proceso de entrenamiento en paralelo asociado a un token de seguimiento del estado de este.

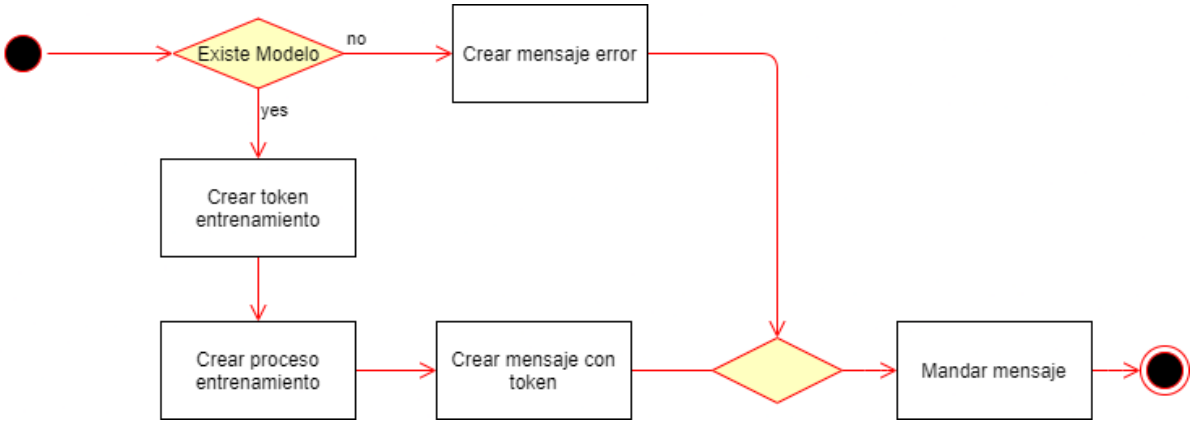

Figura 3.6: Diagrama de actividades inicialización proceso de entrenamiento.

Registrar negocio

Para ofrecer la funcionalidad presente en el CU06 (registrar nuevo negocio), se ha elaborado el diagrama de actividades presente en la figura 3.7 en el que se describe el proceso de creación de un nuevo negocio en el sistema juntamente a su clave de acceso a la API.

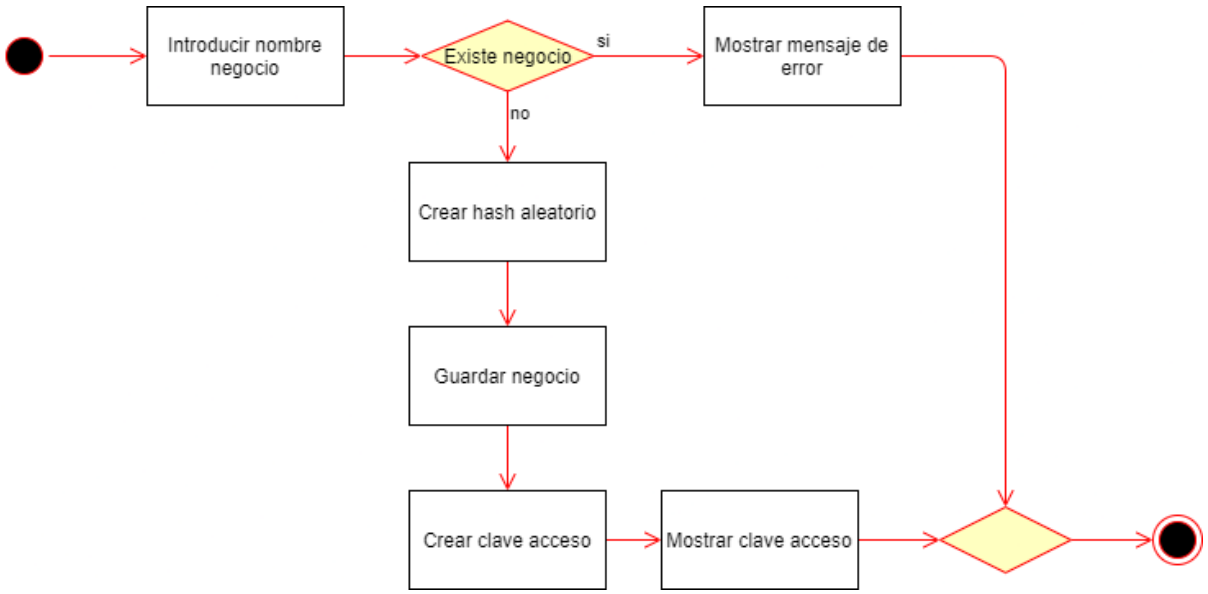

Figura 3.7: Diagrama de actividades registro de nuevo negocio.

3.2. Diseño de la arquitectura del sistema

En esta sección se mostrarán el diseño de la arquitectura del sistema a distintos niveles de abstracción.

3.2.1. Diseño a nivel de sistema

Por motivos de rendimiento ha surgido la necesidad de distribuir el proceso de entrenamiento entre diversos subsistemas, ya que este proceso es bastante costoso en términos de carga de procesamiento y no es viable su ejecución en *threads* por limitaciones del intérprete de *Python*.

Como se puede observar en la figura 3.8, el sistema está dividido en tres subsistemas con tareas específicas de los que se puede identificar:

- **API** Es la parte encargada de comunicarse con el exterior ofreciendo una abstracción de todas las funcionalidades mediante una API REST.
- **DB** Es el sistema de almacenamiento de datos, en concreto es una instancia de *MongoDB*.

- **Cola de entrenamiento** Es el sistema encargado de gestionar y distribuir las tareas de entrenamiento entre los *workers* disponibles.
- **Workers** Son los subsistemas encargados de entrenar los modelos de recomendación, reciben las tareas de la cola de entrenamiento.
- **CLI** Subsistema que permite acceder al administrador a una interfaz por línea de comandos para gestionar la generación de claves de acceso al sistema.

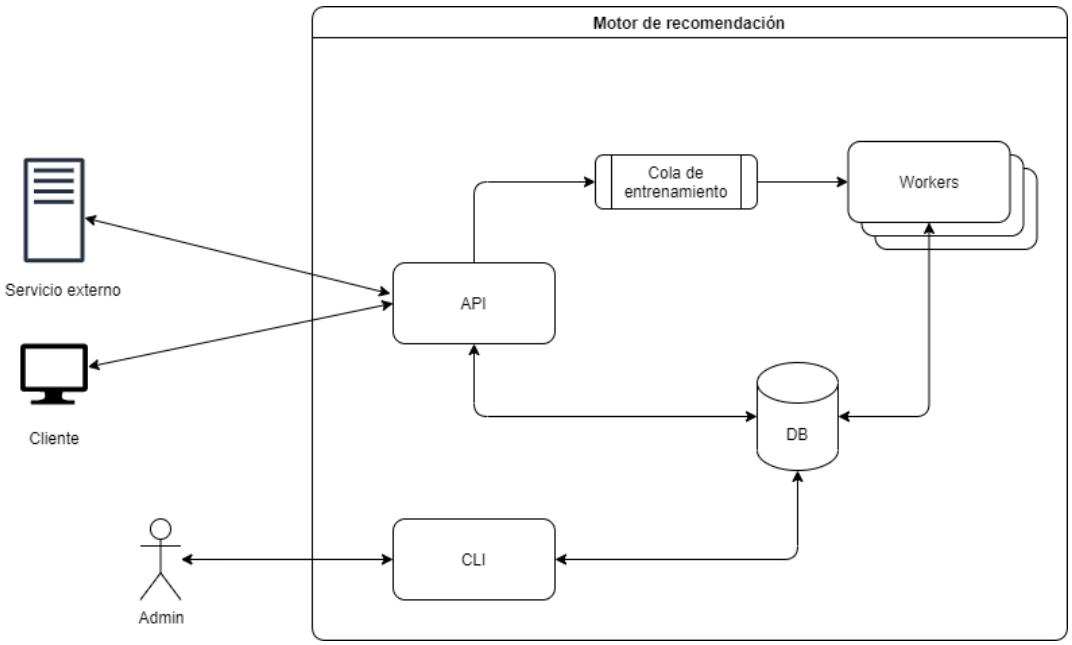

Figura 3.8: Diagrama de diseño del sistema.

3.2.2. Diagrama de clases del diseño

El diagrama de clases del diseño sirve para visualizar cómo se estructurarán las distintas clases y cuales van a ser los datos que van a requerir. Es muy importante la correcta realización de este diagrama puesto que este será la base que se utilizará para crear todas las clases del sistema y las distintas colecciones de la base de datos.

En la figura 3.9 se puede ver el diagrama de clases del diseño final del motor de recomendación.

3.3. Diseño de la interfaz

Este proyecto inicialmente no tenía planteado incluir ninguna interfaz gráfica. Durante los últimos sprints surgió la idea de realizar una pequeña aplicación móvil escrita en *Flutter* para mostrar el motor de recomendación en funcionamiento de manera visual a los clientes.

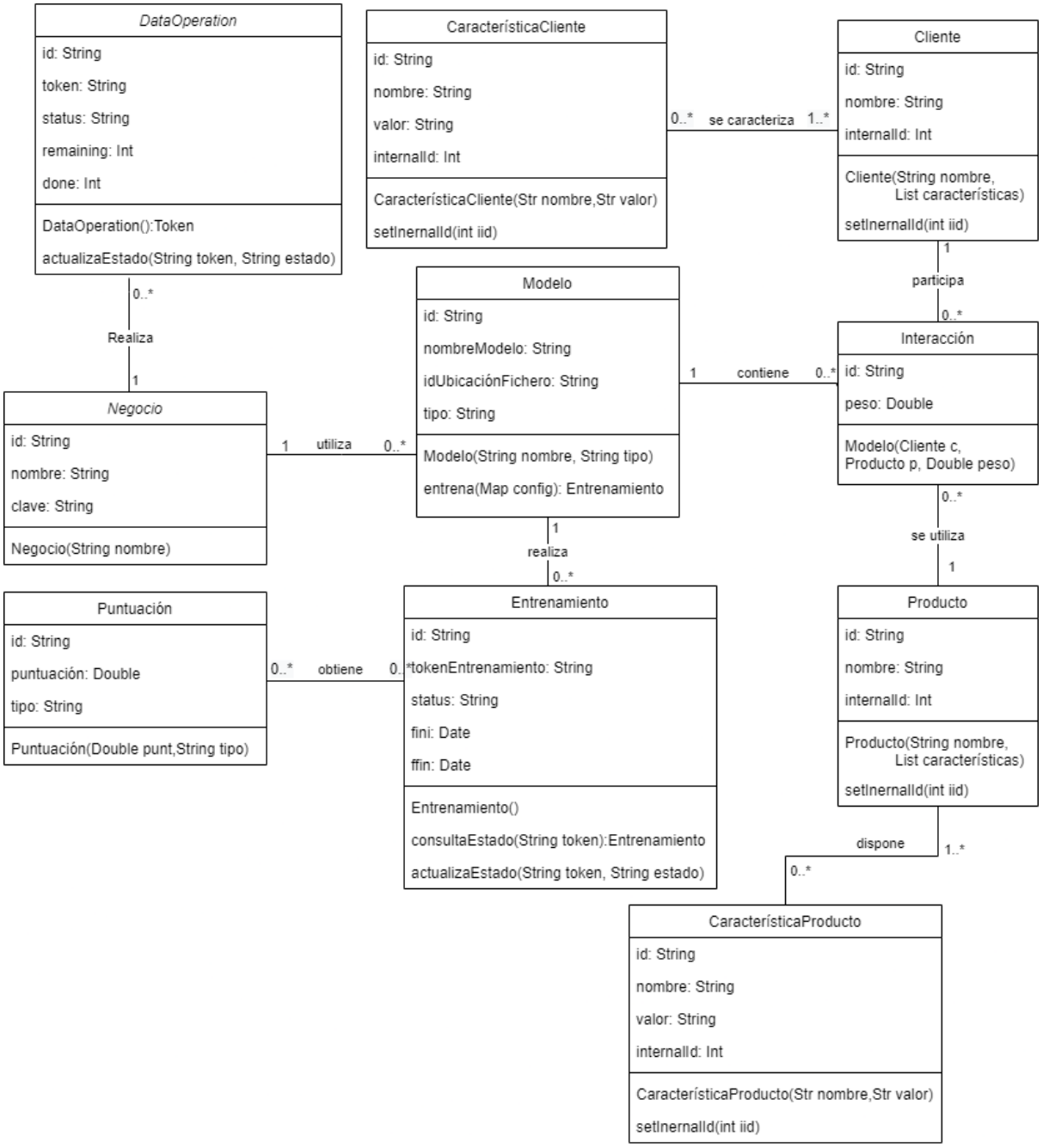

Figura 3.9: Diagrama de clases del diseño del motor de recomendación.

En concreto, se decidió que la aplicación de ejemplo mostrase cómo se podía integrar el sistema en un entorno real para recomendar películas. Puesto que es mucho más sencillo para la mayoría de personas el evaluar la calidad de las recomendaciones.

Por consiguiente, la interfaz de la aplicación se ha diseñado con la idea de visibilizar los resultados de las recomendaciones de manera directa y de permitir compararlos de manera sencilla.

3.3.1. Sitemap de la aplicación

Con la idea de mostrar la mayoría de las funcionalidades del sistema, se han distribuido las vistas de la aplicación como se muestra en el *sitemap* presente en la Figura 3.10.

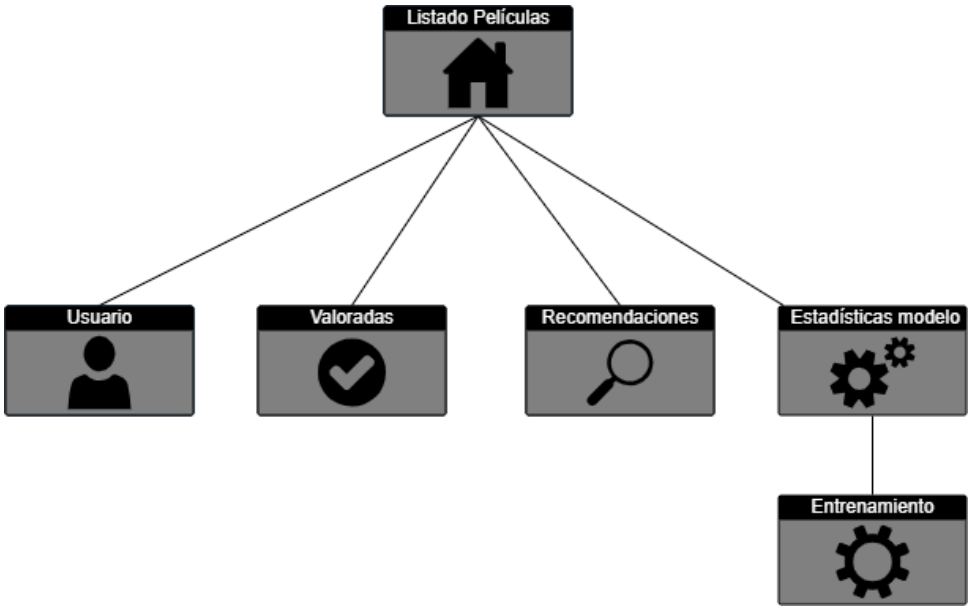

Figura 3.10: *Sitemap* de la aplicación de demostración.

Las distintas vistas de la aplicación mostradas en la figura 3.10 sirven para:

- **Listado de películas**: Vista en la que se listan todas las películas no valoradas por el usuario.
- **Usuario**: Vista en la que se puede cambiar el usuario activo.
- **Valoradas**: Vista en la que se listan todas las películas valoradas por el usuario.
- **Recomendaciones**: Vista en la que se listan las películas recomendadas por el motor de recomendación.
- **Estadísticas modelo**: Vista en la que se muestran todas las estadísticas relativas al modelo que se está utilizando para generar las recomendaciones.
- **Entrenamiento**: Vista en la que se puede reentrenar el modelo con las nuevas valoraciones.

3.3.2. Guía de estilo

En cuanto al estilo de la aplicación, al estar únicamente destinada a ser utilizada durante demostraciones del motor de recomendación, se ha decidido que se utilizarán los componentes de la guía de estilo de *Material Design*[1] los cuales han sido diseñados por expertos en diseño de aplicaciones y forman parte de la librería de componentes base de *Flutter*.

3.3.3. Prototipos

En la figura 3.11 se pueden ver los 4 prototipos realizados para la aplicación donde se muestran todas las vistas descritas en el *sitemap*.

A la hora del diseño de estos prototipos se ha tenido en cuenta que todas las ventanas sean lo más similares posibles, para así, hacer que el usuario se centre en la calidad de las recomendaciones generadas en base a los títulos valorados previamente.

Otro punto importante que se ha tenido en cuenta durante el desarrollo de los prototipos ha sido la distribución de las pantallas en la barra inferior de navegación. Se han ubicado de manera que cada ventana esté situada al lado de las que tienen mayor relación, para así, lograr que la navegación entre ventanas sea más visual.

Revisando los prototipos en el orden de la barra de navegación, el primero representa el listado general de películas, donde se listan todas las películas junto a la puntuación y características de estas.

El segundo prototipo, es la vista de las películas valoradas por el usuario. Como el contenido es muy similar al del listado de películas, también se muestra el prototipo del diálogo de cambio de usuario, cuyo botón de despliegue se encuentra en la barra superior común.

El tercer prototipo presenta el listado de recomendaciones. En este listado, se sigue la misma estructura que en los otros dos prototipos. Aunque en esta, las estrellas que representan la puntuación están vacías porque las recomendaciones siempre se realizan de películas no vistas.

Para finalizar, el cuarto prototipo representa la vista de control de estado del modelo. En esta destaca el recuadro gris, donde se muestran las estadísticas relativas a la precisión del modelo. En ella también se encuentra el botón que sirve para mandar la orden de entrenamiento del modelo de recomendación. Durante el proceso de entrenamiento este botón indicará que el modelo se está entrenando y no permitirá ser pulsado.

[1] Web oficial de Material Design: material.io

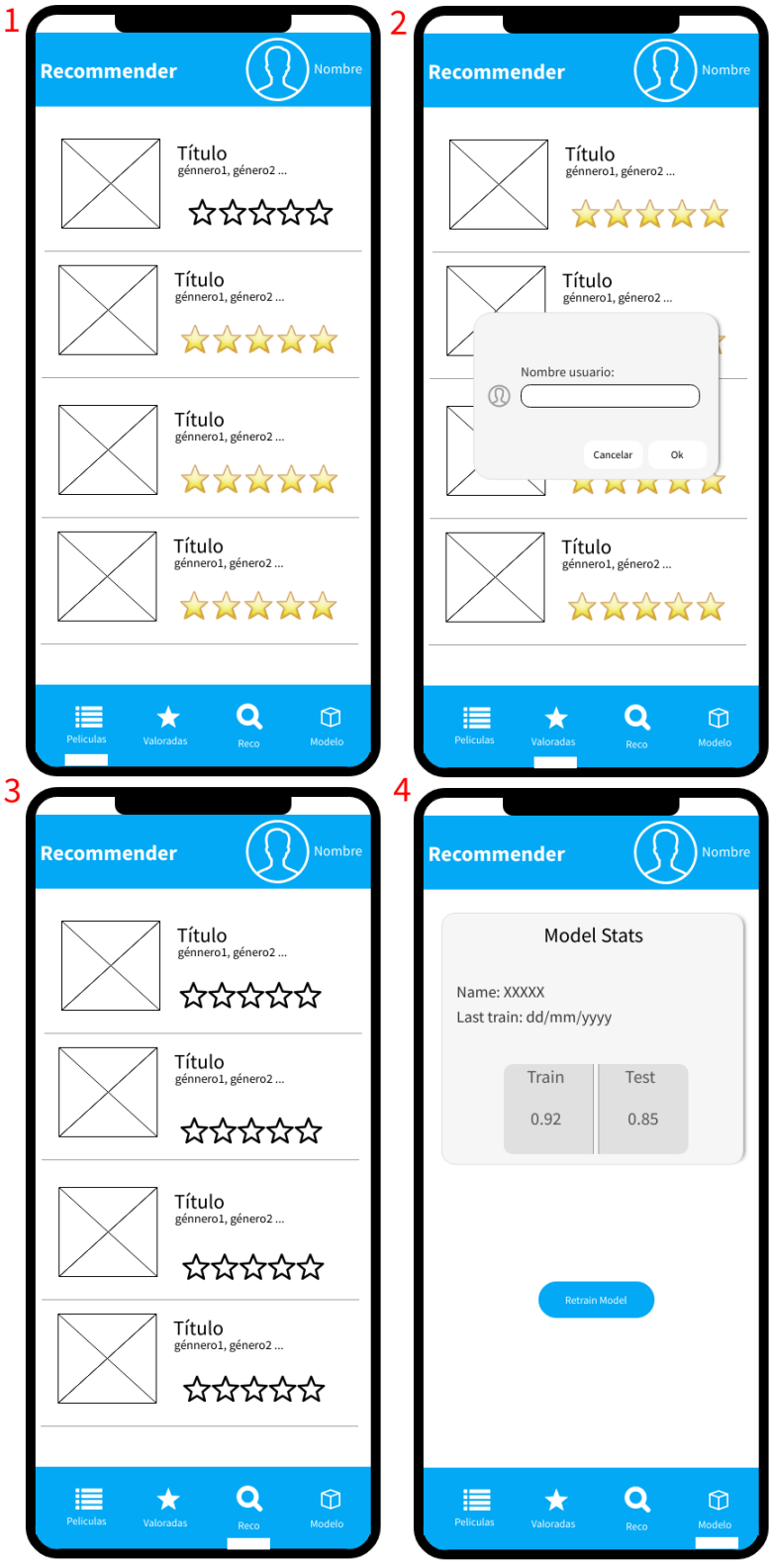

Figura 3.11: Prototipos de aplicación de demostración.

Capítulo 4

Implementación y pruebas

En este capítulo se detalla el proceso de implementación de las distintas funcionalidades del proyecto. Además, también se describirán todas las pruebas realizadas durante este proceso para asegurar que el motor de recomendación funciona correctamente.

4.1. Detalles de implementación

En esta sección se verán los detalles del proceso de implementación tanto del motor de recomendación como de la aplicación de demostración.

4.1.1. Árbol de ficheros del motor de recomendación

Como puede verse en el árbol de ficheros presente en la figura 4.1, cada fichero se identifica por un nombre que hace referencia a la finalidad de su contenido.

Cada uno de los distintos ficheros sirve para:

- **recoapi**: Contiene todos los directorios del motor de recomendación.
- **algorithms**: Contiene todas las implementaciones de algoritmos de recomendación.
- **db**: Contiene las clases encargadas de realizar operaciones de gestión sobre las distintas colecciones de la base de datos.
- **enums**: Contiene las definiciones de todas las enumeraciones requeridas.
- **router**: Es el encargado de contener las implementaciones de todos los endpoints de la API.
- **schemas**: Contiene las implementaciones de los modelos de datos del sistema.

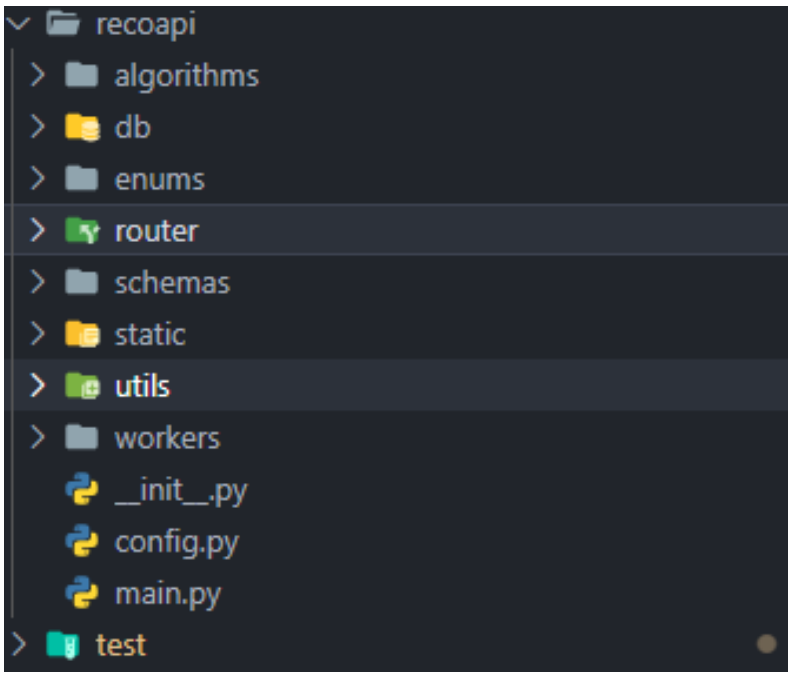

Figura 4.1: Árbol de ficheros de la implementación del motor de recomendación.

- **static**: Este directorio no contiene código, contiene todos los ficheros de documentación.
- **utils**: Contiene la implementación de métodos de utilidad y de control de acceso a la API.
- **workers**: Contiene la implementación de los *workers* encargados de realizar tareas en segundo plano.

4.1.2. Árbol de ficheros de aplicación de demostración

El árbol de ficheros presente en la figura 4.2 ha sido generado por *Flutter* y su estructura se debe conservar para su correcto funcionamiento. Esto es debido a que exceptuando el fichero *lib*, el cual contiene el código específico de la aplicación, el resto alberga ficheros de configuración relativos a las distintas plataformas compatibles.

Dentro de lib se han creado diversos ficheros para organizar la implementación de la aplicación. Cada fichero dentro de lib tiene la finalidad de:

- **models**: Es el fichero que contiene los modelos de datos.
- **pages**: Fichero en el que se encuentran las implementaciones de todas las vistas de la aplicación.
- **providers**: En este fichero se encuentran los *providers* de la aplicación los cuales se encargan de la obtención de los datos del *backend*.
- **widgets**: Contiene la implementación de los componentes que se utilizan en las distintas vistas de la aplicación.

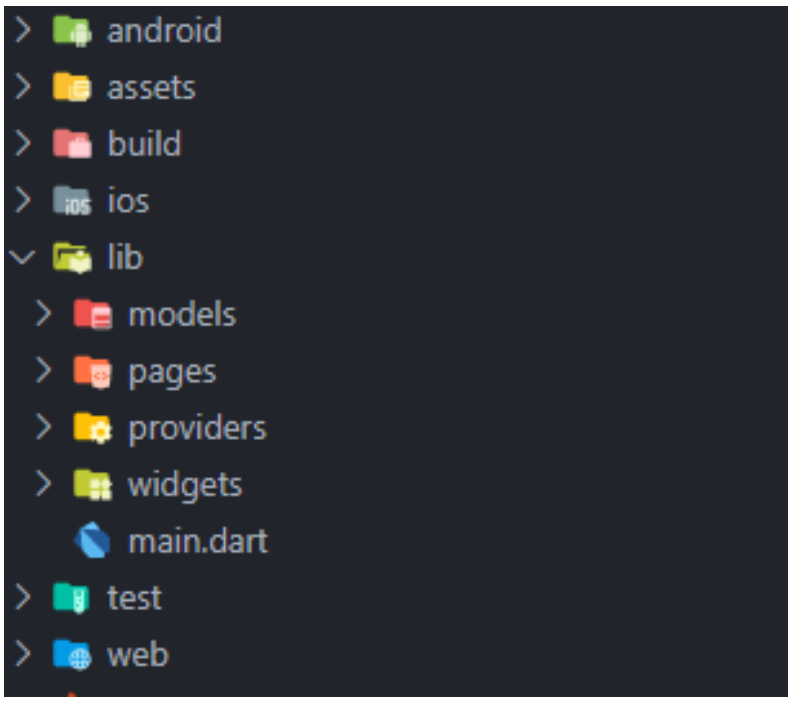

Figura 4.2: Árbol de ficheros de la implementación del motor de recomendación.

4.1.3. Documentación

Para la documentación del proyecto, se ha decidido seguir la especificación OpenAPI[4]. Esta especificación permite tanto a humanos como máquinas saber cuáles son los servicios ofrecidos por la API del sistema.

Seguir esta especificación conlleva distintas ventajas. Por un lado, a partir de esta se puede generar una web sencilla en la que figura una documentación interactiva de todos los endpoints que ofrece la API. Por otro lado, existen herramientas las cuales, dada la especificación de la API, son capaces de generar el código de la parte del cliente en múltiples lenguajes de programación agilizando así el proceso de desarrollo.

En la figura 4.3 se puede ver un ejemplo de uno de los *endpoints* del motor de recomendación. Para acceder a la documentación completa, se puede acceder mediante el siguiente enlace: [**documentación**].

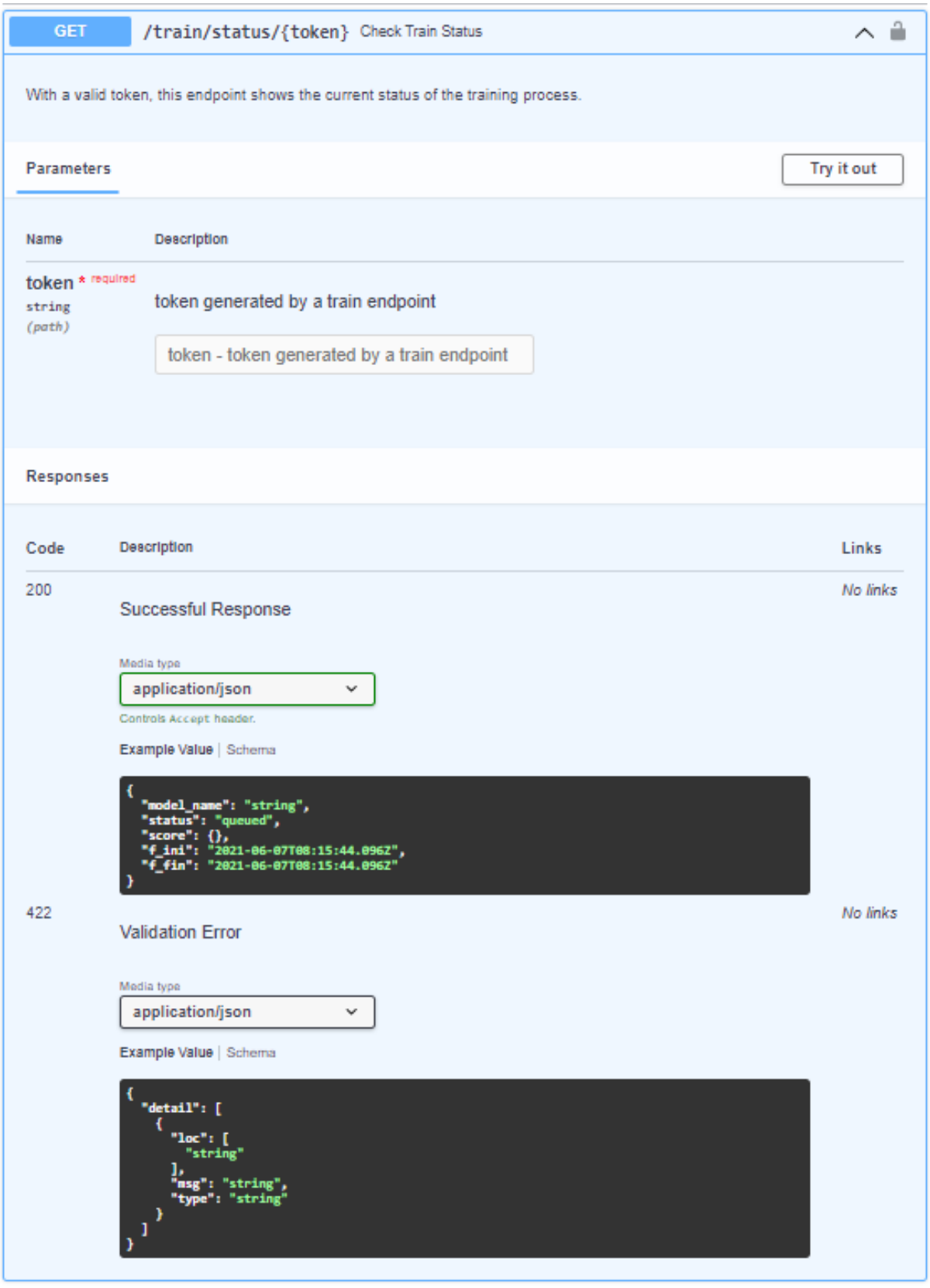

Figura 4.3: Documentación del *endpoint* para comprobar el estado del entrenamiento de un modelo.

En el caso de la documentación de la API del motor, se ha utilizado la generación automática incluida en FastAPI, ya que esta permite generarla a partir del código de los propios *endpoints*.

En el fragmento de código 4.1, se puede ver la implementación del *endpoint* presente en la figura 4.3. Como se puede deducir de ambas imágenes, el texto explicativo del *endpoint* es generado a partir del *docstring* y los datos referentes a los parámetros de entrada requeridos son especificados en la llamada del propio método.

```python
@router.get("/status/{token}", response_model=CheckStatus)
async def check_train_status(
    token: str = Path(..., description="token generated by a train endpoint"),
    user=Depends(get_current_user)
                             ):
    """
    With a valid token, this endpoint shows the current status of the training
    process.
    """

    return await TrainActions.get_status(token,user)
```

Listing 4.1: Ejemplo de documentación de endpoint.

4.1.4. Estilo arquitectónico

Para la implementación del motor de recomendación, uno de los objetivos que se debía de ofrecer era el de ser fácil de integrar. Por este motivo, se contempló la creación de una API siguiendo el estilo arquitectónico REST[3], del cual destacaría los siguientes principios:

- La interfaz debe de se uniforme y todos los recursos del sistema se deben de representar por una única URI.
- Debe de ser un protocolo sin estado, es decir, el servidor no almacena el contexto de una operación. Todos los mensajes deben de contener toda la información necesaria para realizar la operación.
- Se debe de seguir una arquitectura de tipo cliente servidor.
- Las respuestas deben de indicar si se deben de almacenar en caché.
- El sistema se puede distribuir en capas para separar distintas funcionalidades de la API sin que el usuario sea consciente de ello.

4.1.5. Patrones de diseño

Con respecto a los patrones de diseño, tanto en el motor de recomendación como en la aplicación de demostración, se han utilizado los más apropiados para solucionar los problemas surgidos durante el desarrollo.

Inyección de dependencias

Para el paso de parámetros y dependencias dentro del motor de recomendación se ha utilizado el patrón de inyección de dependencias. Este viene implementado por la librería FastAPI y se utiliza para inyectar información de utilidad en las llamadas de los *endpoint*.

Como se puede ver en el fragmento de código 4.2, se inyecta el nombre del usuario mediante el método *Depends* el cual inyecta el nombre del usuario que está enviando el mensaje a partir de una referencia al método *get_current_user* que recibe como parámetro.

```
@router.get("/get_product_consumers", response_model=List[UserRating])
def get_product(pid: str,user_name:str=Depends(get_current_user)):
    ...
```

Listing 4.2: Ejemplo de inyección de dependencias

Patrón Strategy

Para la realización de operaciones de actualización de datos se ha utilizado el patrón *Strategy*. Este permite cambiar el funcionamiento o estrategia de un objeto en tiempo de ejecución. Esta característica del patrón se ha aprovechado para realizar operaciones sobre los datos de manera dinámica.

En la figura 4.4 e puede ver un diagrama de clases de este patrón aplicado en el proyecto.

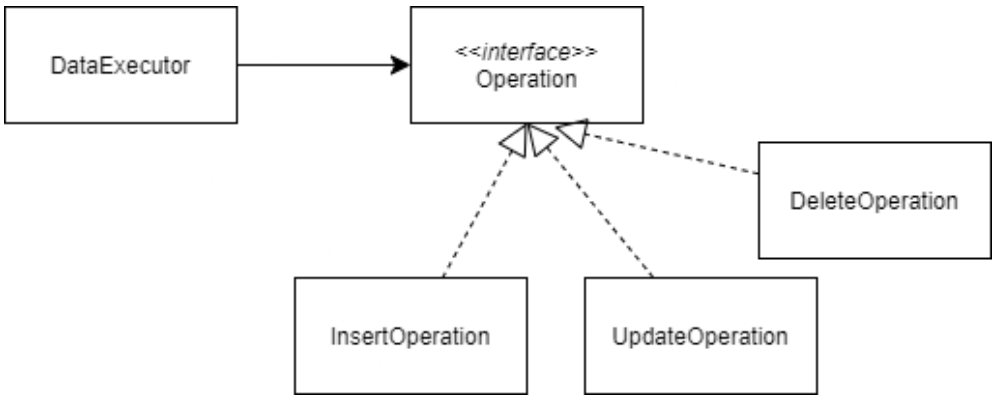

Figura 4.4: Diagrama patrón *Strategy*.

Patrón MVVM

En la aplicación de demostración se ha utilizado el patrón MVVM (*Model View ViewModel*). Este divide la lógica de la aplicación en tres componentes:

- *Model*, almacena el contexto de la aplicación y la capa de acceso a los datos.

- *View*, se encarga de toda la lógica de la visualización de la interfaz.
- *ViewModel*, se encarga de contener toda la lógica y datos requeridos por la vista. Cabe destacar que los datos contenidos por el *ViewModel* están directamente asociados a los de la vista por un enlace (en inglés *binding*) por lo que cualquier cambio sobre los datos se ve reflejado en ambos lados.

En la figura 4.5 se puede ver un el diagrama de clases de este patrón.

Figura 4.5: Diagrama patrón MVVM.

Patrón Composite

Para el diseño de las vistas de la aplicación se ha utilizado el patrón *Composite*, el cual sirve para estructurar los componentes de la vista en árboles de objetos. Este patrón permite trabajar sobre estos objetos compuestos como si se tratara de uno.

El patrón *Composite* es utilizado por defecto en *Flutter* a la hora de implementar la interfaz gráfica. Esto ofrece una enorme flexibilidad y permite crear tanto la lógica como las interfaces gráficas utilizando únicamente un lenguaje de programación.

En la figura 4.6 se puede ver el diagrama de clases que se debe de seguir para aplicar el patrón *Composite*.

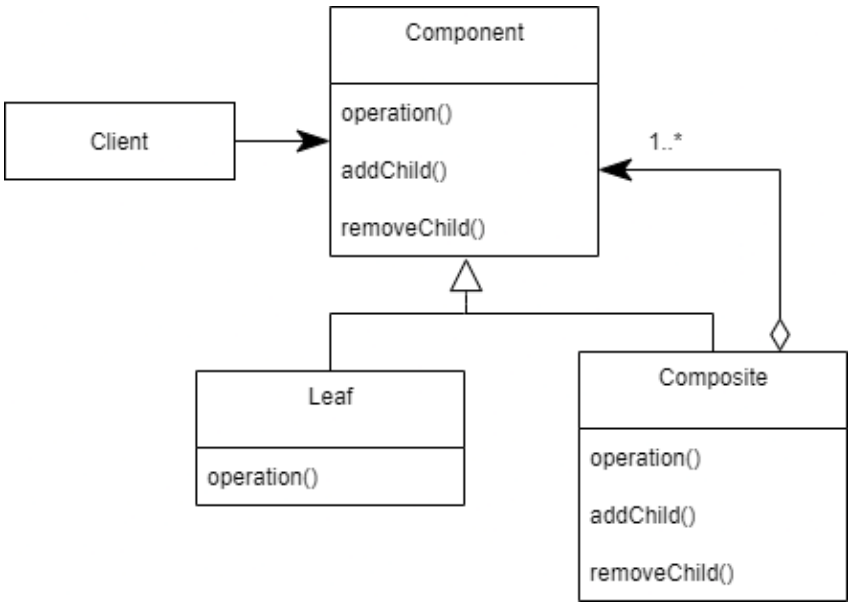

Figura 4.6: Diagrama patrón *Composite*.

4.1.6. Implementación algorítmica

Uno de los principales puntos del proyecto es su base algorítmica, ya que para generar recomendaciones a partir de los datos del usuario se requiere del uso de algoritmos de recomendación.

Dado que en este proyecto uno de los objetivos era el ser parametrizable y adaptarse a las necesidades del cliente, se ha decidido utilizar implementaciones de dos de los algoritmos de recomendación que mejores resultados dan.

La primera de las implementaciones ha consistido en un algoritmo de filtrado colaborativo basado en el algoritmo de aprendizaje automático de los k vecinos mas próximos KNN[1] combinado con el cálculo de matrices de similitud.

El funcionamiento del algoritmo se puede descomponer en los siguientes pasos:

1. Asignar una clave única a todos los usuarios y productos.
2. Crear una matriz de valoraciones donde las filas son los clientes y las columnas los productos.
3. A partir de la matriz de valoraciones, se crea una matriz de similitud producto-producto mediante uno de los algoritmos de cálculo ofrecidos por la librería *Surpise*, estos son: cosine[2], msd[3] y Pearson[4].
4. A partir de la matriz de similitud, se buscan los k vecinos mas similares a los productos previamente valorados por el usuario y a partir de estos mediante la fórmula 4.1 se genera una predicción de la valoración que el usuario le daría.

$$Pred(u,i) = \bar{r}_u + \frac{\sum_{j \in S_i} sim(i,j) \times r_{u,j}}{\sum_{j \in S_i} sim(i,j)} \tag{4.1}$$

Donde:

- S_i hace referencia al conjunto de tamaño **i** formado por los productos valorados previamente por el usuario.
- $r_{u,j}$ es la valoración dada por el usuario al producto **j**.
- $Pred(u,i)$ es la predicción de la valoración dada por el usuario **u** al producto **i**.
- $sim(i,j)$ es la similitud entre los productos **i** y **j**.
- $\bar{r}_u$ es la media de las valoraciones del usuario.

[1] KNN sirve para clasificar los elementos de un conjunto en función de la clase de los K vecinos mas próximos[14]

[2] Mide la similitud existente entre dos vectores a partir del coseno comprendido entre ambos dando un resultado entre 1 y -1 siendo que 1 significa igualdad.

[3] El msd o root-mean-square deviation sirve para calcular la diferencia entre pares de valores.[5]

[4] Calcula el coeficiente de correlación de Pearson el cual es un coeficiente de correlación lineal entre dos datos.[6]

La segunda implementación ha consistido en adaptar el algoritmo de recomendación híbrida basado en el *paper* con mismo nombre conocido como *Lightfm*[11]. *Lightfn* combina diversos algoritmos de filtrado colaborativo[5] con algoritmos de filtrado basado en contenido[6]. Esto permite que las recomendaciones tengan en cuenta tanto las valoraciones del usuario como las propias etiquetas relativas al usuario y producto.

Además, una de las ventajas y principales motivos por los que se decidió integrar este algoritmo de recomendación en el proyecto, es que no sufre de uno de los mayores problemas de los algoritmos de filtrado colaborativo conocido como *cold start*[7] (inicio frío) ya que en las situaciones de carencia de información utiliza las etiquetas del filtrado basado en contenido.

4.1.7. Implementación de la gestión de datos

Uno de los puntos más importantes para el correcto funcionamiento del motor de recomendación, a parte de los algoritmos, es la gestión de los datos utilizados para el entrenamiento de estos.

Para la implementación de esta funcionalidad, uno de los principales objetivos era que no se necesitara del uso de diversos *endpoint* para la ejecución de las operaciones básicas (crear, insertar, actualizar y eliminar). Esto va en contra de los principios REST que siguen el resto de *endpoints* de la API ya que ellos definen distintas cabeceras para cada tipo de operación (GET, DELETE, POST, PUT...).

La forma de gestionar estas operaciones se ha inspirado, en su mayoría, en el motor de búsqueda *ElasticSearch*. En este, desde un único *endpoint* se pueden realizar varias operaciones sobre los datos almacenados indicándolo mediante una etiqueta en la que figura la operación a realizar.

En el siguiente fragmento de código 4.3, se puede visualizar el formato de envío de los datos requeridos por el sistema. Estos datos se agrupan en un listado de operaciones descritas por el campo *action* y son ejecutadas en segundo plano dado que estas, dependiendo de la cantidad de datos y de la operación, pueden tardar varios minutos.

Como el proceso es realizado en segundo plano, el *endpoint* que permite realizar estas operaciones retorna un *token* único para realizar un seguimiento del estado de las operaciones.

[5] El filtrado colaborativo consiste en generar recomendaciones en base a otros productos (o usuarios) similares.

[6] El filtrado basado en contenido se basa en recomendar productos con etiquetas similares a los que se ha interactuado.

[7] El *cold start* es causado por la falta de datos a la hora de generar recomendaciones. Suele estar presente cuando se inicia un negocio y no se dispone de datos o cuando se registra un usuario o producto nuevo en el sistema.

```
[
  {
    "action": ENUM["INSERT","UPDATE","DELETE"],
    "data": {
      "interactions": [
        {
          "userId": "String",
          "productId": "String",
          "weight": "Number"
        }
      ],
      "users": [
        {
          "userId": "String",
          "features": Object
        }
      ],
      "products": [
        {
          "productId": "String",
          "features": Object,
          "metadata": Object
        }
      ]
    }
  }
]
```

Listing 4.3: Pseudo json para realizar operaciones sobre datos

4.1.8. Implementación del control de acceso

El control del acceso a los distintos *endpoints* de la API es fundamental por motivos de seguridad y privacidad porque desde la API se pueden acceder a datos sensibles del negocio.

Para la realización del control de acceso se ha aprovechado una de las funcionalidades de FastAPI, el control de acceso mediante *API Key*.

En cuanto a la creación de las *API Key*, se ha utilizado el estándar ampliamente utilizado en la industria conocido como JWT (*Json Web Token*), que permite compartir información codificada de manera segura.

En el fragmento de código 4.4 se puede ver la implementación del método encargado de validar a partir de una *API Key* si esta es válida y pertenece a un usuario existente en el sistema.

```python
api_key_header = APIKeyHeader(name="api_key", auto_error=False)

async def get_current_user(apk_encoded: str = Security(api_key_header)) -> str:
    try:
        decoded = jwt.decode(apk_encoded, JWT_SECRET, algorithms=['HS256'])
        apk = decoded["key"]
        tag = decoded["tag"]
    except Exception:
        raise HTTPException(
            status_code=status.HTTP_403_FORBIDDEN, detail="Wrong api key"
        )

    find = await db.api_key_db.find_one({"tag": tag})

    verified = False

    if find is not None:
        verified = bcrypt.verify(apk, find["hashed_key"])

    if verified:
        return tag
    else:
        raise HTTPException(
            status_code=status.HTTP_403_FORBIDDEN, detail="Wrong api key"
        )
```

Listing 4.4: Codigo de control de acceso mediante *API Key* en la cabecera

Para la creación de las claves de acceso, al tratarse de un punto delicado, se ha decidido implementarlo como un *script* de línea de comandos simple, el cual puede utilizarse desde el terminal del servidor.

En el fragmento de código 4.5 se puede ver el método encargado de la creación de una nueva clave de acceso mediante JWT. Cabe destacar que dentro de las claves generadas se incluye un token hexadecimal y una etiqueta para identificar el usuario.

```python
async def generate_api_key(tag: str) -> str:
    find = await db.api_key_db.find_one({"tag": tag})

    if find is None:
        key = secrets.token_hex(nbytes=32)
        hashed_key = bcrypt.hash(key)
        apd = ApiKey(tag=tag, hashed_key=hashed_key)
        await db.api_key_db.insert_one(apd.dict())
        encoded = jwt.encode({"tag": tag, "key": key}, JWT_SECRET)
        return encoded
    else:
        raise Exception(f"Tag {tag} is already in use")
```

Listing 4.5: Codigo de generacón de*API Key*

4.1.9. Implementación de la API

Para la implementación de la API se ha utilizado FastAPI el cual es un *framework* de desarrollo de APIs REST. FastAPI se caracteriza por ser muy sencillo de utilizar, y a diferencia de otros *frameworks* web de *Python*, este es asíncrono y tiene validación de parámetros.

Como puede verse en el fragmento de código 4.6, los endpoints se implementan a partir de métodos *Python* simples a los cuales se les ha añadido un decorador donde se indica el tipo de operación y opcionalmente el tipo de respuesta.

Una característica de *FastAPI* es que para la validación de los datos se utilizan las etiquetas de tipado opcional que ofrece *Python*, sólo que en este caso son obligatorias.

```
@router.get("/record/{model_name}", response_model=List[TrainStatistics])
async def model_records(model_name: str = Path(..., description="Name of the
    model"),user=Depends(get_current_user)):
    """
    Returns the train records of a model
    """
    await check_model(model_name, user, CheckOptions.exists_model)
    records =  await TrainActions.get_train_history(user,model_name)
    records = sorted(records,key=lambda x: x.date)
    records = reversed(records)
    return list(records)
```

Listing 4.6: Ejemplo de endpoint de la API

En cuanto a los endpoints implementados, en la documentación del sistema se puede ver una definición detallada de todos ellos. [**documentación**]

4.1.10. Implementación de la aplicación móvil

Como se ha mencionado en apartados anteriores, la aplicación móvil del sistema se ha implementado utilizando *Flutter* el cual es un *framework* multiplataforma.

Flutter se caracteriza por utilizar un único lenguaje de programación (Dart) para desarrollar tanto las interfaces gráficas como la lógica interna. En el fragmento de código 4.7 se puede ver un ejemplo de un componente de la aplicación, en concreto, es el componente encargado de mostrar el diálogo de cambio de usuario activo.

```
void openDialog() {
    showDialog(
      context: context,
      barrierDismissible: false,
      builder: (context) => AlertDialog(
        shape: RoundedRectangleBorder(
          borderRadius: BorderRadius.all(Radius.circular(20)),
        ),
        title: Text(
          'Switch current user',
          textAlign: TextAlign.center,
        ),
        content: _crearInput(),
        actions: [
          TextButton(
              onPressed: () => Navigator.of(context).pop(),
              child: Text('Cancelar')),
          TextButton(
            onPressed: () {
              widget.action(_nombre);
              Navigator.of(context).pop();
            },
            child: Text('Ok'),
          ),
        ],
      ),
    );
  }
```

Listing 4.7: Ejemplo de componente de Flutter

La lógica interna de la aplicación, como puede verse en el fragmento de código 4.8, consiste en la realización de llamadas HTTP a la API del motor de recomendación. Como gran parte de las llamadas tratan con datos similares se ha implementado el método *_procesaRespuesta* el cual se encarga de realizar todas las peticiones de tipo *get* al servidor.

```
Future<List<PeliculaBase>> getRatedMovies(String user) async {
    final recoapiURL = Uri.http(_urlLocal, 'data/get_rated_products',
        {'model_name': modelName, 'uid': user});

    final resp = await _procesarRespuesta(recoapiURL);
    this.peliculasValoradas = {};
    resp.forEach((p) => this.peliculasValoradas[p.title] = p);

    notifyListeners();
    return peliculasValoradas.values.toList();
  }

Future<List<PeliculaBase>> _procesarRespuesta(Uri url) async {
    final resp = await http.get(url, headers: {'api_key': _recoapiKey});
    final decodedData = json.decode(resp.body);

    final peliculas = PeliculasBase.fromJsonList(decodedData);

    return peliculas.movies;
  }
```

Listing 4.8: Ejemplo de la lógica interna de la aplicación móvil

Las interfaces se han implementado siguiendo los prototipos creados durante la fase de diseño (figura 3.11). Durante la implementación de las vistas se ha intentado ser lo más fiel a estos, en la figura 4.7 se puede ver el aspecto final de la aplicación tras su implementación.

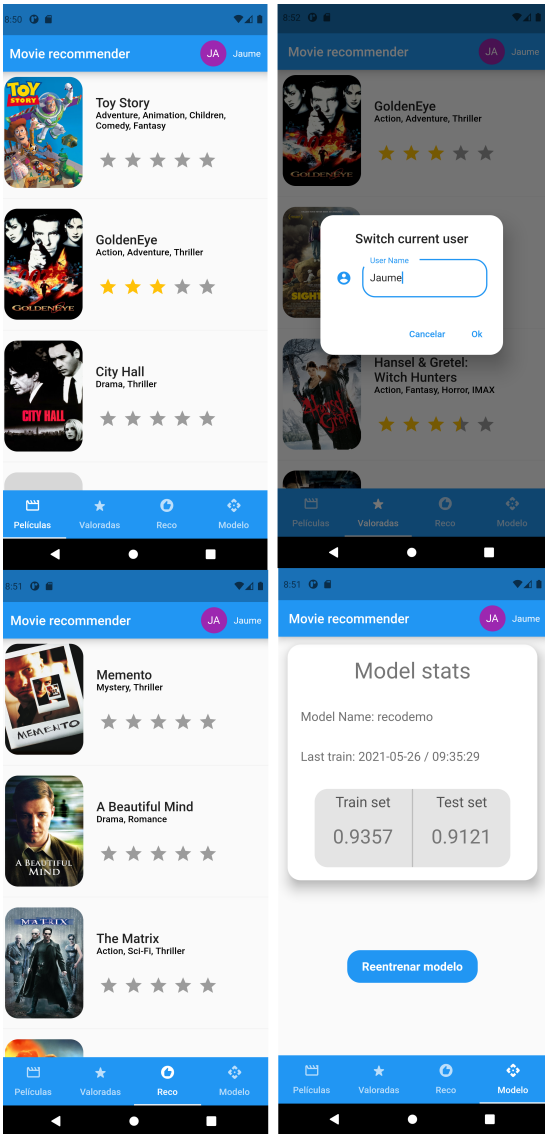

Figura 4.7: Vistas aplicación móvil.

4.2. Problemas y dificultades

En esta sección se describirán los principales problemas y dificultades que han surgido durante el desarrollo del proyecto.

Empezando por las dificultades, una de las más presentes a lo largo de todo el proyecto ha sido la falta de experiencia con las bases de datos no relacionales. Esta carencia fue la causa de que se tuviese que rehacer la forma en la que se estructuran los datos varias veces.

Otra de las dificultades ha sido la falta de experiencia y conocimientos del mundo de los sistemas de recomendación. Esto causó que costara más tiempo adaptar e incluir los distintos algoritmos de recomendación al tener que investigar y probar distintas alternativas.

Pasando a los problemas, el que más costó solventar fue el del funcionamiento del proceso de entrenamiento de los modelos de recomendación. Inicialmente, el entrenamiento era realizado en segundo plano dentro del subsistema de la API. Aparentemente, todo funcionaba correctamente, pero al pasarlo a *Docker* con un sistema operativo basado en *Linux*, surgió un problema que no se consiguió resolver. Al realizar el proceso, la API se quedaba congelada y no aceptaba ni devolvía mensajes.

Tras revisar todas las posibles causas se descubrió que era un problema relacionado con el sistema operativo del contenedor *Docker*, por lo que, como solución, se decidió el desplazar el proceso de entrenamiento a *workers* alojados en otros subsistemas.

En la figura 4.8 se puede ver una imagen que representa la solución descrita. En la imagen se puede distinguir cómo la API utiliza la librería *Celery* para crear tareas, las cuales son enviadas a la cola de tareas gestionada por el software gestor de mensajes *RabbitMQ*. Este sistema hace la función de encolar las tareas y distribuirlas entre los distintos *workers* de *Celery* que son sistemas gestionados por *Celery* encargados de ejecutar las tareas que se les asignan.

Una de las ventajas de solucionar el problema de esta manera ha sido que se ha ganado en escalabilidad de la API ya que ahora se pueden agregar tantos workers de entrenamiento como sea necesario.

El último problema relevante ha sido relativo al acceso a la base de datos. Inicialmente se utilizaba la librería oficial de *Mongodb* para *Python* que realiza las operaciones de manera síncrona. El problema surgió cuando se querían realizar operaciones de gran carga, ya que estas dejaban bloqueada la API y ralentizaban el tiempo de respuesta. La solución consistió en cambiar la librería oficial por *Motor* la cual es una adaptación de la librería oficial creada con la finalidad de trabajar de manera asíncrona.

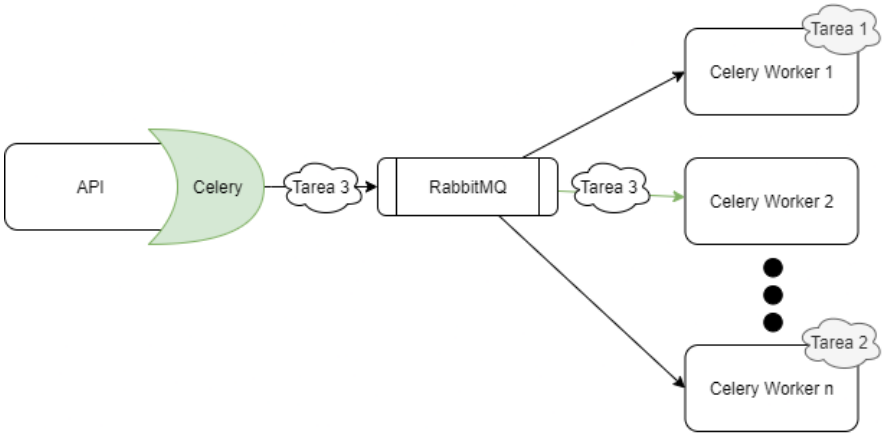

Figura 4.8: Representación de funcionamiento de la cola de procesos de entrenamiento.

4.3. Verificación y validación

En todo proyecto de desarrollo de software es vital que todos los componentes del sistema funcionen correctamente y de la manera que se espera. Es por esto que la correcta realización de pruebas de verificación y validación sea fundamental para el éxito del producto.

En esta sección se describirán las pruebas que se han realizado para asegurar el correcto funcionamiento del sistema.

4.3.1. Análisis estático del código

Durante el transcurso del proyecto se han realizado 2 revisiones de código para buscar tanto errores relativos a la lógica como mejoras de la calidad de este. Estas revisiones han consistido en hacer trazas mentales de la ejecución de este para detectar los posibles problemas en su lógica.

A parte de la revisión manual, también se han utilizado durante el desarrollo herramientas de análisis estático[8] del código, en concreto se ha utilizado *Pylint*, herramienta que ofrece entre sus funcionalidades el análisis estático del código, detectando y destacando posibles fallos lógicos y sintácticos y ofreciendo una puntuación de la calidad del código.

[8]El análisis estático consiste en analizar el correcto funcionamiento del código sin ejecutarlo

4.3.2. Pruebas de integración

Dada la naturaleza del proyecto, se han realizado pruebas de integración porque el motor de recomendación está compuesto por varios subsistemas como se ve en la figura 3.8 y esto dificulta la creación de otro tipo de pruebas.

En total se han realizado 33 pruebas de integración organizadas por funcionalidad específica de la API. En la figura 4.9 se puede ver cómo se han organizado las distintas pruebas en distintos archivos donde están nombrados siguiendo una convención que consiste en iniciar su nombre con *test* seguido de la funcionalidad que se va a testear. Por ejemplo *test_knn_model_creation*.

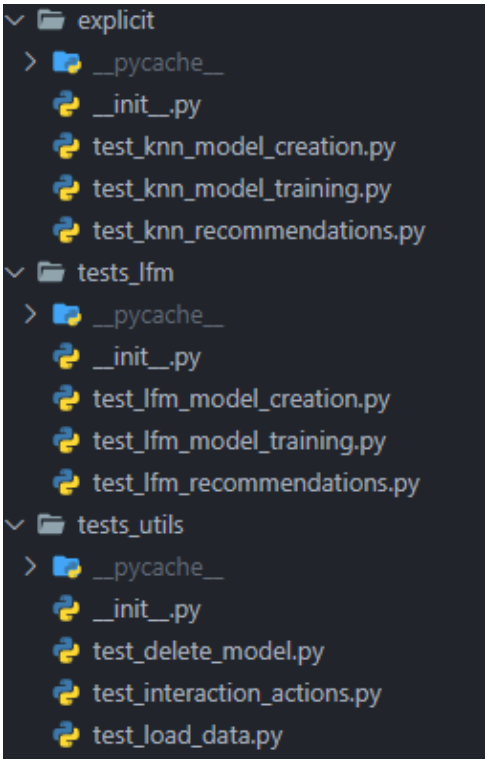

Figura 4.9: Estructura de las pruebas de aceptación del sistema.

Las pruebas se han realizado sobre los distintos endpoints de la API para comprobar que el comportamiento del sistema ante distintas entradas de valores frontera es el esperado. Un ejemplo de implementación de una de las pruebas se puede ver en el fragmento de código 4.9 donde se prueba la creación de un nuevo modelo de recomendación.

```
def test_new_lfm_model(api_token):
    with TestClient(app) as client:

        test_model = "test_lfm"
        response = client.post(f"/lightfm/?model_name={test_model}",
                               headers={**api_token},
                               json=LFM_BASE_CONFIG)
        assert response.status_code == 200
        assert response.json() == {"status": "done"}
```

Listing 4.9: Ejemplo de prueba de integración

Capítulo 5

Conclusiones

La realización del proyecto durante mi estancia en prácticas ha sido un éxito porque se han implementado todas las funcionalidades previstas en menos tiempo de lo que se esperaba inicialmente. El haber terminado antes de lo previsto, me ha permitido añadir mejoras y extras al proyecto, tales como, la aplicación de demostración. Uno de los principales motivos de la rapidez en el desarrollo del proyecto se ha debido a las tecnologías escogidas. De entre ellas *FastAPI* ha resultado ser una de las que destacaría por su sencillez y enorme utilidad.

El desarrollo del motor de recomendación, juntamente con la aplicación de ejemplo, me han servido para aprender una enorme cantidad de conocimientos, tanto técnicos como teóricos. Al inicio de las prácticas empecé leyendo el libro *Practical Recommender Systems by Kim Falk[2]*, el cual me introdujo en el mundo de los sistemas de recomendación y en el que aprendí muchos conceptos y algoritmos que me fueron de gran utilidad a la hora de implementar el motor de recomendación. También dediqué muchas horas leyendo artículos y guías en línea para aprender a utilizar las distintas herramientas y tecnologías (*FastAPI*, *Flutter*, *Docker*...) ya que la mayoría eran nuevas para mi.

Además de los nuevos conocimientos mencionados, un aspecto destacable del proyecto es que este me ha permitido poner en práctica gran parte de las enseñanzas que he adquirido durante los cuatro cursos de ingeniería informática. En especial, destacaría los conocimientos relativos a la rama de ingeniería del software. De entre las asignaturas de esta rama, resaltaría:

- **Mètodes Àgils**: me ha sido de gran ayuda a la hora de aplicar la metodología *Scrum*.
- **Diseño de Software**: me ha servido para saber cuándo y qué patrones de diseño debía de aplicar para que el software fuera de calidad.
- **Verificación y Validación** y **Paradigmes del Programari**: me han aportado los conocimientos necesarios para realizar las distintas pruebas sobre el sistema.

En cuanto al ámbito profesional, mi experiencia en la empresa ha sido muy buena. Los compañeros siempre han estado dispuestos a ayudar en lo posible y se han mostrado muy entusiastas en mi trabajo realizado. Este interés, juntamente con el apoyo que me ha brindado Sergio, me ha ayudado a integrarme en el equipo y sentirme uno más.

Otro aspecto positivo de mi estancia en la empresa ha sido que he podido formar parte de un entorno profesional. He asistido a varias reuniones con clientes y he participado activamente en las reuniones de departamento. Esto me ha hecho comprender mejor cómo funciona el mundo del desarrollo de *software* de primera mano.

En los aspectos personales, si bien es cierto que por lo general mi estancia y elaboración del proyecto han sido experiencias muy buenas, en un inicio, dada mi poca experiencia, me sentía abrumado por el miedo de no ser capaz de estar a la altura y de no poder obtener los resultados esperados. Este miedo solo lo tuve durante los primeros días, pero como ya he mencionado antes, gracias a mis compañeros que me apoyaron en todo momento este se fue desvaneciendo.

Para finalizar, cabe resaltar que a mitad de mi estancia en prácticas se me ofreció la opción de incorporarme a la plantilla de Cuatroochenta una vez finalizase mis prácticas, para así seguir con la puesta en producción del proyecto. Sin lugar a dudas la acepté.

Bibliografía

[1] Celery. Celery - distributed task queue, 2009. URL https://docs.celeryproject.org/en/stable/.

[2] Kim Falk. *Practical Recommender Systems*. Manning, 2019.

[3] Roy Fielding. Representational state transfer (rest), 2000. URL https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm.

[4] Linux Fundation. Openapi specification, 2021. URL https://spec.openapis.org/oas/v3.1.0.

[5] Stephanie Glen. Rmse: Root mean square error, 2021. URL https://www.statisticshowto.com/probability-and-statistics/regression-analysis/rmse-root-mean-square-error/.

[6] Stephanie Glen. Correlation coefficient: Simple definition, formula, easy steps, 2021. URL https://www.statisticshowto.com/probability-and-statistics/correlation-coefficient-formula/.

[7] Google. Flutter, 2018. URL https://flutter.dev/.

[8] Nicolas Hug. Surprise, a python sdk for recommender systems, 2017. URL http://surpriselib.com/.

[9] Solomon Hykes. Docker, 2013. URL https://www.docker.com/.

[10] MongoDB Inc. Mongodb, 2009. URL https://www.mongodb.com/.

[11] Maciej Kula. Metadata embeddings for user and item cold-start recommendations, 2015. URL https://arxiv.org/pdf/1507.08439.pdf.

[12] Lyst. Lightfm's documentation, 2015. URL https://making.lyst.com/lightfm/docs/home.html.

[13] Microsoft. Visual studio code, 2015. URL https://code.visualstudio.com/docs.

[14] Abdelmalik Moujahid, Iñaki Inza, and Pedro Larrañaga. Clasificadores k-nn, 2021. URL http://www.sc.ehu.es/ccwbayes/docencia/mmcc/docs/t9knn.pdf.

[15] pytest dev. Pytest, 2010. URL https://docs.pytest.org/.

[16] Sebastián Ramírez. Fastapi, 2018. URL https://fastapi.tiangolo.com/.

[17] Pivotal Software. Rabbirmq, 2007. URL https://www.rabbitmq.com/.

[18] Guido van Rossum. Python, 1991. URL https://www.python.org/.