Escuela Politécnica Superior de Jaén
Máster Universitario en Ingeniería Informática

UNIVERSIDAD DE JAÉN
Escuela Politécnica Superior (Jaén)

Trabajo Fin de Máster

MYSELFLIX: PLATAFORMA MÓVIL EN LA NUBE PARA LA RECOMENDACIÓN DE PELÍCULAS

Alumno/a: **Serrano Casas, Javier**

Tutor/a: Don José Ramón Balsas Almagro

Dpto.: Lenguajes y Sistemas Informáticos

Enero, 2024

Universidad de Jaén

Departamento de Informática

Don José Ramón Balsas Almagro, tutor del Trabajo Fin de Máster titulado: '**Myselflix: Plataforma móvil en la nube para la recomendación de películas**', que presenta Javier Serrano Casas, otorga el visto bueno para su entrega y defensa en la Escuela Politécnica Superior de Jaén.

Jaén, Enero de 2024

El alumno: El tutor:

Javier Serrano Casas Don José Ramón Balsas Almagro

FICHA DEL TRABAJO FIN DE MÁSTER	
Titulación	Máster Universitario en Ingeniería Informática
Modalidad	Proyecto de Ingeniería
Idioma	Castellano
Tipo	Específico
TFT en equipo	No
Código TFM	21/22-754
Fecha de asignación	16/01/2023
Descripción corta	El presente trabajo propone el desarrollo de un sistema informático basado en una aplicación móvil multiplataforma que revele a los usuarios una serie de películas atendiendo a sus intereses. El sistema ofrecerá a los usuarios la capacidad de valorar películas en función de sus gustos y así incrementar los datos con sus preferencias que mejorarán las recomendaciones. La aplicación interactúa con un back-end en la nube que gestionará toda la información e integrará el sistema de recomendación, además de ofrecer al usuario información relevante sobre el contenido recomendado.

Contenido

Índice de Ilustraciones

Índice de Tablas

1. INTRODUCCIÓN

Las tecnologías de la información y la comunicación, también conocidas como TIC, constituyen una parte importante en la vida cotidiana de la sociedad actual. Cada vez más y en mayor medida dependemos de estas tecnologías que nos ofrecen todo tipo de herramientas y comodidades para ayudarnos en el día a día. Un claro ejemplo del uso de tecnologías de la información por parte de la ciudadanía lo encontramos en los dispositivos móviles, los cuales se han convertido en un elemento fundamental que no deja de evolucionar y ser cada vez más imprescindible Informes como Digital Report 2022 de la plataforma de administración de redes sociales “Hootsuite” y la agencia creativa “We Are Social” [1] determinan que en España el 94% de personas que habitan el país hacen uso de internet, y que el 92.3% lo hacen a través de sus dispositivos móviles.

Además encontramos estas tecnologías no solo como una herramienta, sino como una fuente de entretenimiento casi ilimitado. Una de las opciones de entretenimiento con más éxito entre la población de 16 a 64 años, es la visualización de contenidos de TV en streaming vía internet mediante plataformas como Netflix, Prime Video, Disney+, HBO o Apple TV. Casi el 94% de la población mundial en la franja de edad comentada, hacen uso de este contenido en streaming [1]. Lo que se traduce en billones de personas buscando que ver, entre millones de opciones diferentes a cada momento, de ahí que nazca la necesidad de crear sistemas capaces de recomendarnos productos que pueden ser de nuestro agrado en función de nuestros gustos. Esto es lo que se conoce como Sistemas de Recomendación, sistemas cuya tarea principal es seleccionar ciertos objetos en un gran espacio de búsqueda, de acuerdo con los requisitos del usuario, haciendo uso de las opiniones de otros usuarios de la comunidad.

La principal finalidad de este Trabajo Fin de Máster es, por lo tanto, crear una aplicación móvil basada en un Sistema de Recomendación de películas que ayude a los usuarios a conocer contenido similar al de sus preferencias y les reduzca el tiempo que pasan decidiendo qué contenido ver. Este sistema será además multiplataforma y basado en la nube, de esta manera podrá llegar a un mayor número de usuarios.

1.1. Antecedentes

Los sistemas de recomendación son un tipo de sistema de filtrado de información que trata de predecir la valoración o preferencia que un usuario daría a un artículo. Se utilizan ampliamente en muchos ámbitos diferentes, como el comercio electrónico, la música, las películas y las redes sociales. El objetivo principal de los sistemas de recomendación es ayudar a los usuarios a encontrar productos, servicios o información de su interés en un gran volumen de opciones disponibles. Demuestran ser una herramienta valiosa para mejorar la experiencia del usuario, aumentar las ventas y fidelizar a los clientes.

Existen varios tipos de sistemas de recomendación, incluyendo filtrado colaborativo, filtrado basado en contenido, sistemas híbridos y sistemas de recomendación basados en conocimiento. Cada uno de estos enfoques utiliza diferentes técnicas para generar recomendaciones precisas y relevantes para los usuarios.

1.1.1. Sistemas basados en conocimiento

Estos modelos utilizan la información sobre las preferencias y necesidades de los usuarios para generar recomendaciones. El conocimiento se obtiene de diversas fuentes, como expertos en el dominio, bases de datos, catálogos de productos, o información histórica, entre otros, y se utiliza para recomendar elementos que se ajusten a las necesidades y preferencias del usuario.

Los sistemas basados en contenido buscan nuevos productos a recomendar basándose en los valorados por el usuario en el pasado. Por tanto, estos sistemas requieren que el usuario haya valorado un mínimo número de productos para realizar las recomendaciones adecuadas a su proceso de búsqueda de nuevos productos.

Aunque los Sistemas de Recomendación más utilizados y más conocidos son los colaborativos y los basados en contenido, no en todas las situaciones son los más adecuados, como por ejemplo si no se dispone de una base de datos con valoraciones de los usuarios sobre los productos ofertados, ya que estos sistemas requieren que el usuario haya valorado un mínimo número de productos para realizar las recomendaciones adecuadas a su proceso de búsqueda de nuevos productos [2].

En cambio, los sistemas basados en conocimiento si ofrecerían recomendaciones bien encaminadas, ya que solo necesitan las propias valoraciones del usuario al que se le recomienda.

La búsqueda de Google, utiliza un sistema de recomendación basado en el conocimiento para recomendar resultados de búsqueda a sus usuarios. También por ejemplo, encontramos a TripAdvisor que utiliza este filtrado para recomendar destinos de viaje a sus usuarios. El sistema analiza las reseñas de destinos de viaje realizadas por otros usuarios para recomendar destinos que probablemente sean de interés para el usuario. En la siguiente ilustración podemos observar un ejemplo de como recomienda esta aplicación.

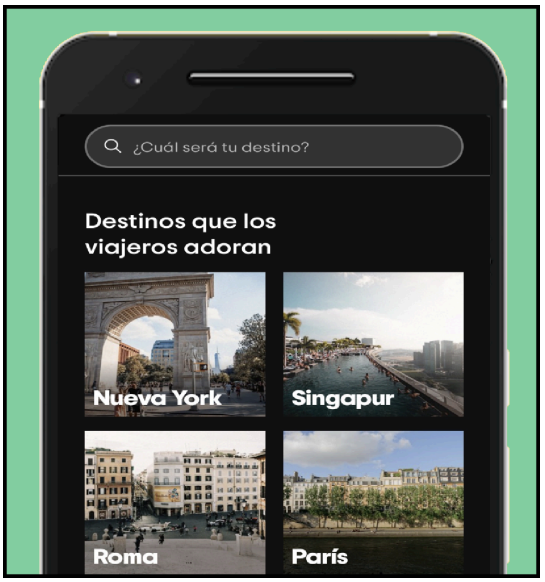

Ilustración 1: Demostración de recomendación en la aplicación de TripAdvisor.

1.1.2. Sistemas de filtrado basado en contenido

Los sistemas de recomendación basados en contenido son una técnica utilizada en los sistemas de recomendación que analizan las características y propiedades de los elementos recomendados, como películas, música o productos, para hacer recomendaciones similares.

Estos sistemas buscan encontrar similitudes en las características de los elementos recomendados y los que el usuario ha mostrado interés en el pasado.

Por ejemplo, en un sistema de recomendación de películas basado en contenido, el sistema analiza las características de la película que el usuario ha visto y ha calificado positivamente, como el género, el director, el elenco, la trama, etc.

El sistema luego recomendaría películas similares que comparten características similares a las películas que el usuario ha calificado positivamente.

Los sistemas de recomendación basados en contenido son útiles en situaciones en las que los datos del usuario son limitados o cuando se recomiendan elementos únicos, como libros o música. Sin embargo, estos sistemas tienen la limitación de no ser capaces de capturar gustos o preferencias en evolución del usuario.

IMDb utiliza principalmente un sistema de recomendación basado en contenido, y aunque también puede ofrecer algunas recomendaciones adicionales basadas en la popularidad, tendencias y evaluaciones de otros usuarios, su enfoque principal se centra en las características y metadatos de los contenidos para brindar recomendaciones a los usuarios.

En la Ilustración 2 vemos un ejemplo de cómo el sistema de IMDb recomienda al usuario indicando que le pueden interesar ciertas películas según las valoraciones que haya dado anteriormente.

Ilustración 2: Demostración de recomendación en aplicación IMDb.

1.1.3. Sistemas de filtrado colaborativo

El filtrado colaborativo es una técnica utilizada en los sistemas de recomendación que se basa en el análisis de las interacciones entre usuarios y elementos recomendados. Estos sistemas buscan identificar patrones en las interacciones de los usuarios con los elementos recomendados para hacer recomendaciones similares a usuarios con patrones de interacción similares.

Por ejemplo, si un grupo de usuarios ha mostrado interés en las mismas películas o canciones, un sistema de recomendación basado en filtrado colaborativo podría recomendarles elementos similares que otros usuarios con patrones de interacción similares hayan encontrado interesantes.

Su éxito radica en que precisamente las personas utilizamos de forma natural un proceso de recomendación muy similar, transmitiendo nuestra opinión a otras personas con la esperanza de que nuestra experiencia les pueda ser útil.

Pongamos en caso que la última película que hemos visto en el cine nos ha gustado especialmente, enseguida se la recomendaremos a nuestros amigos, y de igual forma, ellos nos recomendarán aquellos productos que nos pueden interesar y nos alertarán de aquellos que han supuesto una decepción. Sin embargo, por muy buenas que sean sus intenciones, no depositamos la misma confianza en las opiniones de todos ellos. Con el tiempo, nos damos cuenta de que las recomendaciones de algunos de nuestros amigos son más útiles que las de otros. Es decir, tendemos a confiar más en las opiniones de personas con gustos o intereses similares a los nuestros [3].

La principal diferencia entre un sistema de filtrado colaborativo y la recomendación boca a boca es que en lugar de restringirse a nuestro círculo de amistades, el sistema puede tener en cuenta las opiniones de miles o incluso millones de personas, aprovechando por tanto el conocimiento y experiencias de todas ellas [3].

Existen dos tipos principales de técnicas de filtrado colaborativo: el filtrado colaborativo basado en usuarios y el filtrado colaborativo basado en elementos. A continuación podemos ver una ilustración bastante representativa de cómo funcionan estos filtrados con respecto a la relación entre ítems y usuarios.

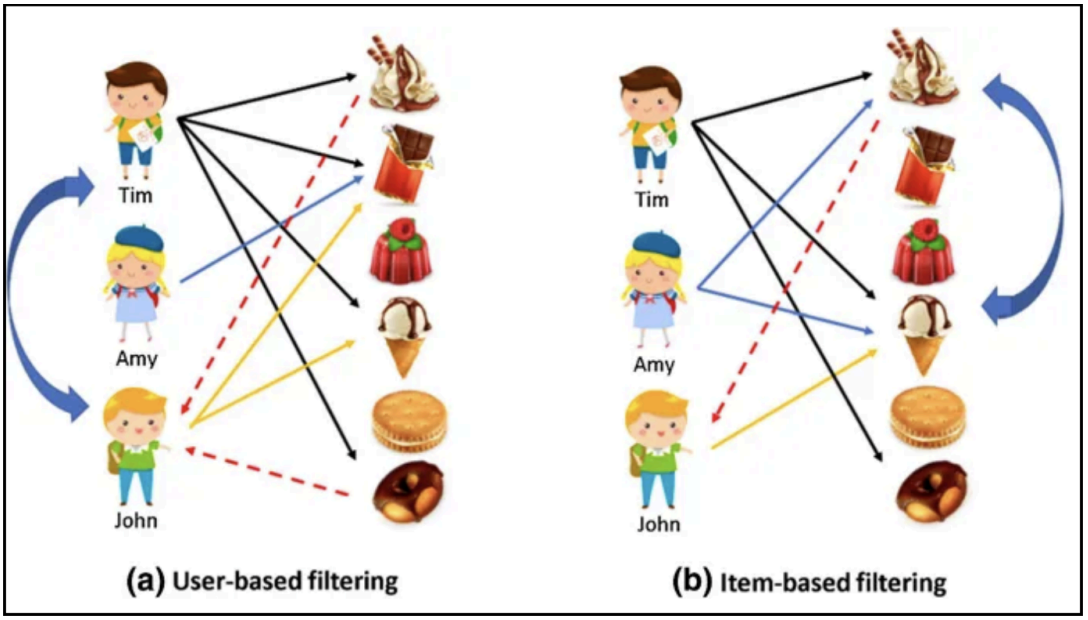

Ilustración 3: Filtrados colaborativos: Basados en usuarios y en ítems.

Ambos tipos se basan en el análisis de las interacciones de los usuarios con los elementos recomendados, pero difieren en la forma en que se realizan las recomendaciones.

El filtrado colaborativo basado en usuarios se basa en la similitud entre usuarios. Este tipo de técnica busca usuarios que hayan tenido patrones de interacción similares en el pasado y, a continuación, recomienda elementos que han sido valorados positivamente por esos usuarios.

Por otro lado, el filtrado colaborativo basado en elementos se basa en la similitud entre elementos. Este tipo de técnica busca elementos que hayan sido valorados de forma similar por los usuarios en el pasado y, a continuación, recomienda elementos similares a aquellos que el usuario ha valorado positivamente.

Es importante destacar que, en la práctica, se suelen combinar ambas técnicas para obtener recomendaciones más precisas y personalizadas [4].

Un ejemplo bastante conocido como aplicación que utiliza este filtrado es Amazon, que proporciona recomendaciones de productos basadas en las compras y evaluaciones de otros usuarios con gustos y preferencias similares. En la siguiente ilustración se puede comprobar como el usuario visualiza estas recomendaciones en base a las compras anteriores que haya realizado.

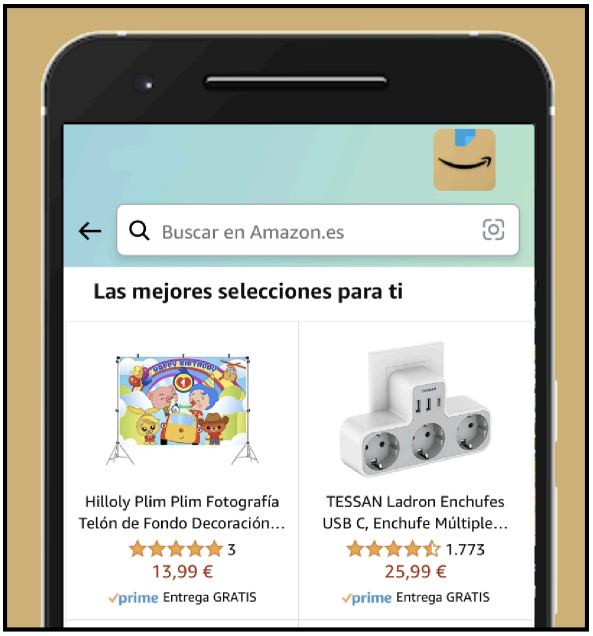

Ilustración 4: Demostración de recomendación en la aplicación de Amazon.

1.1.4. Sistemas de filtrado colaborativo

Estos sistemas de recomendación surgieron con el objetivo de solventar algunos problemas presentados por los modelos antes mencionados (colaborativo y basado en contenido) que no son aplicables por problemas de datos o de tiempo.

Además de los sistemas basados en conocimiento se presentan como alternativa para resolver estas situaciones, los sistemas híbridos que combinan diferentes técnicas de funcionamiento de las anteriores.

Al combinar varias técnicas, los sistemas híbridos de recomendación pueden superar estas limitaciones y proporcionar recomendaciones más precisas y relevantes para el usuario.

Por ejemplo, un sistema híbrido de recomendación podría utilizar técnicas de filtrado colaborativo para recomendar productos en función de las preferencias de los usuarios similares, así como técnicas de filtrado basado en contenido para recomendar productos similares a los que el usuario ha mostrado interés previamente.

Spotify es un buen ejemplo de sistema de recomendación híbrido, que incluso tiene hasta nombre propio, BaRT (Bandits for Recommendations as Treatments).

No obstante, Netflix es el mejor ejemplo de sistema de recomendación híbrido. Utiliza un sistema basado en contenido para descubrir nuevos contenidos similares a los que se hayan disfrutado previamente, un sistema de filtrado colaborativo para aprovechar el conocimiento colectivo de los usuarios y proporcionar recomendaciones basadas en gustos similares, y además utiliza un tercer sistema de recomendación basado en conocimiento. Netflix combina los tres sistemas mencionados en los anteriores apartados. Si nos fijamos en la siguiente ilustración comprobamos como Netflix te propone una serie de películas o series que considera que te van a gustar, aportándole un porcentaje de coincidencia a cada recomendación.

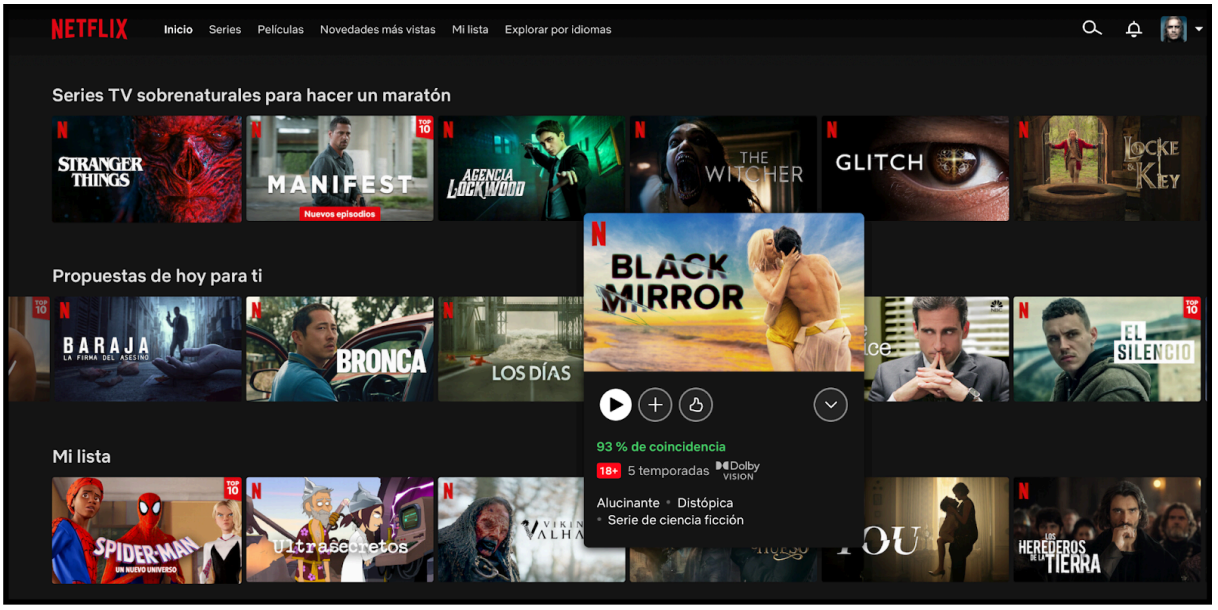

Ilustración 5: Demostración de recomendación en la aplicación de Netflix.

1.2. Objetivos

1.2.1. Objetivo Principal

Como objetivo principal este Trabajo Fin de Máster titulado “MySelflix: Plataforma móvil en la nube para la recomendación de películas”, pretende crear un sistema capaz de hacer recomendaciones a los usuarios registrados en el sistema, sobre las distintas películas que puedan existir en las plataformas actuales de streaming, recomendando independientemente de la plataforma y sin restricciones como sí ocurre en cada una de ellas (solo recomiendan sobre su propio contenido).

Mediante esta herramienta el usuario visualizará recomendaciones de películas que aún no ha visto, y encontrará las recomendaciones principales para él, tanto ordenadas por porcentaje de similitud como segmentadas en diferentes categorías, para que de esta manera pueda encontrar sugerencias dependiendo del tipo de categoría que más le apetezca ver.

Por último, el usuario podrá seguir a otros usuario a modo de red social y podrá ver las recomendaciones que ofrecen estos usuarios.

1.2.2. Objetivos Específicos

Los objetivos específicos de este Trabajo Fin de Máster son los siguientes:

• **O1**. Diseñar e implementar un sistema de recomendación

• **O2**. Crear lógica de usuarios, a modo de autenticación y datos relacionados con el usuario.

• **O3**. Implementar módulo de valoración de películas para crear o mejorar el perfil del usuario.

• **O4**. Implementar un buscador de películas.

• **O5**. Desarrollar lógica para el seguimiento de películas que se desean ver, a modo de lista.

• **O6**. Internacionalización de la aplicación.

• **O7**. Implementar funcionalidad que permita a los usuarios visualizar las plataformas de streaming en las que una película está disponible.

• **O8**. Estudiar la usabilidad de la aplicación realizando pruebas de uso con distintos usuarios.

1.3. Estado del Arte

Los sistemas de recomendación han estado en desarrollo y uso durante varias décadas. Uno de los primeros sistemas de recomendación fue desarrollado en 1992 por John Liedtke y Paul Resnick. Llamado Tapestry, consistía en un revolucionario sistema de correo y repositorio, que permitía buscar en función del contenido del documento y las reacciones registradas por otros usuarios.

Sin embargo, fue en la década de 2000 cuando comenzaron a popularizarse con la creación de plataformas como Amazon o Netflix, que utilizan sistemas de recomendación para personalizar la experiencia del usuario.

Actualmente, la propuesta de sistema de recomendación de películas que este TFM presenta, se puede encontrar ya implementada en otras aplicaciones que se irán detallando a continuación, no obstante cada una tiene sus características y ninguna de ellas tiene un enfoque de tipo red social, en la cual se puedan seguir a otros usuarios y ver las propias recomendaciones que ofrecen.

En primer lugar podríamos destacar a Netflix, que es la pionera en este campo de las recomendaciones de películas y tiene uno de los sistemas más avanzados y sofisticados del mercado. Dispone de un sistema híbrido que combina técnicas de filtrado colaborativo, basado en contenido y basado en conocimiento. Se apoya en el análisis de las películas y series que el usuario ha visto anteriormente, así como en la valoración y el tiempo dedicado a cada título. La plataforma también utiliza información de otros usuarios con patrones de visualización y valoración similares para hacer recomendaciones personalizadas.

Otras plataformas de streaming como Disney+ y HBO Max, también tienen sus propios sistemas de recomendación para las películas y series que ofrecen de contenido. A continuación se muestran dos ilustraciones que demuestran estas recomendaciones para las plataformas.

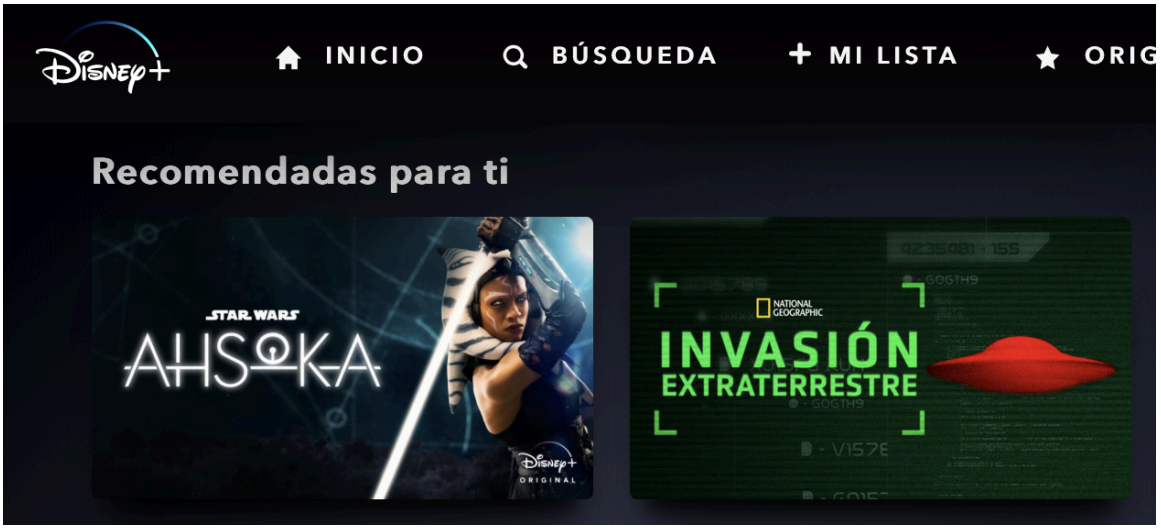

Ilustración 6: Demostración de recomendación en la aplicación de Disney+.

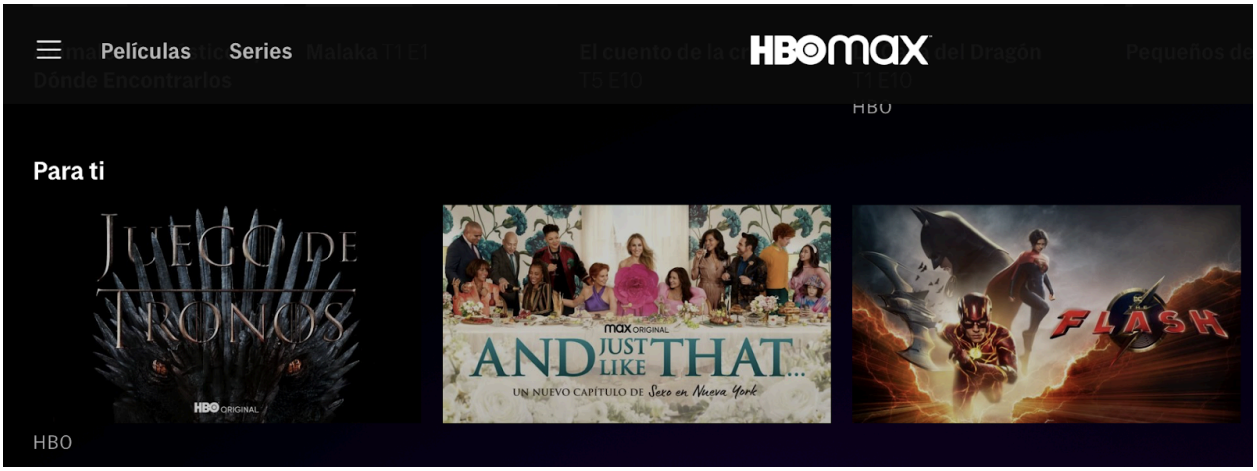

Ilustración 7: Demostración de recomendación en la aplicación de HBOmax.

Pero no solo estas plataformas nos ofrecen recomendaciones, también existen webs especializadas en las que no podemos visualizar este contenido pero sí nos ofrece sugerencias de películas y series. Es el caso de IMDB, una base de datos en línea que contiene información sobre películas, programas de televisión, actores, actrices, directores y otros profesionales de la industria del entretenimiento. Fue fundada en 1990 y adquirida por Amazon en 1998.

IMDb es una de las bases de datos más completas y populares en el mundo del cine y la televisión. Contiene información sobre películas y programas de televisión de todo el mundo, así como sobre sus actores, actrices y otros profesionales. Los usuarios pueden crear una cuenta gratuita, calificar y revisar las películas y programas de televisión que han visto. IMDb también proporciona información sobre las próximas películas y programas de televisión, así como noticias y eventos en la industria del entretenimiento.

Además IMDb dispone de un sistema de recomendación de películas basado en contenido que utiliza información sobre las características de las películas, como el género, el director, los actores principales y la trama, para hacer recomendaciones. El sistema también tiene en cuenta las valoraciones y reseñas de los usuarios para hacer recomendaciones personalizadas.

Igualmente, también encontramos a Tastedive, que está basado en contenido como IMDB pero que además de recomendar películas, y series, recomienda programas de televisión, música, videojuegos y libros.

1.4. Fundamentos Teóricos

En este apartado se detallarán los fundamentos teóricos del sistema de recomendación que se ha decidido implementar. Cabe mencionar que la decisión y motivos del método escogido se explicará de manera más extensa en el apartado de análisis. Mientras tanto en esta sección se explicarán los fundamentos del tipo de sistema de recomendación elegido, el cual se implementará mediante la técnica de filtrado colaborativo basado en factorización de matrices.

Las dos áreas principales del filtrado colaborativo son los métodos de vecindad y los modelos de factores latentes.

Los métodos de vecindad se centran en el cálculo de las relaciones entre ítems o, alternativamente, entre usuarios. El enfoque orientado a los artículos evalúa la preferencia de un usuario por un artículo basándose en las valoraciones de los artículos "vecinos" del mismo usuario. Los productos vecinos de un producto son otros productos que tienden a obtener valoraciones similares cuando son valorados por el mismo usuario.

Los modelos de factores latentes son un enfoque alternativo que trata de explicar las valoraciones caracterizando tanto los ítems como los usuarios en alrededor de 20 a 100 factores inferidos a partir de los patrones de valoración. En el caso de las películas, los factores descubiertos pueden medir dimensiones obvias como la comedia frente al drama, la cantidad de acción o la orientación a los niños, incluso dimensiones menos definidas como la profundidad del desarrollo de los personajes o hasta dimensiones completamente ininteligibles. Para los usuarios, cada factor mide cuánto les gustan las películas que puntúan alto en el factor correspondiente [5].

Algunas de las realizaciones más exitosas de los modelos de factores latentes se basan en la factorización matricial. Los modelos de factorización matricial asignan tanto los usuarios como los ítems a un espacio factorial latente de dimensionalidad k, de modo que las interacciones usuario-ítem se modelan como productos internos

en ese espacio. En consecuencia, cada artículo i se asocia a un vector $q_i \in R^k$ y a cada usuario u se le asocia un vector $p_u \in R^k$.

Para un determinado artículo i, los elementos de q_i miden el grado en que el artículo posee esos factores, positivo o negativo. Para un determinado usuario u, los elementos de p_u miden el grado de interés que el usuario tiene en los artículos que son altos en los factores correspondientes, de nuevo, positivo o negativo.

El producto punto resultante, $p_u \cdot q_i^T$, refleja la interacción entre el usuario u y el ítem i: el interés general del usuario por las características del ítem. Esto se aproxima a la valoración del usuario u del artículo i, que se denota por r_{ui}, lo que lleva a la estimación.

$$r_{ui} = p_u \cdot q_i^T$$

Este modelo está estrechamente relacionado con la descomposición de valores singulares (SVD) para descomponer la matriz de calificaciones en dos matrices más pequeñas.

La SVD es una técnica de álgebra lineal que descompone una matriz en tres componentes: una matriz de afinidad del usuario U_k ($m \times k$), una matriz de valores singulares Σ ($k \times k$) y una matriz de relevancia del ítem $V_k^{'}$ ($k \times n$) [5].

$$R = P\Sigma Q^T$$

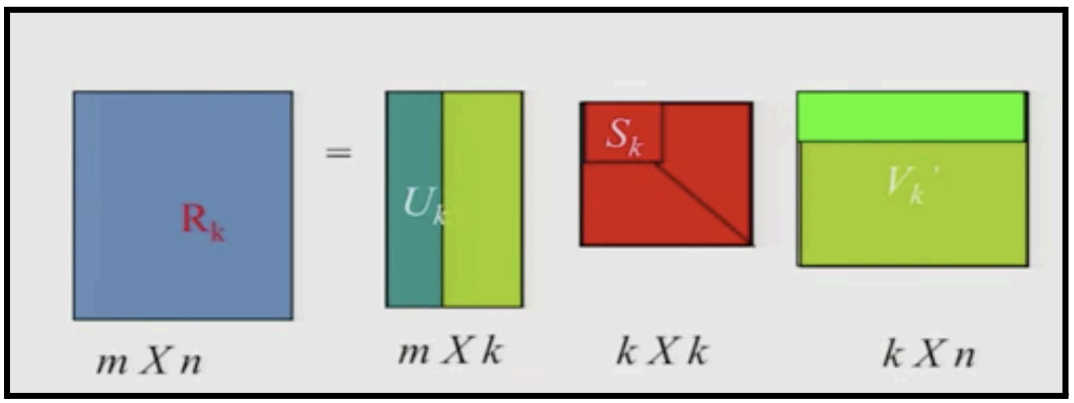

Ilustración 8: Técnica de descomposición de valores singulares.

En el contexto de los sistemas de recomendación, la matriz original (R_k) representa las calificaciones de los usuarios para los elementos. La matriz se

descompone en matrices más pequeñas: la matriz de usuarios y características (U_k) y la matriz de elementos y características (V_k'). La matriz de valores singulares (Σ) representa la importancia de cada característica para explicar las calificaciones de los usuarios [5].

La SVD convencional es indefinida cuando el conocimiento sobre la matriz es incompleto. Además, si sólo se tienen en cuenta las relativamente pocas entradas conocidas, se corre el riesgo de un ajuste excesivo.

Por ello, trabajos más recientes sugieren modelar directamente sólo las valoraciones observadas, evitando el sobreajuste mediante un modelo regularizado. Para aprender los vectores factoriales (p_u y q_i), el sistema minimiza el error cuadrático regularizado en el conjunto de valoraciones conocidas:

$$\min_{q^*,p^*} \sum_{(u,i)\in K} (r_{ui} - q_i^T p_u)^2 + \lambda(q_i^2 + p_u^2)$$

El sistema aprende el modelo ajustándose a las valoraciones observadas anteriormente. Sin embargo, el objetivo es generalizar esas valoraciones previas de forma que predigan valoraciones futuras desconocidas. Así, el sistema debe evitar el sobreajuste de los datos observados regularizando los parámetros aprendidos, cuyas magnitudes se penalizan. La constante λ controla el grado de regularización y suele determinarse mediante validación cruzada.

Para el entrenamiento del algoritmo se utilizará el método de descenso por gradiente estocástico. Para cada caso de entrenamiento, el sistema predice r_{ui} y calcula el error de predicción asociado:

$$e_{ui} = r_{ui} - p_u \cdot q_i^T$$

A continuación, modifica los parámetros en una magnitud proporcional a γ en la dirección opuesta al gradiente, lo que da como resultado:

$$q_i \leftarrow q_i + \gamma \cdot (e_{ui} \cdot p_u - \lambda \cdot q_i)$$
$$p_u \leftarrow p_u + \gamma \cdot (e_{ui} \cdot q_i - \lambda \cdot p_u)$$

Por último, y para concluir con los fundamentos, habría que definir el concepto de bias. Gran parte de la variación observada en los valores de calificación se debe a efectos asociados a los usuarios o a los ítems, conocidos como sesgos, independientes de cualquier interacción. Al fin y al cabo, algunos productos se

perciben como mejores (o peores) que otros. Por lo tanto, no sería prudente explicar el valor total de la calificación mediante una interacción de la forma $p_u \cdot q_i^T$.

En su lugar, el sistema trata de identificar la parte de estos valores que los sesgos individuales del usuario o del artículo pueden explicar, sometiendo sólo la parte de los datos correspondiente a la verdadera interacción al modelado factorial [5], con lo cual la fórmula será:

$$b_{ui} = \mu + b_u + b_i$$

Y por lo tanto, la ecuación final quedará como:

$$r_{ui} = \mu + b_u + b_i + p_u \cdot q_i^T$$

1.5. Definiciones y abreviaturas

- **TIC**: Abreviatura de Tecnologías de la Información y las Comunicaciones.
- **SR**: Sistema de Recomendación.
- **SVD**. Descomposición de valores singulares
- **XP**. Extreme Programming.
- **SCRUM**: Metodología ágil basada en Sprints.
- **TDD**: Metodología ágil Desarrollo Guiado por Pruebas.
- **SP**. Story Point. Unidad de medida para estimaciones en Scrum.
- **Git**. Software de control de versiones.
- **BaRT**. Bandits for Recommendations as Treatments.
- **RMSE.** Error cuadrático medio.
- **MAE.** Error absoluto medio.
- **DCG.** Ganancia acumulada con descuento.
- **CRUD.** Acrónimo de las operaciones Create (Crear), Read (Leer), Update (Actualizar) y Delete (Borrar).
- **REST**. Transferencia de Estado Representacional

2. ANÁLISIS

2.1. Metodología

En esta sección se muestra la metodología de desarrollo de software empleada para este proyecto, por eso a continuación se hará en un primer lugar mención a diferentes metodologías y posteriormente la justificación de la escogida, ya que dependiendo del producto software a desarrollar se adaptará mejor o peor cada metodología.

Las metodologías en general se clasifican según su enfoque y características esenciales, por lo que a la hora de analizar el tipo de metodología a escoger distinguiremos entre metodologías tradicionales y ágiles agrupando así estas de una forma más comprensible que permitan descartar un amplio número de metodologías de un golpe.

En la siguiente tabla, observamos las distintas ventajas e inconvenientes que tendrá cada una y que influirá en la toma de decisiones.

Tradicionales		**Ágiles**	
Ventajas	**Inconvenientes**	**Ventajas**	**Inconvenientes**
Objetivos claros	Resultados más lentos	Extremadamente flexible	La comunicación debe ser clara y exacta
Adecuado para proyectos más pequeños	No hay lugar para cambios	Las soluciones se elaboran de una manera rápida	Puede no funcionar si el cliente no sabe lo que quiere
Sin necesidad de equipos especializados o recursos	Costos elevados si el proyecto debe reiniciarse	Los tiempos de respuesta y resultados son más rápidos e utilizables	No es adecuada para proyectos pequeños
		Satisface al cliente	Costos y precios variables

Tabla 1. Ventajas e inconvenientes de las metodologías tradicionales y ágiles [6].

Ambas persiguen el mismo objetivo, la creación de un producto software, pero debido a las características del proyecto, el cual estará afectado por cambios se decide la metodología ágil para trabajar.

Las metodologías ágiles surgen como una alternativa a las tradicionales y están basadas en el desarrollo iterativo que se centra más en capturar mejor los requisitos cambiantes y la gestión de riesgos, rompiendo el proyecto en iteraciones de diferente longitud, cada una de ellas generando un producto completo y entregable e incremental donde un producto se construye bloque a bloque durante todo el ciclo de vida de desarrollo del producto [6].

Para el presente proyecto se ha elegido una metodología ágil, principalmente escogida para proyectos de desarrollo de aplicaciones móviles como es el caso. A continuación, tiene lugar una segunda etapa de análisis en la que se distinguen algunas metodologías ágiles como XP, SCRUM y TDD.

Extreme programming (XP), se centra en las mejores prácticas para el desarrollo de software. Consta de doce prácticas: el juego de planificación, pequeñas emisiones, la metáfora, el diseño sencillo, las pruebas, la refactorización, la programación en parejas, la propiedad colectiva, integración continua, semana 40-h, los clientes en el lugar, y los estándares de codificación. Es una metodología con un conjunto de principios probados y fiables, bien establecidos como parte de la sabiduría convencional de la ingeniería de software, pero llevado a un extremo nivel de ahí el nombre de programación extrema [6].

SCRUM es un proceso adaptativo rápido y auto-organizado de desarrollo de productos. Scrum deriva del mismo término en rugby y que hace referencia a cómo se devuelve un balón que ha salido fuera del campo, al terreno de juego de una manera colectiva. Se centra en la gestión de proyectos en situaciones en las que es difícil planificar el futuro con mecanismos de control, donde los bucles de retroalimentación constituyen el elemento central. El software es desarrollado por un equipo de auto-organización en incrementos llamados Sprints, empezando por la planificación y finalizando con un comentario. Las características que deben aplicarse en el sistema se registran en un backlog. Entonces el dueño del producto decide que elementos del backlog se deben desarrollar en el Sprint siguiente. Un miembro del equipo, llamado Scrum Master, es el encargado de resolver los problemas que impiden que el equipo trabaje eficazmente. Este equipo

generalmente es pequeño, recomendable de cinco integrantes, dividiendo el equipo principal en equipos más pequeños si fuera necesario. Puede tener problemas de requisitos variables y tecnologías con características muy diferentes. En esta situación se recomienda que el primer Sprint tenga una funcionalidad implementada con la tecnología que está dando problemas, priorizando tareas a ejecutarse, con el objetivo de subir la moral a los desarrolladores y a todo el equipo en general [6].

TDD o desarrollo orientado a las pruebas, condiciona la mentalidad de los desarrolladores, guiándonos a través del desarrollo y enfocándose en la calidad del producto final. A veces es entendido como un procedimiento para asegurar la calidad y originalmente fue pensado como una técnica para mejorar la productividad. El aumento de la calidad fue un efecto secundario, por eso hoy en día podemos encontrar muchas experiencias en las cuales se ha utilizado TDD como parte de Programación Extrema. Es más difícil encontrar experiencias en las cuales se documente la utilización de TDD como metodología aislada, normalmente aparece siempre complementando a otra metodología [6].

Finalmente se opta por utilizar para este proyecto una metodología ágil iterativa qué se basará en Scrum. A pesar de qué solo exista un desarrollador en el proyecto, se ha escogido para ayudar a mantener un enfoque claro en los objetivos, priorizar y enfocar en las tareas que aportan el mayor valor para tu proyecto, adaptarse a los cambios y desafíos que puedan surgir durante el desarrollo, e identificar posibles obstáculos. Además se hará uso de historias de usuario o épicas, las cuales se detallan más adelante.

El modelo Scrum emplea un desarrollo iterativo e incremental, denominado Sprint para cada iteración de desarrollo. Un Sprint puede abarcar una duración de entre quince días o dos meses, aunque lo recomendable es una duración de alrededor de quince días por eso ese será el tiempo elegido para cada Sprint.

Para cada Sprint se llevarán una serie de pasos expuestos a continuación [7]:

- **Revisión de Iteraciones.** Al finalizar cada iteración se lleva a cabo una revisión con las personas implicadas en el proyecto, en este caso, desarrollador y scrum master.
- **Desarrollo incremental**. El desarrollador no trabaja con diseños o abstracciones. El desarrollo incremental implica que al final de cada iteración

se dispone de una parte del producto operativa que se puede inspeccionar y evaluar.

- **Desarrollo evolutivo.** El diseño y la arquitectura del proyecto se generarán de forma evolutiva.
- **Auto-organización.** Durante el desarrollo serán muchos factores impredecibles que surgirán por lo que se auto-organiza con margen de decisión suficiente para tomar las decisiones que se consideren oportunas.

En la Ilustración 9 se representa gráficamente este proceso, donde los Sprints durarán dos semanas y finalizarán con un entregable al final de este.

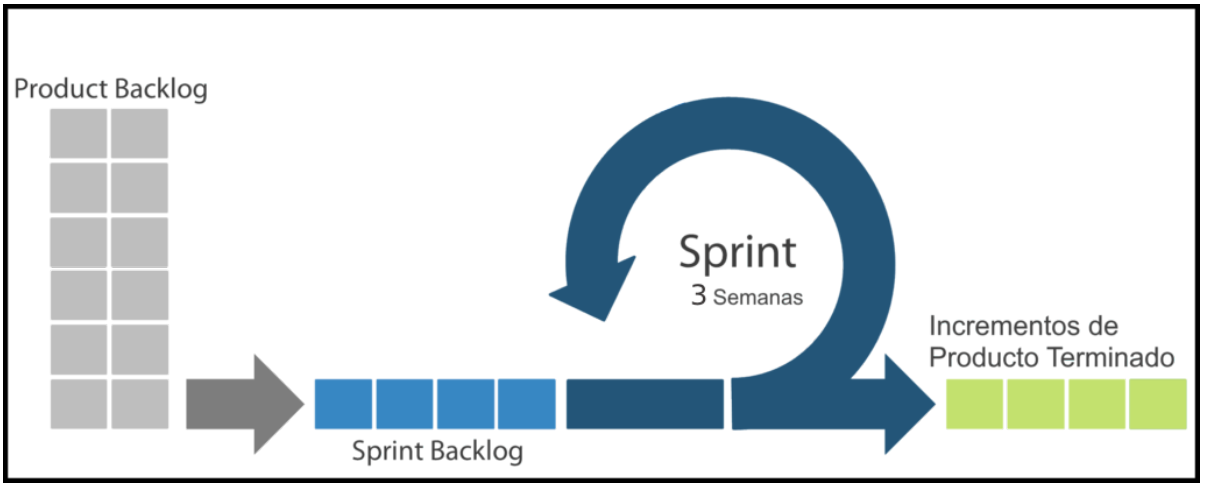

Ilustración 9: Proceso de desarrollo Scrum del proyecto [7].

Podemos observar también distintos componentes en la Ilustración 9, donde la pila del producto será una lista de requisitos de usuario que evolucionará a medida que avance el proyecto, la pila del Sprint será la lista de trabajos que se deben de realizar o requisitos que debe cumplir el sistema y por último el entregable será el producto final de cada Sprint en el que el cliente podrá ver resultados.

Cabe aclarar la designación de roles en SCRUM que se llevará a cabo en este proyecto, distinguiendo entre Master Scrum, Propietario del producto, Equipo de Scrum.

El **Master de Scrum** asegura que se siga el procedimiento, elimina los impedimentos y protege al equipo de las perturbaciones [8]. Este rol para este trabajo fin de grado corresponderá a la tutora del proyecto.

El **Propietario del producto** tiene como responsabilidades, tener una visión de lo que desea crear y transmitir esa visión al equipo de Scrum [8]. Rol llevado a cabo por el autor de este TFG.

Y para finalizar, el **Equipo de Scrum** que realiza análisis, implementaciones, diseño y pruebas [8], será asumido por una única persona y corresponderá también al autor de este TFG.

Una vez asignados los roles, el método de comunicación entre las partes se llevará a cabo mediante reuniones de revisión del Sprint donde se aclaran dudas qué hayan podido surgir.

2.2. Descripción de la solución propuesta

El sistema de recomendación estará implementado como un API REST en la nube al cual se le harán las peticiones a través de una aplicación móvil para obtener lógica de servicio, y responder así a la interacciones con el usuario.

Para conseguir cumplir con la implementación de este sistema se propone una solución que utilice el framework Spring Boot para desarrollar el lado del servidor, conocido cómo back-end, que contendrá la lógica del sistema de recomendación y un framework como Ionic para desarrollar la parte frontal, conocida como front-end, que permitirá construir una aplicación híbrida multiplataforma para usuarios de Android e iOS.

La elección del framework correspondiente al back-end se basa en la experiencia previa del creador de este TFM. Seguirá una arquitectura software definida cómo arquitectura de 3 capas, muy común en el desarrollo de microservicios con Spring Boot, ya qué ofrece beneficios significativos en términos de modularidad, escalabilidad, mantenimiento y colaboración, lo que facilita la creación y gestión de microservicios efectivos.

Por otro lado, se ha elegido Ionic para el front-end porque ofrece un conjunto de herramientas de interfaz de usuario móvil de código abierto para crear aplicaciones móviles multiplataforma modernas y nativas tanto para Android como para iOS, además de ser un framework fácil de aprender, incluso si no eres un desarrollador de JavaScript. Esto hace que sea una buena opción para los principiantes que quieren construir aplicaciones móviles. Ionic brinda la posibilidad de trabajar con diferentes frameworks de Javascript, siendo estos: Vue, React y

Angular. Para este proyecto se usará con Angular, siendo este también un framework conocido por el creador de este TFM.

Por último, la elección para el sistema de recomendación será usando la técnica de filtrado colaborativo basado en factorización de matrices mediante descenso por gradiente estocástico.

De esta manera, entrenaremos un modelo qué contenga la descomposición de las matrices tanto para los usuarios cómo para las películas y cuando se necesite obtener la recomendación únicamente haya qué hacer una única multiplicación, lo cual es muy eficiente y ágil, teniendo en cuenta lo enorme qué puede llegar a ser la matriz original, con millones de valoraciones.

Los sistemas de recomendación deben evolucionar con el tiempo a medida que cambian las preferencias de los usuarios y se agregan nuevos contenidos y la factorización de SVD permite esta evolución continua sin requerir una reestructuración significativa. Cuando se agregan nuevos usuarios o elementos, pueden ser fácilmente incorporados en este espacio latente sin requerir una revisión significativa de todo el sistema. No es necesario reentrenar todo el modelo desde cero, ya que puedes recalcular las matrices relevantes para acomodar los nuevos datos sin afectar las recomendaciones existentes [5].

Esto garantiza una mejora en los tiempos del sistema de recomendación, ya qué el entrenamiento de estos modelos para el filtrado colaborativo suele ser enorme, teniendo en cuenta además la cantidad de datos qué suelen manejar lo cual lo convierte en algo inviable a no ser qué se haga mediante este proceso qué nos ofrece unos tiempos difícilmente mejorables a la hora de ofrecer recomendaciones.

Por último, cabe mencionar que al implementar el sistema de recomendación mediante descenso por gradiente estocástico encontraremos diferentes parámetros, los cuales serán muy importantes a la hora de entrenar el modelo [9].

- ➔ **Steps**. Número de iteraciones qué el modelo repite el entrenamiento. A medida que aumentamos el número de iteraciones, el modelo tiene más oportunidades para ajustar sus parámetros y reducir el error de entrenamiento, sin embargo, es importante tener en cuenta que un número excesivo de iteraciones puede llevar al sobreajuste. Es necesario encontrar un equilibrio entre permitir que el modelo converja y evitar el sobreajuste.

➔ **K.** Número de características. Cada característica contribuye a un parámetro del modelo. Por lo tanto, un mayor número de características aumenta la dimensión del espacio de parámetros. Esto significa que el modelo debe ajustar más parámetros para representar la relación entre las características y la variable objetivo y tiene un impacto significativo en la eficiencia y el rendimiento del modelo.

➔ **Alpha**. Se refiere a la tasa de aprendizaje, cuya función principal es controlar la magnitud de los pasos que se dan al ajustar los parámetros de un modelo durante el proceso de entrenamiento, controlando la velocidad y la estabilidad del proceso de aprendizaje del modelo. Elegir la tasa de aprendizaje adecuada es esencial para garantizar una convergencia eficiente y precisa durante el entrenamiento.

➔ **Beta**. Se refiere al parámetro de regularización, que se utiliza para reducir el exceso de ajuste, lo que en última instancia nos permite usar más características.

2.3. Análisis de requisitos

Los requisitos son la base para el diseño y la implementación en la ingeniería de software. Son una lista completa de las propiedades específicas y la funcionalidad que debe tener la aplicación, expresada con todo detalle. Estos se enumeran, etiquetan y se registra su traza durante la implantación.

Esta fase es esencial en un proyecto ya que los errores más numerosos y más costosos de reparar se producen aquí, por lo que constituye una etapa fundamental para el éxito del proyecto [10].

Debido a que el proyecto se llevará a cabo bajo la metodología de Scrum y este implica cambios o evolución en los requisitos, se ha presentado un análisis inicial del que partirá el Sprint 0.

Los requisitos iniciales del sistema software que se quiere llevar a cabo se detallarán en los siguientes subapartados, separándolos en requisitos funcionales y no funcionales.

2.3.1. Requisitos Funcionales

- **RF01**. El sistema debe ser capaz de recoger los datos de las valoraciones de los usuarios sobre los artículos.
- **RF02**. El sistema debe ser capaz de crear una matriz de valoraciones que represente dichas valoraciones.
- **RF03**. El sistema debe ser capaz de realizar la descomposición de valores singulares (SVD) de la matriz de valoraciones.
- **RF04**. El sistema debe ser capaz de seleccionar el número k de valores singulares a retener en la descomposición.
- **RF05**. El sistema debe ser capaz de utilizar el producto matricial de las matrices U, Σ y VT para predecir las valoraciones de los usuarios sobre los artículos que aún no han visto.
- **RF06**. El sistema debe ser capaz de evaluar la precisión del modelo utilizando métricas como el error cuadrático medio (MSE) o la precisión top-N.
- **RF07**. El sistema debe permitir a los usuarios iniciar sesión con un nombre de usuario y una contraseña o registrarse para obtener una cuenta.
- **RF08**. El sistema debe permitir a los usuarios buscar artículos.
- **RF09**. El sistema debe proporcionar detalles sobre cada artículo, como su título, descripción y valoraciones.
- **RF10**. El sistema debe recomendar artículos a los usuarios en función de sus valoraciones y preferencias.

2.3.2. Requisitos no Funcionales

- **RNF01**. El sistema debe ser capaz de procesar los datos de las valoraciones de los usuarios en tiempo real.
- **RNF02**. El sistema debe ser capaz de proporcionar predicciones precisas.
- **RNF03**. El sistema debe ser seguro
- **RNF04**. El sistema debe garantizar una interfaz sencilla e intuitiva.
- **RNF05**. El sistema dará opción a tres idiomas, español, francés e inglés.

2.4. Historias de Usuario Iniciales

Las historias de usuario proporcionan un enfoque visualmente estructurado para que los equipos de Scrum gestionen la acumulación de productos.

El mapa de la historia visual permite la disposición de la red troncal del producto (actividades del usuario), tareas del usuario, epics e historias del usuario en una estructura de arriba hacia abajo manejable, basada en la naturaleza, la prioridad y el nivel de sofisticación de los elementos del mapa.

Ahora se explicará los diferentes componentes de estas historias para la gestión eficaz de la pila del producto [11]:

- **Epics**. El mapa de historias de 4 niveles introduce un nivel de epics, que describen un requisito funcional de gran amplitud que se desglosa en historias más pequeñas, lo cual es bueno para proyectos con mayor complejidad.
- **Actividades de usuario o tareas**. Desglose de la acumulación de un producto en elementos manejables de la pila del producto.
- **Planificación de entrega.** Las historias de los usuarios se pueden mantener en divisiones de lanzamientos, lo que refleja un calendario de entrega acordado por el equipo y las partes interesadas.

Teniendo en cuenta que la aplicación tendrá un único usuario principal (el espectador) se presentarán los epics correspondientes a este proyecto.

Recomendar película por categoría	EPIC 1
Como usuario quiero encontrar recomendaciones de películas según mi estado de ánimo pudiendo visualizar recomendaciones de distintas categorías	

Comprobar la fiabilidad de las recomendaciones	EPIC 2
Como usuario quiero que las recomendaciones que el sistema me ofrece sean de mi gusto.	

Ofrecer búsquedas de películas	EPIC 3
Como usuario quiero buscar películas concretas y visualizar información relevante sobre dichas películas, a parte del grado de coincidencia con mis gustos	

Autenticación de usuario	EPIC 4
Como usuario quiero poder mantener información relacionada con mi usuario y acceder a ella cada vez que inicie sesión	

Internacionalización	EPIC 5
Como usuario quiero poder leer cualquier texto de la aplicación en mi idioma para así comprender con total fluidez las intenciones del programa.	

A continuación se desglosan las épicas citadas anteriormente en diferentes historias de usuario que dependiendo de la complejidad se asignan con una estimación de tiempo específica en días, de igual manera se asignan los requisitos funcionales y no funcionales que llega a cubrir cada historia.

Visualizar por categorías	US 1.1
Como usuario, quiero poder visualizar las películas por categorías para centrarme en las que me interesen	
Estimación: 3 Días	

Ver recomendaciones	US 1.2
Como usuario, quiero poder ver una lista de películas que se recomiendan en función de la categoría que he seleccionado.	
Estimación: 34 Días	

Ofrecer películas mejor valoradas	US 1.3
Como usuario, quiero poder ver las películas mejor valoradas de la actualidad.	
Estimación: 4 Días	

Valorar recomendación	US 2.1
Como usuario, quiero poder valorar las películas que me han recomendado.	
Estimación: 12 Días	

Ver impacto recomendación	US 2.2
Como usuario, quiero poder ver cuántos usuarios han visto las películas que me han recomendado.	
Estimación: 2 Días	

Buscar película	US 3.1
Como espectador quiero buscar una película por su título	
Estimación: 10 Días	

Marcado de película	US 3.2
Como espectador quiero marcar como vista una película o como pendiente de ver	
Estimación: 4 Días	

Inicio de sesión	US 4.1
Como usuario quiero acceder a mi información al acceder a la aplicación	
Estimación: 8 Días	

Cierre de sesión	US 4.2
Como usuario quiero poder cerrar sesión en la aplicación y no mostrar la información relacionada conmigo una vez cerrado	
Estimación: 3 Días	

Internacionalización	US 5.1
Como usuario quiero poder leer cualquier texto de la aplicación en mi idioma para así comprender con total fluidez las intenciones del programa.	
Estimación: 5 Días	

Con lo cual las historias de usuario iniciales que dan vida a este proyecto y que forman la pila de producto del Sprint 0 quedarían como se muestra en la **Tabla 2**, teniendo en cuenta que las historias de usuario actuales podrán incrementarse, modificarse o eliminarse, existiendo una evolución continua del producto dado el tipo de proyecto y el tipo de metodología ágil escogida.

Historia de Usuario	Cod.	Tarea	Requisito	*
Seleccionar una categoría	US 1.1	Creación de menú para selección de la categoría.	RNF04	3
Ver recomendaciones	US 1.2	Obtener conexión con dataset de películas	-	34
		Crear sistema de recomendación de filtrado colaborativo	RF02 RF03 RF04 RF05 RF10 RNF02	
		Realizar test de fiabilidad	RF06	

Historia de Usuario	Cod.	Tarea	Requisito	*
Ofrecer películas mejor valoradas	US 1.3	Añadir a la interfaz recomendaciones de películas de actualidad	RF10	4
Valorar recomendación	US 2.1	Crear interfaz para valoración de películas	RF01	12
		Actualización del modelo tras valoración	RNF01	
		Crear página correspondiente a la visualización de película	RF09	
Ver impacto recomendación	US 2.2	Añadir número de visualizaciones de usuario de película	RNF04	2
Buscar película	US 3.1	Crear búsqueda de referencia	RF08	10
Marcado de película	US 3.2	Crear botón de película vista	RNF04	2
		Crear botón de película para ver	RNF04	2
Inicio de sesión	US 4.1	Crear sistema de inicio de sesión	RF07 RNF03	8
Cierre de sesión	US 4.2	Añadir funcionalidad de cierre de sesión	RNF03	4
Internacionalización	US 5.1	Implementar estructura para internacionalizar el sistema	RNF05	5

Tabla 2. Historias de usuario iniciales.

Con estas historias de usuario iniciales se pretende cubrir todos los requisitos propuestos, aunque en el caso de algunos no funcionales puede ser más difícil o ambiguo de hacerlo, cómo puede ser el caso de la **RNF04**.

2.5. Análisis de riesgos

En el proyecto también se ha llevado a cabo un análisis de los posibles riesgos, como pueden afectar, evaluación de impactos y los planes de contingencia que se emprenderán. Para el análisis y gestión de riesgos se han consultado otros ejemplos tipificados por el Consejo General de Colegios Profesionales de Ingeniería Informática o CCII. Se detalla en la Tabla 3.

ANÁLISIS DE RIESGOS				
Activo	**Amenaza**	**Probabilidad**	**Impacto**	**Riesgo**
Proyecto	Fuga de información	2	3	6
Ordenador	Corte del suministro eléctrico	2	1	1
Dispositivo	Difusión de software dañino	1	2	5
Documentación	Corrupción de información	2	3	7

Tabla 3. Análisis de riesgos.

Siendo la probabilidad un rango de posibilidad temporal de que ocurra, que estará estimada entre 1 y 3.

• **Bajo** (1). La amenaza se materializa a lo sumo una vez al año.

• **Medio** (2). La amenaza se materializa a lo sumo una vez cada mes.

• **Alto** (3). La amenaza se materializa a lo sumo una vez cada semana.

Siendo el impacto un rango de efecto, que estará estimado entre 1 y 3.

• **Bajo** (1). El daño derivado de la materialización de la amenaza no tiene consecuencias relevantes para el proyecto.

• **Medio** (2). El daño derivado de la materialización de la amenaza tiene consecuencias reseñables para el proyecto.

• **Alto** (3). El daño derivado de la materialización de la amenaza tiene consecuencias gravemente reseñables para el proyecto.

Siendo el riesgo un rango de peligro, que podrá ser mayor o menor de 4.

• **Riesgo <= 4**. El proyecto tiene un riesgo poco reseñable.

• **Riesgo > 4**. El proyecto tiene un riesgo reseñable y se debe proceder a su tratamiento.

Debido al alto riesgo de corrupción o borrado accidental de contenidos del sistema de archivos, se creará un sistema de versiones gestionado por Git que garantice la integridad del proyecto software, estando garantizado su almacenamiento en la nube y en su defecto en el repositorio local.

También supondría un riesgo una incursión de software dañino en el dispositivo móvil de las pruebas, por lo que este solo se utilizará para la comprobación de funcionalidad del código.

Por último, cabe la posibilidad de corrupción de la información para la documentación del proyecto, debido en gran medida a un excesivo uso del programa Word, por lo que se usará complementariamente google docs para tener almacenado en la nube los avances y un fichero word donde pasar a limpio esos avances.

3. PLANIFICACIÓN

3.1. Planificación Temporal

En este apartado se presenta la planificación temporal del proyecto mediante una herramienta gráfica de planificación de tareas, como es el Diagrama de Gantt.

El diagrama de Gantt es una herramienta de planificación y seguimiento de proyectos que muestra las tareas que deben completarse, el tiempo estimado para completarlas y las dependencias entre ellas. El diagrama de Gantt que se muestra en esta memoria proporciona una visión general de la planificación del proyecto.

El diagrama de Gantt muestra que el proyecto se divide en tres fases principales:

- **Fase 1**: Planificación y diseño: Esta fase incluye las tareas de recopilación de requisitos, análisis de requisitos, diseño de arquitectura y diseño de software.
- **Fase 2**: Implementación: Esta fase incluye las tareas de desarrollo de software, pruebas de software y despliegue de software.
- **Fase 3**: Mantenimiento: Esta fase incluye las tareas de resolución de problemas, mejoras.

El diagrama de Gantt es una herramienta importante para la planificación y el seguimiento de proyectos, ya que permite visualizar el progreso del proyecto e identificar cualquier posible retraso. Es una herramienta valiosa que puede ayudar a garantizar el éxito de un proyecto.

Dado que cada 3 semanas se da por finalizado cada Sprint, en esta planificación se ha estimado la necesidad de 5 Sprint y una duración de proyecto de 4 meses.

Se recomienda realizar un seguimiento del progreso del proyecto en relación con el diagrama de Gantt. Si alguna tarea se retrasa, es importante identificar la causa del retraso y tomar medidas para corregirlo. En la siguiente ilustración podemos ver el diagrama seguido para la fase de implementación.

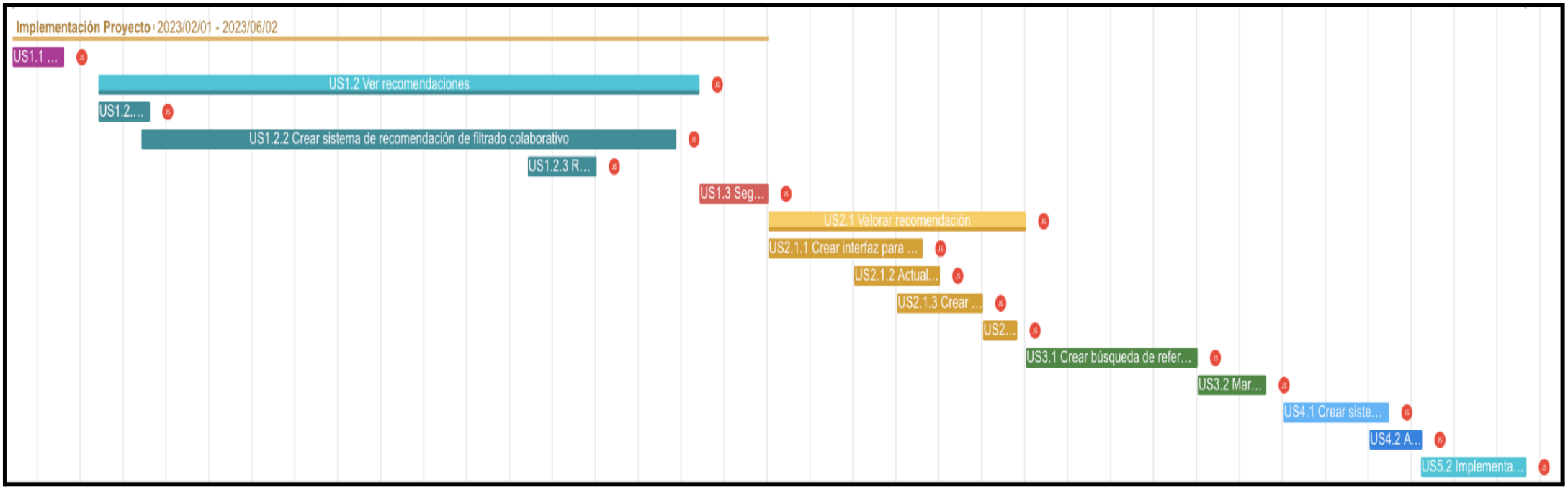

Ilustración 10. Diagrama de Gantt.

3.2. Presupuesto

A continuación se va a determinar y justificar el coste económico que supondrá la ejecución del proyecto. Esta estimación de costes se calculará en base a la planificación del calendario del trabajo fin de máster, para así garantizar la mínima tasa de error del tiempo invertido en cada una de las tareas. Para esta estimación se tendrán en cuenta aspectos de tipo hardware, software, personal y costes indirectos, con el fin de conseguir la mayor aproximación posible de costes totales.

3.2.1. Costes de Hardware

Para este proyecto se ha necesitado la adquisición de equipamiento informático. En concreto un ordenador para el desarrollo y dos dispositivos móviles que realizan las pruebas, tanto en Android como en iOS. El ordenador utilizado es un pc marca HP, modelo probook 450 G8, el cual tiene un valor de mercado de 900 € y una vida útil media de 5 años, por lo tanto supone un coste de 15 € por mes utilizado, considerando la amortización por uso.

El dispositivo Android elegido para este proyecto ha sido un móvil marca Huawei, modelo Honor 20. Ha supuesto un coste mensual de 8,125 €, ya que tiene un precio de mercado actual de 195 € y una vida útil media de 2 años debido a su obsolescencia programada.

El dispositivo iOS elegido para este proyecto ha sido un móvil iPhone 14 256 GB. Ha supuesto un coste mensual de 23,73 €, ya que tiene un precio de mercado actual de 1139 € y una vida útil media de 4 años, (más que el dispositivo Android al ser de gama alta).

Por último, teniendo en cuenta que las apis creadas se almacenarán en la nube, supondrá unos costes de servicio qué añadir a la lista. Tras consultar varios presupuestos y estimaciones, se ha elegido la opción de AWS, la cual implica un gasto de 34.82 € al mes.

Teniendo en cuenta que el tiempo de duración de este proyecto ha sido de 4 meses, el coste total en hardware ha sido de 326,70 € considerando la amortización por uso.

3.2.2. Costes de Software

La realización de esta aplicación ha supuesto el uso de distintas herramientas software. En concreto han sido las aplicaciones de Gitlab, Visual Studio Code, Visual Paradigm e IntelliJ IDEA (Ultimate Edition).

El empleo de las mismas, ha implicado varios gastos, exceptuando el entorno de integración Visual Studio Code, que no ha conllevado el desembolso de ningún efectivo, al igual que Gitlab, al ser adecuada la suscripción personal gratuita para la realización de este proyecto. En cambio, el software de Intellij ha incrementado los gastos considerablemente al ser necesaria una suscripción mensual de 59,90 € [12]. De igual manera, aunque en menor cantidad, la herramienta de planificación Visual Paradigm ha significado una cuota de 35 € en total, como resultado de un mes de suscripción [13].

Todo esto haría un total como coste de software de 379,60 €.

3.2.3. Costes de Personal

Teniendo en cuenta que el personal está compuesto por una única persona, la cual asumirá los perfiles arquitecto de sistemas, desarrollador front, desarrollador back, tester y UX), el personal constará de 1 empleado.

El salario estará basado en el XVII convenio colectivo estatal de empresas de consultoría y estudios de mercados y de la opinión pública, por lo que el coste será de 2.099,07 € al mes por personal.

Esto se debe a que dicho convenio establece que el salario base de un titulado de grado superior es de 1.961,90 € al mes (23.542,78 € anuales), más un plus para horarios laborales de ocho horas diarias de 137,17 € (1.646,24 € anuales) [14]. A este gasto habría que sumarle costes de seguridad social del trabajador que oscila entre el 32% y 38% de la base de cotización según el cargo desempeñado. Supondremos que esto implica un aumento para el proyecto de un tercio de la base de cotización del trabajador que es 671,97 €.

Finalmente el coste total del personal contando el salario base, plus de convenio y cotización a la seguridad social por parte de la empresa será de 2.771,04€ mensuales.

3.2.4. Costes Indirectos

En este apartado se explican los costes indirectos que se derivan por los gastos de teletrabajo por el miembro del proyecto. Como compensación por la prestación de los servicios profesionales en la modalidad de trabajo a distancia, el trabajador tendrá derecho a percibir una cantidad calculada de la siguiente manera:

a) Se tendrán en cuenta los gastos en los que incurra el trabajador para el abono de los suministros de electricidad, climatización e internet de manera proporcional a la jornada de trabajo que se realice en modalidad de teletrabajo.

b) Para un trabajador a jornada completa y 1800 horas de trabajo anuales, el proyecto asumirá el 20,54% del coste estimado de los suministros (1800/8760*100).

c) Según los datos y estudios publicados por las distintas organizaciones de consumidores en España:

I. El coste medio de electricidad se estima en 51,83 €/mes.

II. El coste medio de climatización se estima en 70,33 €/mes.

III. El coste medio del servicio de internet de banda ancha se estima en 51,53 €/mes.

a) Por tanto, la compensación máxima que percibirá el trabajador será de 35,67 €/mes, para el caso de que el trabajador preste sus servicios de manera exclusiva en la modalidad de teletrabajo.

b) En este caso, se estará estrictamente a lo dispuesto en el convenio colectivo o en el acuerdo de la comisión mixta paritaria al respecto de la compensación de los gastos por teletrabajo.

Por lo tanto, teniendo en cuenta que el proyecto consta de 1 persona, el coste mensual es de 35,67 €, que resultaría en un total de 142,68 € de costes indirectos.

3.2.5. Costes Totales

En la Tabla 4. Se presenta un glosario final de los costes cuya estimación final será de 11.933,14 € por 4 meses de duración del proyecto.

RECURSO	€ / MES	COSTE TOTAL
Hardware	81,67 €	326,70 €
Software	94,90 €	379,60 €
Personal	2.771,04 €	11.084,16 €
Indirectos	35,67 €	142,68 €
Total	5.755,17 €	**11.933,14 €**

Tabla 4. Estimación de costes.

4. DISEÑO

En este apartado, se trata un aspecto fundamental del sistema de recomendación, donde la visión cobra vida y se materializa en una estructura sólida y efectiva. Aquí, la atención se centra en cada pieza del rompecabezas que forma nuestra solución, con un énfasis especial en cómo los elementos individuales se ensamblan para brindar una experiencia coherente y atractiva.

Comenzaremos explorando el **Modelo Entidad/Relación**, la base sobre la cual se sustenta nuestra aplicación. Este modelo actúa como el esqueleto estructural de nuestros datos, determinando cómo se almacenan y relacionan las entidades cruciales de nuestro sistema. Adentrémonos en los detalles de cómo estos componentes se entrelazan y aseguran un flujo de datos eficiente.

Posteriormente, nos sumergimos en los **Diagramas de Clases**, que visualizan cómo los objetos interactúan y colaboran en nuestro sistema. Estos diagramas son esenciales para comprender cómo se organizan y comunican las diferentes partes de la aplicación y los microservicios.

Además veremos los **Diagramas de Secuencia**, que ilustran la forma en que los actores y los componentes interactúan en escenarios específicos. Esto nos proporciona una visión de alto nivel de cómo ocurren las transiciones y procesos dentro de nuestro sistema.

El corazón de nuestra aplicación reside en el **Diseño del Sistema de Recomendación**, donde desentrañamos la lógica detrás de las recomendaciones. Explicaremos las técnicas y algoritmos que impulsan nuestras sugerencias y cómo se adaptan a las necesidades cambiantes de nuestros usuarios.

La interfaz de usuario se convierte en el centro de atención al explorar el **Diseño UX o Storyboard**, que ayuda a visualizar ideas y conceptos. Aquí, presentamos cómo los usuarios interactúan con nuestra aplicación y cómo les proporcionamos una experiencia envolvente y atractiva.

Por último, pero no menos importante, desvelaremos la columna vertebral de comunicación de nuestra aplicación, la **API REST**. Esta interfaz define cómo nuestros microservicios interactúan y comparten datos, proporcionando una forma sencilla de acceder y manipular información.

Al comprender estos componentes clave, comenzaremos a apreciar los aspectos trascendentales de la solución y cómo se unen para crear una experiencia de usuario efectiva.

4.1. Modelos de Base de Datos

Para este punto se expondrán los modelos para representar las bases de datos del proyecto, que en este caso serán dos, ya que al seguir una arquitectura de microservicios y a pesar de qué ambos microservicios estarán conectados a la misma base de datos cada uno tendrá acceso a diferentes tablas.

El microservicio de recomendación de películas se compondrá en la parte de persistencia por una base de datos NoSQL de tipo documental, ya que el volumen de valoraciones, películas y usuarios que se manejan son enormes y una base de datos de este tipo es ideal para esa cantidad de datos.

Se ha elegido un modelo normalizado que es un modelo de datos en el que los datos se dividen en tablas relacionadas. La elección de un modelo normalizado o empotrado depende de las necesidades específicas de la aplicación. Los modelos normalizados son ideales para aplicaciones que requieren un alto rendimiento y flexibilidad, mientras que los modelos empotrados son ideales para aplicaciones que requieren una gran cantidad de datos relacionados.

Para representarlo observamos en primer lugar un diseño de base de datos relacional que contendrá como entidades, los documentos que serán almacenados en base de datos. Ahí encontraremos los esenciales, como pueden ser las películas, los usuarios y las propias valoraciones. Pero también aparecen entidades claves para el modelo como son los vectores de características en los que se descompone el modelo de predicción y que serán almacenados para las predicciones. Además también se necesitan conocer los índices tanto de películas cómo de usuarios para hacer un mapeo entre el índice que ocupan en la matriz y el id que tiene. De igual manera también se necesitará almacenar las bias resultantes del entrenamiento, debido a que como se ha mencionado en varias ocasiones, forman parte de los cálculos de predicción.

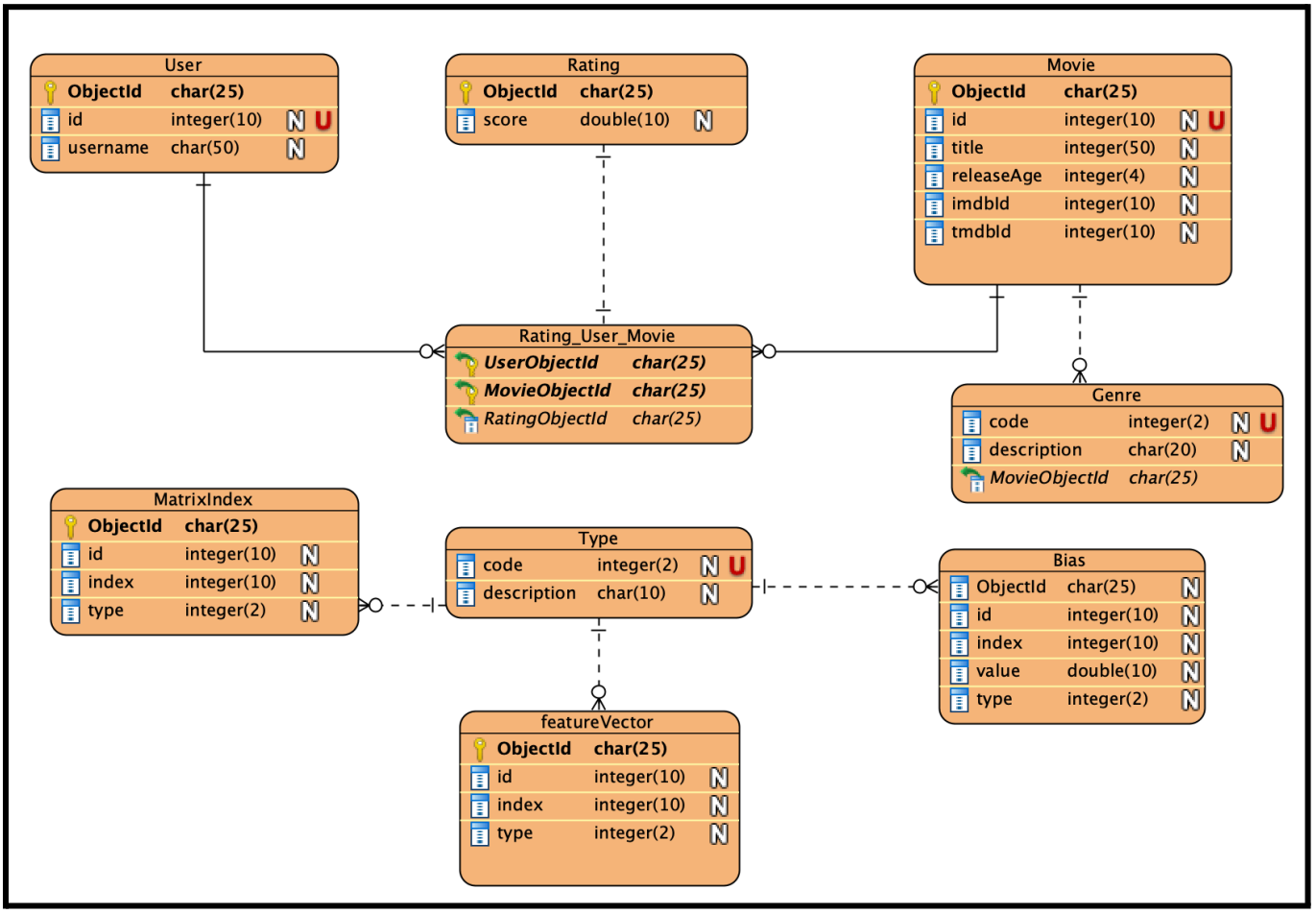

Ilustración 11: Modelo entidad relación NoSQL API - Sistema Recomendación.

Posteriormente podemos comprobar los modelos normalizados de los documentos de las colecciones. Dado que son modelos normalizados, se hará referencia a otros documentos por medio del id.

User:

```
{
    _id: <ObjectId1>,
    userId: 1,
    username: "1234xyz"
}
```

Movie:

```
{
    _id: <ObjectId1>,
    movieId: 1,
    title: "abc xyz" ,
    releaseAge: 2000,
    imdbId: "1234",
    tmdb: "1234",
    genre: [
        0: "ADVENTURE",
        1: "CHILDREN",
        2: "FANTASY"
    ]
}
```

Rating:

```
{
    _id: <ObjectId1>,
    userId: 1,
    movieId: 1,
    score: 4
}
```

Bias:

```
{
    _id: <ObjectId1>,
    id: 1,
    index: 0,
    value: 0.075,
    type: "GLOBAL"
}
```

Matrix Index:

```
{
    _id: <ObjectId1>,
    id: 1,
    index: 0
    type: "USER"
}
```

Feature Vector:

```
{
    _id: <ObjectId1>,
    id: 1,
    index: 0,
    value: [
        0: 0.1234,
        1: -0.1234
    ]
    type: "ITEM"
}
```

Para terminar con este subapartado, se aporta de igual modo el modelo para el microservicio de autenticación de usuario y registro de seguimiento de películas.

Este modelo de datos representa los datos necesarios para registrar que películas ha visto el usuario o quiere verse en algún momento a modo de seguimiento. Por otro lado encontramos los datos relacionados con el perfil de usuario y los cuales serán esenciales para determinar la autenticidad de un usuario. Por último existe también una entidad para controlar los token y permitir refrescar el token que ya ha expirado.

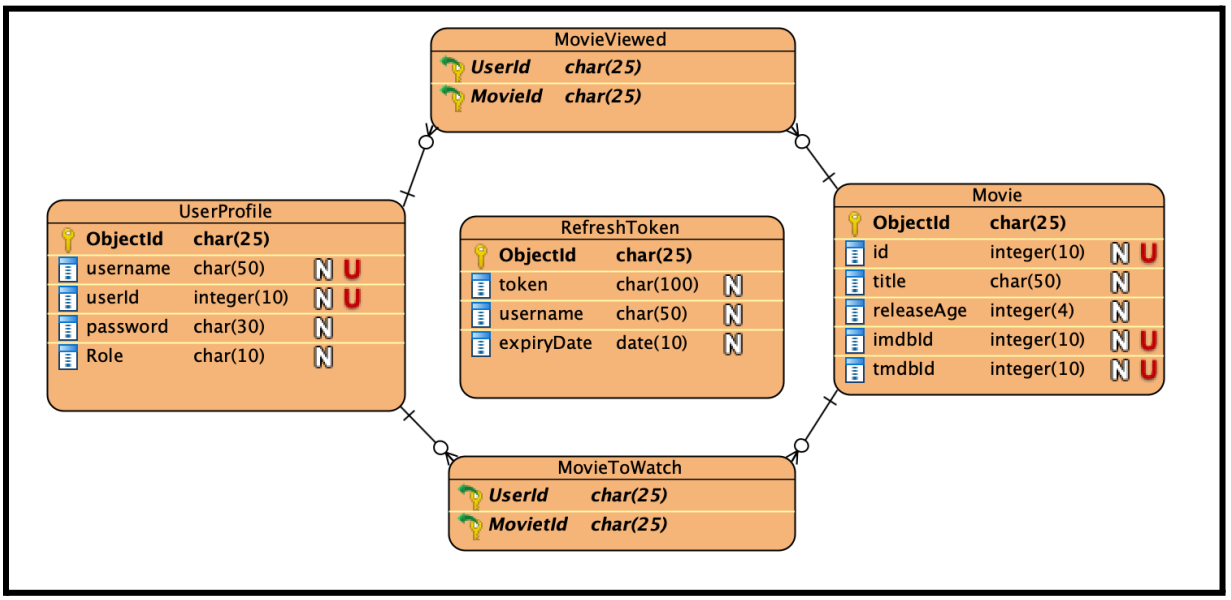

Ilustración 12: Modelo Entidad relación NoSQL API - Identidad.

A continuación ejemplo de estas entidades:

UserProfile:


```
_id: "65a8609788f5074b3c5a61f9"
userId: 611
username: "jsc00019"
password: "$2a$10$ygwsplWP2stunzjO8v.LNOm7kh.YypdpRK14a/Ps3EtWDgQsCFqLa"
role: "USER"
```


MovieViewed:
MovieToWatch:


```
_id: "65a1d822c5e93206801c4aef"
userId: 611
movieId: 5971
```


RefreshToken:


```
_id: "65a87a200733ba02bb41306a"
token: "22dc7c87-aad7-4d26-ad16-ae0bc93065d9"
expiryDate: 2024-01-18T01:18:48.358+00:00
username: "jsc00019"
```


En esta ocasión podemos comprobar como el diseño es más sencillo, ya que la funcionalidad del microservicio es más simple. Existirán dos conexiones entre las películas y los usuarios, para representar las películas vistas y las que se desean ver.

4.2. Diagramas de Clases

En este subapartado se tratarán los diagramas de clases de los distintos componentes que forman en conjunto la aplicación. En primer lugar, hablaremos de los diagramas de clases diseñados para los microservicios.

Dado que la capa de negocio muestra la lógica principal de estos, los diagramas aplicados al servidor se centrarán en los servicios de cada uno.

En primer lugar se presentan los diagramas del microservicio de recomendación, y se harán además por orden de relevancia y complejidad. En cada diagrama se pueden encontrar referencias a los demás, lo que ayuda a comprender la relación entre los posibles servicios. Spring boot implementa un patrón de diseño conocido como inyección de dependencias que tiene como objetivo tomar la responsabilidad de crear las instancias de las clases que otro objeto necesita y suministrarselo para que esta clase los pueda utilizar, por eso será muy común que los servicios llamen a otros componentes por medio de esta inyección.

Para empezar, encontramos el servicio ***PredictiveModelService*** (Ilustración 13), el cual puede considerarse como el más decisivo de este proyecto,

encargándose de la creación del modelo de recomendación donde se encuentra el algoritmo de factorización de matrices. Además también se encarga de guardar el modelo predictivo en la base de datos, para que luego pueda usarse y dar respuestas de recomendaciones en tiempo real. Y por último ofrecer predicciones a partir del usuario y la película, o a partir del modelo, ambos calculando la predicción.

A continuación, tenemos el servicio ***RecommendationService*** (Ilustración 14) que puede obtener una cantidad de recomendaciones especificada por parámetro y de forma ordenada de mayor a menor del gusto del usuario y del año de lanzamiento. Este servicio hace uso del anterior para aportar las predicciones y recomendaciones. Al final, este será el servicio predilecto del frontal para conseguir las películas a mostrar en la interfaz de recomendaciones.

Igualmente, existe otro servicio llamado ***EvaluationService*** (Ilustración 15), que como indica el propio nombre, se encarga de evaluar el modelo predictivo y en definitiva presentar las métricas que han devuelto según su parametrización. Implementa el patrón de diseño ***Strategy***, para calcular las métricas de evaluación del sistema de recomendación. El patrón permite que las métricas se puedan calcular de forma independiente del código que las utiliza. Esto hace que el código sea más flexible y adaptable a los cambios.

También tenemos los servicios ***UserService*** (Ilustración 16), ***MovieService*** (Ilustración 16) y ***RatingService*** (Ilustración 17) que principalmente actúan como CRUD de los datos utilizados en el sistema.

Por último, se observa también el servicio ***TmdbService*** junto con el de MovieService en la Ilustración 16, y sirve para poder obtener información de las películas a través de TMdB, mediante una librería que proporciona una envoltura java de la API JSON proporcionada por TMdB.

Aunque existen otros servicios en esta API, se hablará de ellos más adelante, ya que forman parte de la funcionalidad de seguridad que se cubre en mayor medida en el otro microservicio. En este se ha añadido lo necesario para que los controladores sean capaces de comprobar si un token es válido para continuar con las operaciones y en caso de que no, devuelva un mensaje de error con código de estado Http 403 Forbidden, el cual significa que el servidor deniega la acción solicitada, por la página web o servicio.

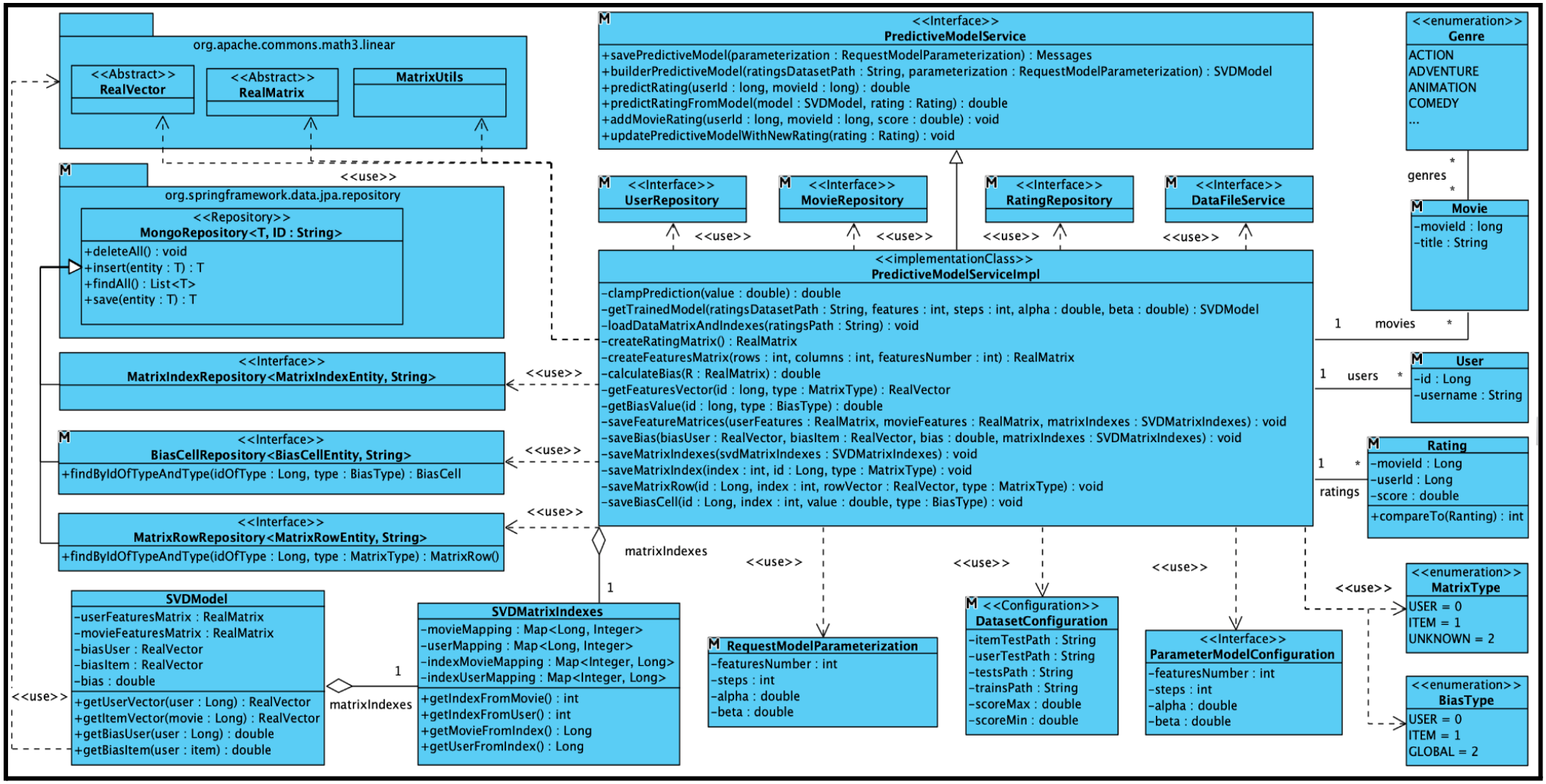

Ilustración 13: Diagrama de clase del servicio PredictiveModelService.

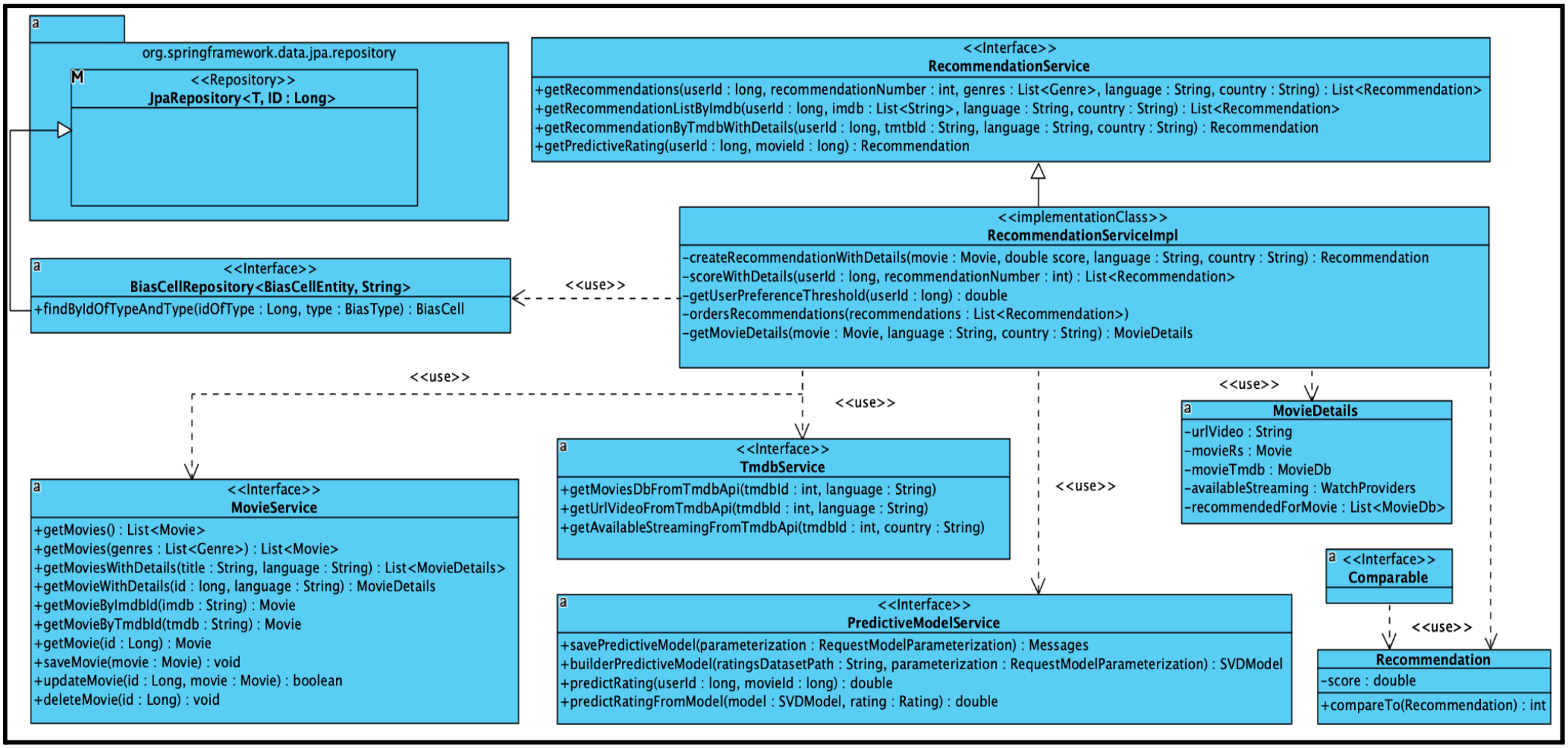

Ilustración 14: Diagrama de clase del servicio RecommendationService.

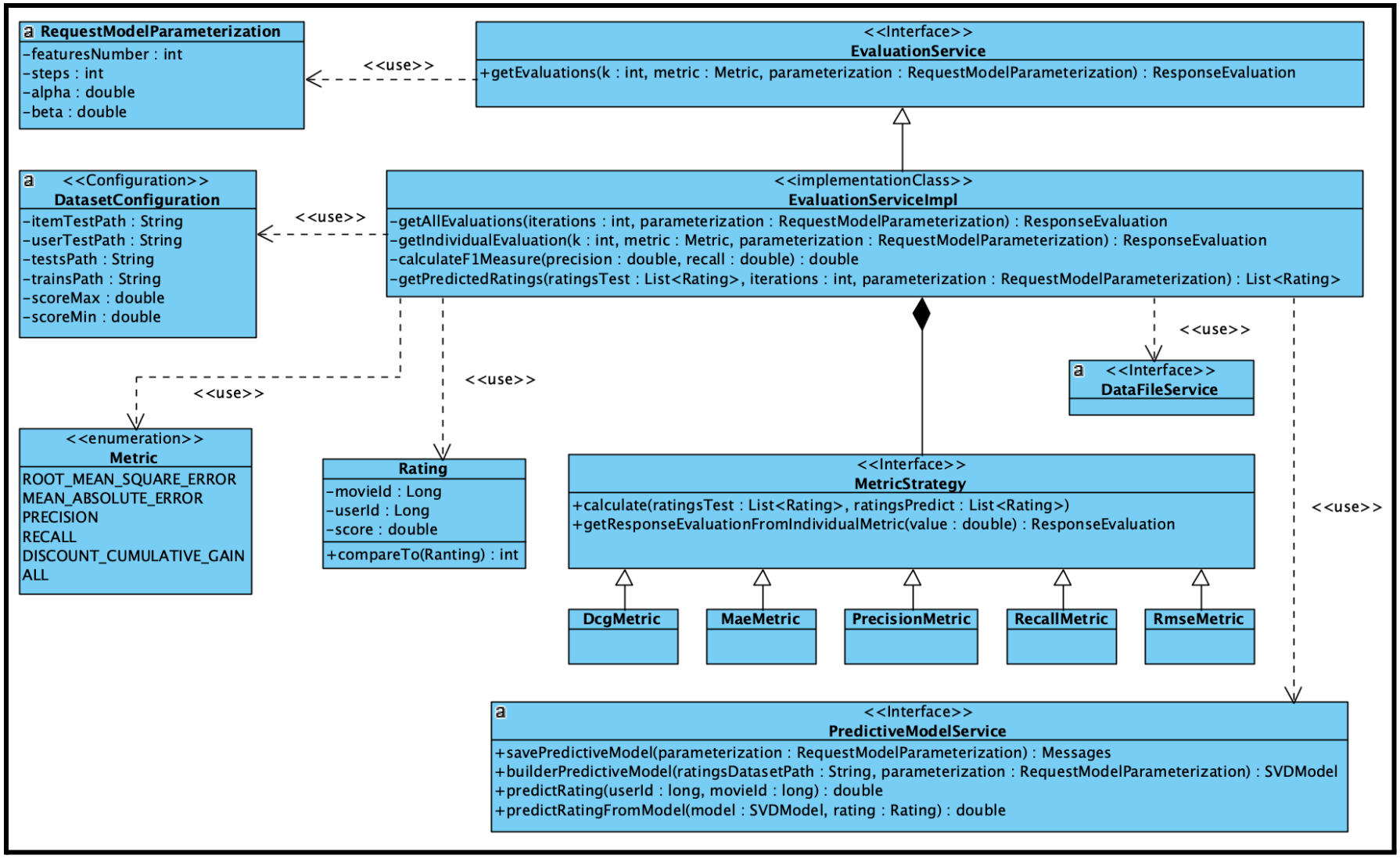

Ilustración 15: Diagrama de clase del servicio EvaluationService.

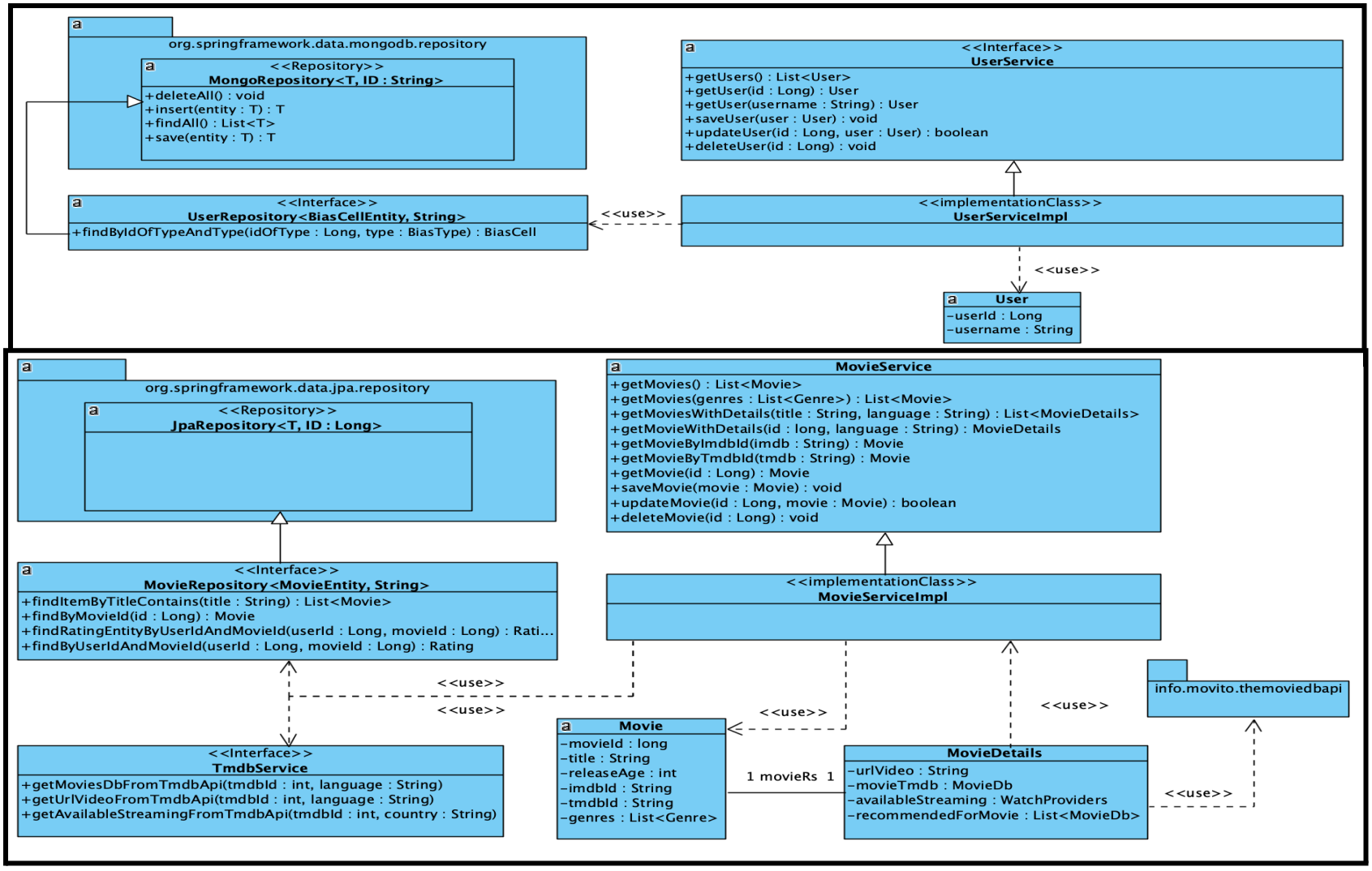

Ilustración 16: Diagrama de clase del servicio UserService, MovieService y TmdbService.

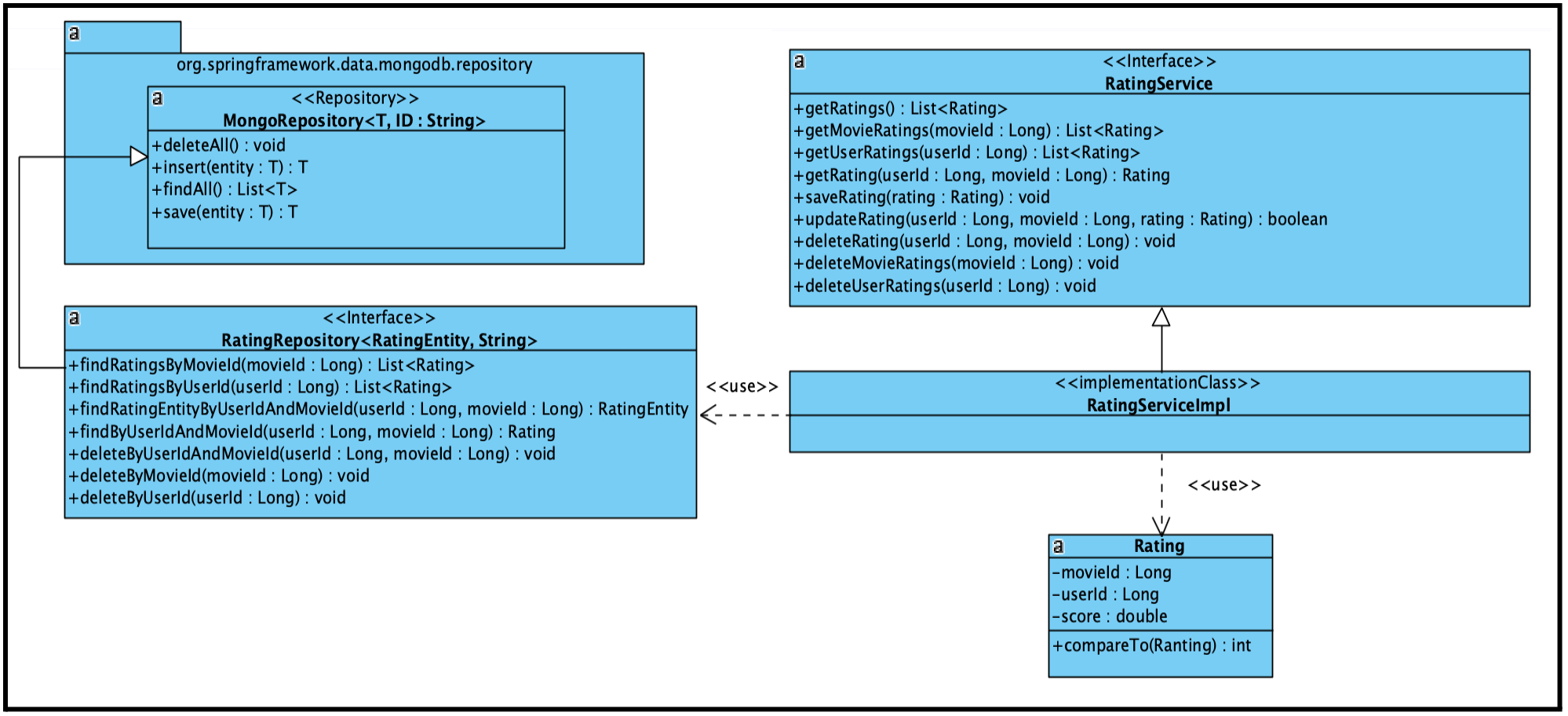

Ilustración 17: Diagrama de clase del servicio RatingService.

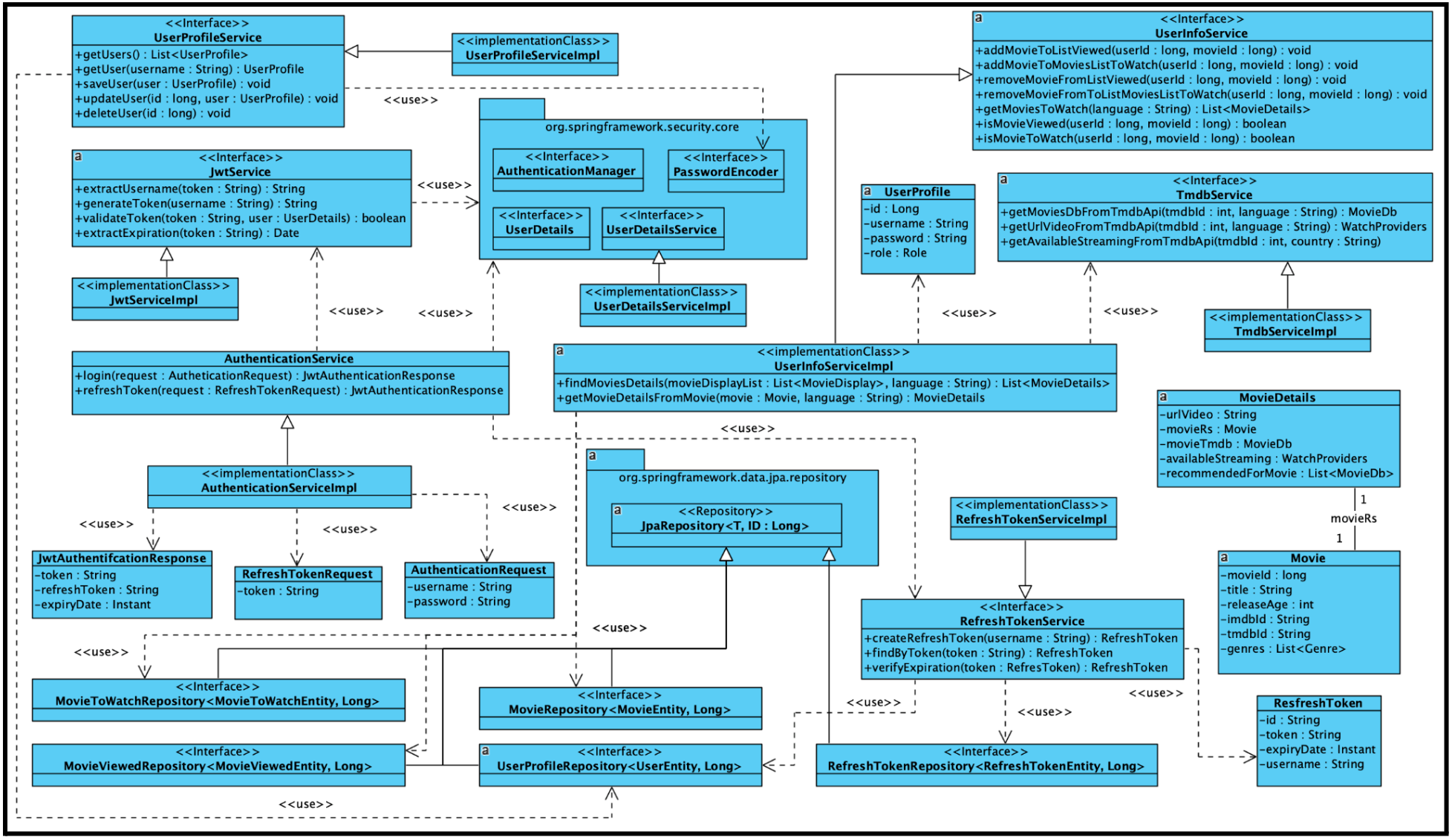

Ilustración 18: Diagrama de clases del microservicio de identidad.

En la Ilustración mostrada anteriormente podemos visualizar el diagrama de clases perteneciente al microservicio de identidad de usuario. Este microservicio se encarga entre otras cosas del registro y login de usuarios a partir de spring security, pero además gestiona otros temas relacionados con el usuario cómo pueda ser la gestión de las películas que desea ver o que ya ha visto. Este diagrama de clases es más simple que el del otro microservicio ya que conlleva menos lógica y por lo tanto puede verse por completo en una única ilustración.

En esta Ilustración 18 vemos varios servicios, como el ***UserProfileService***, que se encarga de obtener los detalles del usuario para comprobar sus permisos o ***UserInfoService*** que gestiona las películas relacionadas con el usuario. Por otro lado tenemos los servicios de ***AuthenticationService*** y ***JwtService*** que controlan todo lo relacionado con la autenticación del usuario.

Con respecto a los diagramas de clases para la aplicación frontal de Angular se puede hacer siguiendo los mismos principios que para cualquier otro tipo de aplicación. Sin embargo, hay algunas consideraciones específicas que hay que tener en cuenta.

En primer lugar, las aplicaciones frontales de Angular suelen estar compuestas por componentes. Un componente es una unidad de código que se encarga de representar una parte de la interfaz de usuario. Por lo tanto, los diagramas de clases para aplicaciones frontales de Angular suelen centrarse en los componentes.

En segundo lugar, los componentes de Angular suelen estar organizados en un árbol. El componente raíz es el que representa la aplicación completa. Los componentes secundarios son los que representan las partes individuales de la aplicación. Por lo tanto, los diagramas de clases para aplicaciones frontales de Angular suelen mostrar el árbol de componentes.

En tercer lugar, los componentes de Angular suelen comunicarse entre sí a través de servicios. Un servicio es un componente que proporciona una funcionalidad común a otros componentes. Por lo tanto, los diagramas de clases para aplicaciones frontales de Angular suelen mostrar las relaciones entre los componentes y los servicios. A continuación se muestra el diagrama correspondiente a la aplicación frontal.

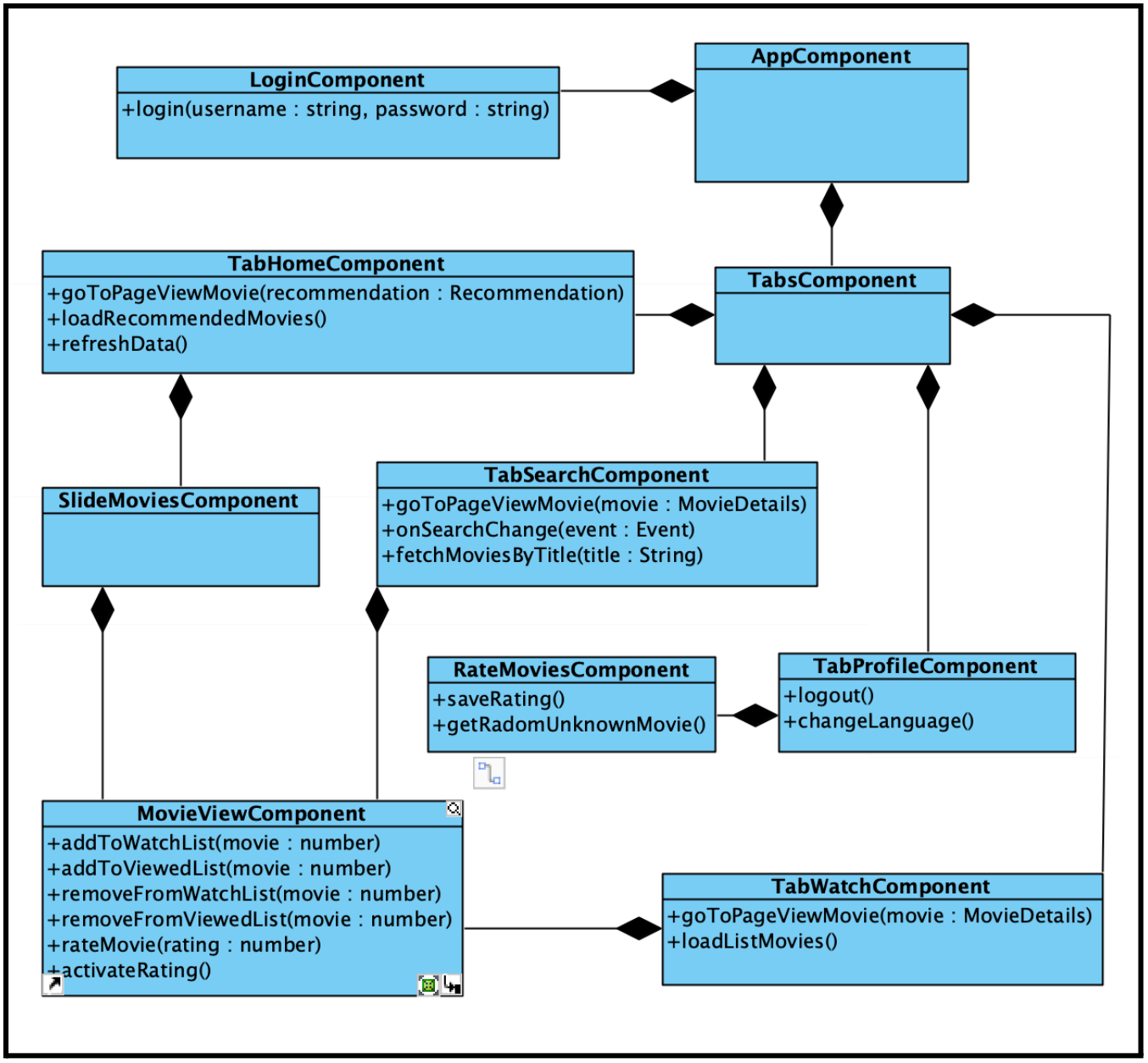

Ilustración 19: Diagrama de clase de la aplicación frontal de Ionic.

4.3. Diagramas de Secuencia

Los diagramas de secuencia son un tipo de diagrama de interacción en UML que muestran cómo los objetos de un sistema se comunican entre sí a través del tiempo. Los diagramas de secuencia se utilizan para modelar el comportamiento de un sistema, en particular la forma en que los objetos interactúan para realizar tareas [15].

Seguidamente se mostrarán los más relevantes de la aplicación. El primero en presentarse es el diagrama de secuencia de la evaluación del modelo de recomendación, pues este flujo implica a la parte más importante de la aplicación qué es la concerniente a la creación del modelo y así describe el corazón del sistema. Se corresponde con la historia de usuario US 1.2.

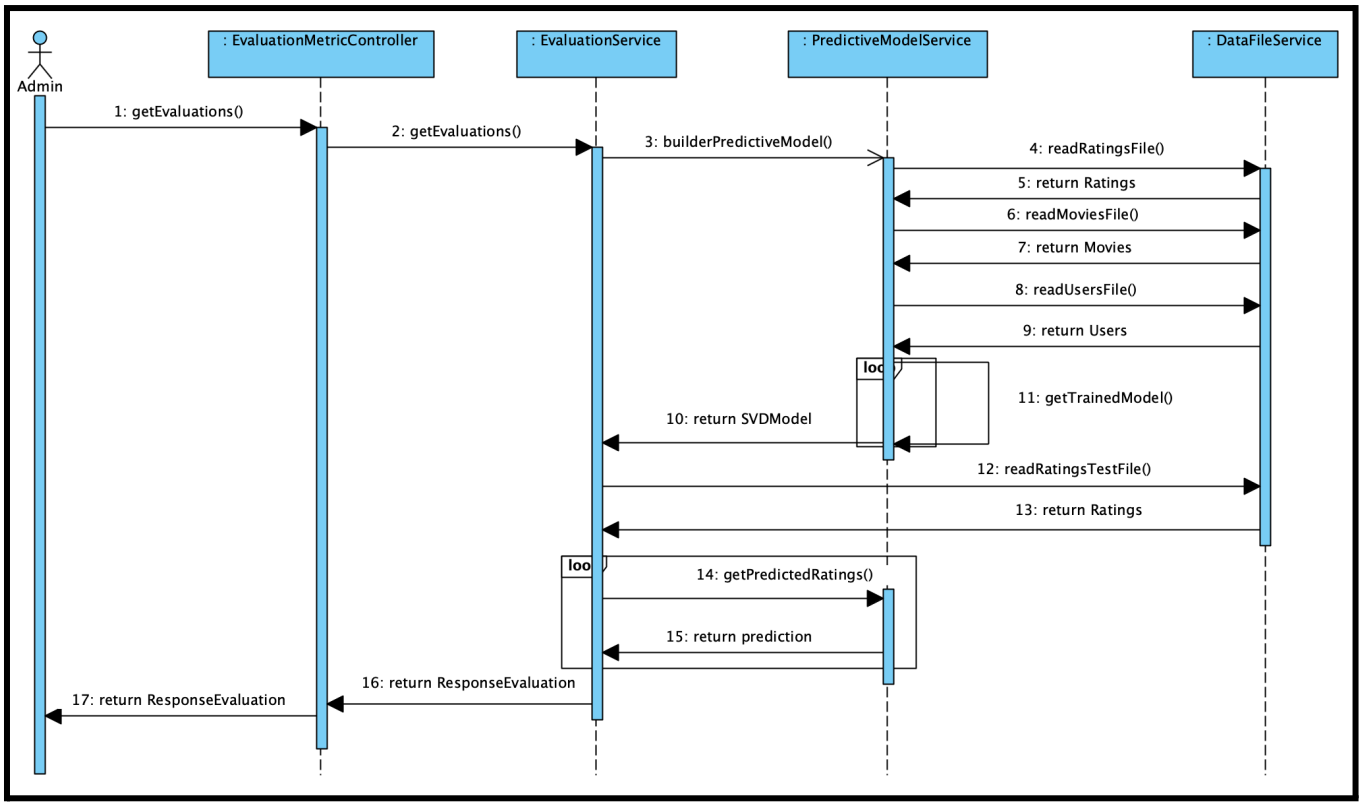

Ilustración 20: Diagrama de secuencia del servicio de evaluación de recomendaciones.

En el flujo se puede apreciar como se hace una llamada al controlador indicando que se obtengan todas las métricas. Este a su vez llama al servicio de evaluación que a su vez le indica al servicio predictivo que cree el modelo predictivo y éste devolverá un modelo ***SVDModel*** el cual se utiliza para las predicciones posteriormente. Finalmente con las predicciones se hacen los correspondientes cálculos y se envía la respuesta de las evaluaciones.

Veremos para finalizar con este subapartado otro diagrama que ejemplifica los demás diagramas de secuencia ya que utiliza interacciones similares gracias al diseño que se ha propuesto.

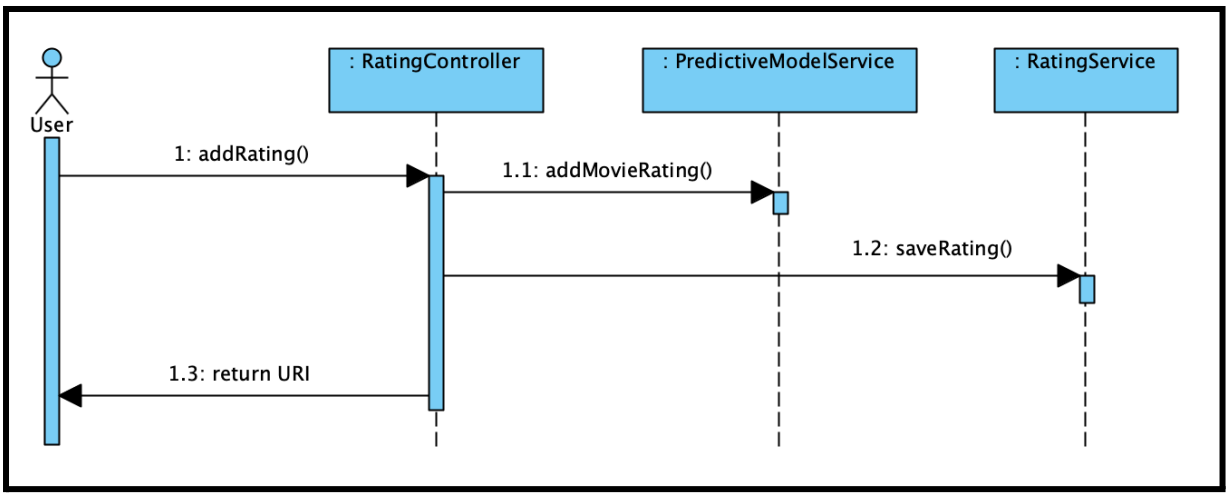

Ilustración 21: Diagrama de secuencia de añadir valoraciones de usuario.

Este diagrama se corresponde con la historia de usuario US2.1 y tiene un flujo muy sencillo en el que se le indica al servicio de predicción que actualice el modelo de predicción con la nueva valoración, donde solo afectará a las columnas de la matriz para ese usuario y esa película. Inmediatamente guarda en la base de datos la valoración

4.4. Diseño UX / Storyboards

Antes de comenzar con la maquetación de la aplicación móvil e incluso con el propio diseño de la aplicación, se ha hecho un storyboard para visualizar las ideas, poder detectar problemas tempranos y garantizar que la experiencia del usuario sea considerada y diseñada de manera efectiva desde las primeras etapas del proceso de diseño.

En primer lugar, se definen las páginas del storyboard para el registro y el inicio de sesión de usuario, representados en la ilustración 22.

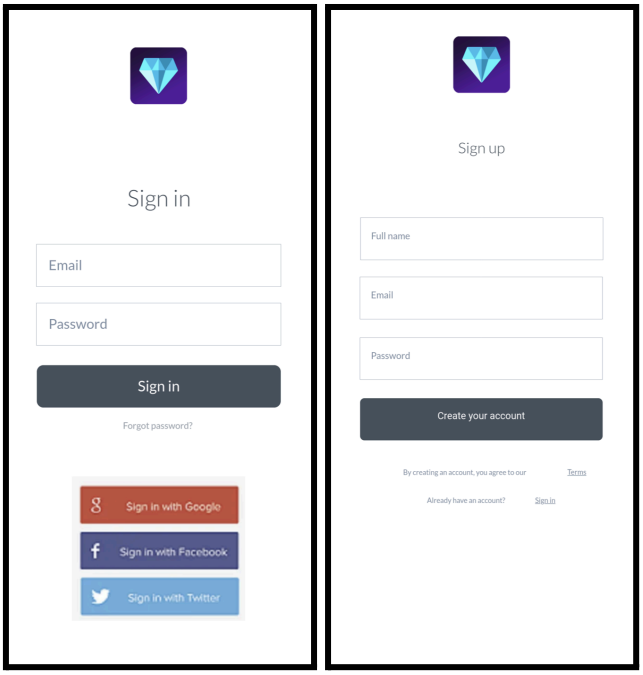

Ilustración 22. Páginas del Storyboard para el logueo de usuario.

Dado qué cada usuario necesitará un conjunto de datos de inicio qué aporten información sobre el tipo de recomendaciones qué le gustan, se representa también una página al storyboard para la valoración aleatoria de películas.

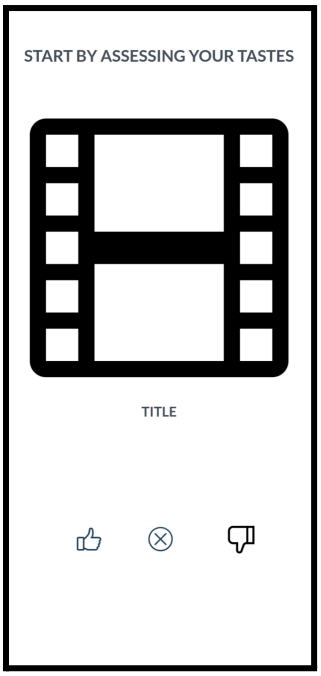

Ilustración 23. Página del Storyboard para la valoración de películas.

Cómo estructura principal de la aplicación, se definirán 4 pestañas principales:

- **Inicio**. Página inicial donde se presentan las principales recomendaciones para el usuario.
- **Búsquedas**. Otra página de búsquedas qué permite al usuario buscar directamente películas concretas y obtener información y la predicción del sistema sobre la coincidencia con sus gustos.
- **Lista**. Una página para gestionar una lista de películas para ver.
- **Configuración**. Una página para gestionar la configuración del usuario.

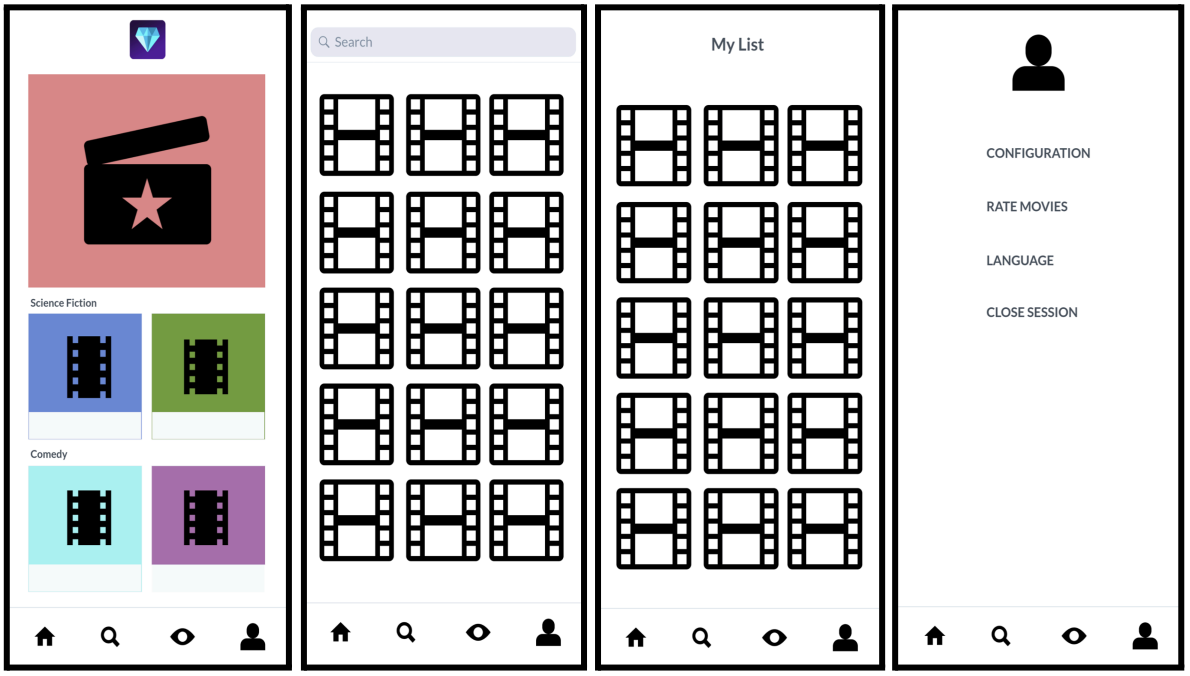

Ilustración 24. Páginas del Storyboard de las pestañas principales.

Por último, definimos una página del storyboard qué representa la visualización de una película de manera individual.

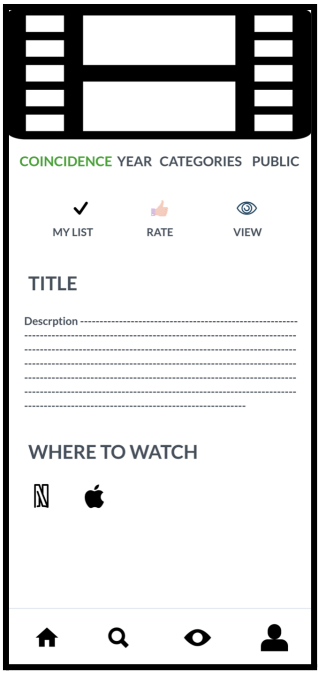

Ilustración 25. Página del Storyboard de una película.

El diseño de una experiencia de usuario efectiva no solo se limita a la estética visual, sino que también se enfoca en cómo los usuarios navegan a través de la aplicación y acceden a sus funciones. En ese sentido, se ha elaborado un mapa de navegación que ilustra el flujo seguido por el usuario a medida que interactúa con la aplicación. Este mapa de navegación proporciona una visión general de las diferentes pestañas y funciones disponibles.

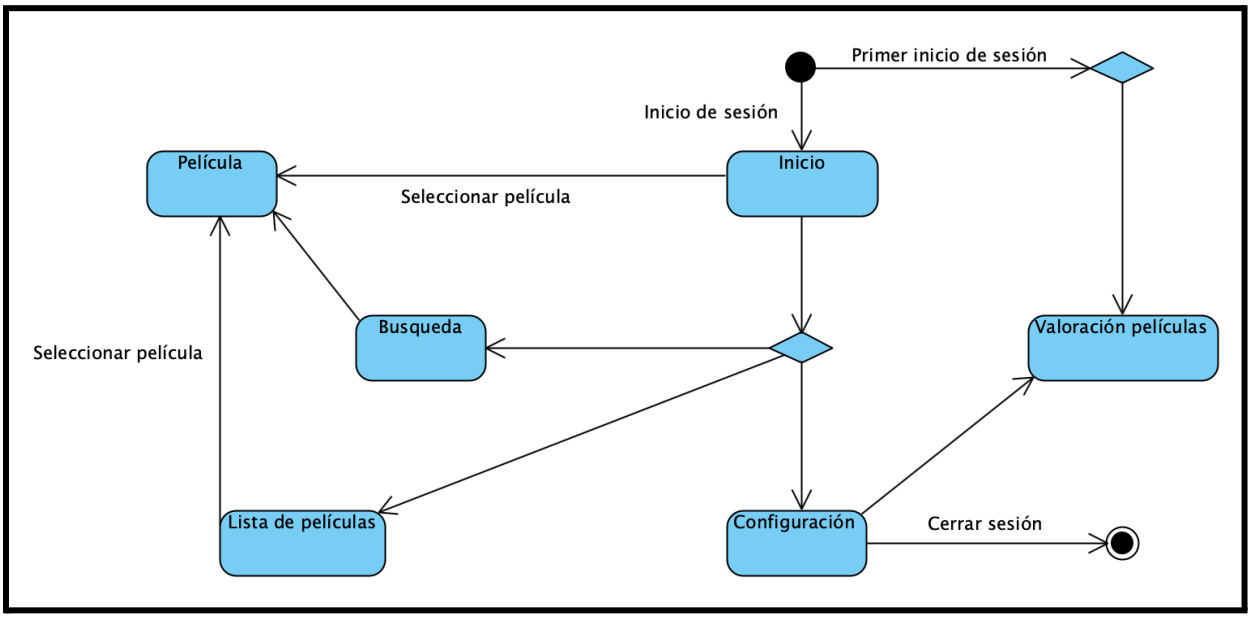

Ilustración 26. Diagrama de navegación de la aplicación.

4.5. API Rest

En este punto se muestra la comunicación entre los microservicios y el frontal mediante una interfaz de programación de aplicaciones (API) qué se ajusta a los límites de la arquitectura **REST** y permite la interacción con los servicios web de RESTful.

Las API son conjuntos de definiciones y protocolos que se utilizan para diseñar e integrar el software de las aplicaciones. Suele considerarse como el contrato entre el proveedor de información y el usuario, donde se establece el contenido que se necesita por parte del consumidor (la llamada) y el que requiere el productor (la respuesta).

REST no se considera un protocolo o estándar en sí mismo, sino más bien un conjunto de principios arquitectónicos. Los desarrolladores que crean APIs pueden aplicar estos principios de diversas maneras.

Cuando un cliente envía una solicitud a través de una API basada en REST, está solicitando una representación del estado de un recurso específico. Esta representación se transfiere al solicitante a través del protocolo HTTP y se puede presentar en varios formatos, como JSON (Notación de Objetos de JavaScript), HTML, XML, Python, PHP o texto sin formato. Aunque el nombre "JSON" sugiere un lenguaje de programación, es un formato ampliamente utilizado que es comprensible tanto para máquinas como para humanos, y no está vinculado a un lenguaje de programación en particular [16].

Para esta aplicación se utilizará el formato de JSON, ya que es un formato de datos ligero y legible para los humanos que se basa en una estructura de pares clave-valor, lo cual lo hace fácil de entender y de trabajar con él. Se ha vuelto muy popular en el desarrollo de aplicaciones web y servicios web, ya que es compatible con muchos lenguajes de programación y frameworks.

Una de las ventajas de JSON es su simplicidad y concisión. Los datos en este formato son más compactos en comparación con XML, lo que los hace ideales para transmitir grandes cantidades de información de manera eficiente. En términos de procesamiento, JSON es rápido y eficiente, algo especialmente importante en entornos de alto rendimiento, donde la velocidad de procesamiento es crucial [17].

Además de la representación del recurso, también es importante considerar otros aspectos en las solicitudes realizadas a través de una API RESTful. Los encabezados y los parámetros desempeñan un papel fundamental en los métodos HTTP utilizados en estas solicitudes. Proporcionan información clave relacionada con metadatos, autorización, identificadores uniformes de recursos (URI), almacenamiento en caché, cookies y otros elementos de la solicitud. Los encabezados se dividen en encabezados de solicitud y encabezados de respuesta, y cada uno de ellos lleva su propia información de estado y detalles de conexión en el contexto del protocolo HTTP [16].

A continuación se presentan todos los endpoints de los que se compondrá la API, tanto los parámetros de entrada cómo los datos de salida que corresponden con clases ya detalladas en los diagramas de clases del punto 4.2. En los parámetros de entrada se muestra cómo estilo tipográfico negrita los parámetros que forman parte de la url, tanto los de tipo query parameters cómo los path variables. En el caso de los parámetros que forman parte del body de la petición irán sin la tipografía negrita. No se menciona una descripción de cada endpoint, dado que el significado se puede deducir fácilmente gracias a los nombres de los recursos y métodos http asociados.

API Sistema de Recomendación				
METHOD	**ENDPOINT**	**ENTRADA**	**SALIDA**	**STATUS**
GET	api/v1/users	-	List<User>	200, 400, 403
GET	api/v1/movies	-	List<Movie>	200, 400, 403
GET	api/v1/movies/genre	genres : List<Genre>	List<Movie>	200, 400, 403, 404
GET	api/v1/movies/id/{id-movie}	**movieId** : Long	Movie	200, 400, 403, 404
GET	api/v1/movies/id/{id-movie}/details	**movieId** : Long language : String	MovieDetails	200, 400, 403, 404
GET	api/v1/movies/imdb/{id-imdb}	**imdbId** : Long	Movie	200, 400, 403, 404
GET	api/v1/movies/title/{title}	**title** : String	Movie	200, 400, 403, 404

API Sistema de Recomendación				
METHOD	**ENDPOINT**	**ENTRADA**	**SALIDA**	**STATUS**
		language: String		
POST	api/v1/movies	movieId : Long title : String releaseAge : int imdbId : String tmdbId : String genres : Genre	URI	201, 400, 403
PUT	api/v1/movies/{id-movie}	**movieId** : Long	-	200, 204, 400, 403, 404
DELETE	api/v1/movies/{id-movie}	**movieId** : Long	-	204, 403
GET	api/v1/ratings	-	List<Rating>	200, 400, 403
GET	api/v1/ratings/movies/{id-movie}	**movieId** : Long	List<Rating>	200, 400, 403, 404
GET	api/v1/ratings/users/{id-user}	**userId** : Long	List<Rating>	200, 400, 403, 404
GET	api/v1/ratings/{id-user}/{id-movie}	**userId** : Long **movieId** : Long	Rating	200, 400, 403
POST	api/v1/ratings	userId : Long movieId : Long score : Double	URI	201, 400, 403
PUT	api/v1/ratings/{id-user}/ {id-movie}	**userId** : String **movieId** : String score : Double	-	200, 204, 400, 403, 404
DELETE	api/v1/ratings/movies/{id-movie}	**movieId** : Long	-	204, 400, 403, 404
DELETE	api/v1/ratings/users/{id-user}	**userId** : Long	-	204, 400, 403, 404
DELETE	api/v1/ratings/{id-user}/{id-movie}	**userId** : Long **movieId** : Long	-	204, 400, 403, 404
GET	api/v1/recommendations/{id-user}	**userId** : Long size : int	List<Recommendation>	200, 400, 403, 404

API Sistema de Recomendación				
METHOD	**ENDPOINT**	**ENTRADA**	**SALIDA**	**STATUS**
GET	api/v1/recommendations/{id-user}/imdb	**userId** : Long language : String country : String imdbList : List<String>	Recommendation	200, 400, 403, 404
GET	api/v1/recommendations/{id-user}/tmdb/{id-tmdb}	**userId** : Long **tmdbId** : Long language : String country : String	Recommendation	200, 400, 403, 404
GET	api/v1/recommendations/{id-user}/{id-movie}	**userId** : Long **movieId** : Long	Recommendation	200, 400, 403, 404
POST	api/v1/evaluations	foldNumber : int metric : Metric parametrization : RequestModelParameterization	ResponseEvaluation	201, 400, 401, 403, 500
POST	api/v1/predictive-models	features : int steps : int alpha : Double beta : Double	Messages	201, 400, 401, 403, 500

Tabla 5. Endpoints de la API de Recomendación.

API Sistema de Perfil de Usuario				
METHOD	**ENDPOINT**	**ENTRADA**	**SALIDA**	**STATUS**
GET	api/v1/users/id/{id-user}	**userId** : Long	UserDetails	200, 400, 403, 404
GET	api/v1/users/username/{username}	**username** : String	UserDetails	200, 400, 403, 404
GET	api/v1/users/list-to-watch	language : String	List<MovieDetails>	200, 400, 403
GET	api/v1/users/id/{id-user}/list-viewed/{id-movie}	**userId** : Long **movieId** : Long	Boolean	200, 400, 403

API Sistema de Perfil de Usuario				
METHOD	**ENDPOINT**	**ENTRADA**	**SALIDA**	**STATUS**
GET	api/v1/users/id/{id-user}/list-viewed/{id-movie}	**userId** : Long **movieId** : Long	Boolean	200, 400, 403
POST	api/v1/users	userId : Long username : String	URI	201, 400, 403
PUT	api/v1/users/{username}	**username** : String email : String	-	200, 204, 400, 403, 404
DELETE	api/v1/users/{id-user}	**userId** : Long	-	204, 400, 403, 404
POST	api/v1/users//list-to-watch	display : MovieDisplayRequest	URI	201, 400, 403
DELETE	api/v1/users//list-to-watch	display : MovieDisplayRequest	-	204, 400, 403, 404
POST	api/v1/users//list-viewed	display : MovieDisplayRequest	URI	201, 400, 403
DELETE	api/v1/users//list-viewed	display : MovieDisplayRequest	-	204, 400, 403, 404
POST	api/v1/auth/signup	firstName : String lastName : String password : String email : String	Messages	201, 400
POST	api/v1/auth/signin	username : String password : String	Messages Acces Token	201, 400, 401, 403
POST	api/v1/auth/logout	Acces Token	Messages	201, 400
POST	api/v1/password-recovery	email : String	Messages	201, 400, 401, 403

API Sistema de Perfil de Usuario				
METHOD	**ENDPOINT**	**ENTRADA**	**SALIDA**	**STATUS**
POST	api/v1/password-change	currentPassword : String newPassword : String	Messages	201, 400, 401, 403

Tabla 6. Endpoints de la API de Perfil de usuario.

La elección de los códigos de estado para los endpoints de la API REST es una parte crucial del diseño, ya que proporcionan información valiosa a los clientes de la API sobre el resultado de sus solicitudes. Seguidamente, se detallan los códigos de estado elegidos, sus razones y posibles mensajes, en la siguiente tabla:

Para las solicitudes **GET**:	
Código 200 (Éxito): Este código se utiliza cuando la solicitud GET se completa con éxito y devuelve los datos solicitados.	
Código 400 (Solicitud incorrecta): Indica que la solicitud GET no se procesó debido a un error en los parámetros o la estructura de la solicitud.	
Código 404 (No encontrado): Se utiliza específicamente para solicitudes GET que obtienen objetos por un {id}. Si el objeto solicitado no existe, se devuelve este código.	
Para las solicitudes **PUT**:	

<table>
<tr><td>Código 200 (Éxito): Indica que la solicitud PUT se completó con éxito.</td><td></td></tr>
<tr><td>Código 204 (Sin contenido): Se utiliza para notificar que la solicitud PUT se procesó correctamente, pero no se ha modificado el objeto, ya sea porque no se encuentra objeto a modificar o porque la modificación es igual al valor que ya hay en base de datos.</td><td></td></tr>
<tr><td>Código 400 (Solicitud incorrecta): Se emplea cuando la solicitud PUT contiene datos incorrectos o estructura inapropiada.</td><td><pre>{
  "messageDetails": [
    {
      "code": "error.format.invalid",
      "description": "Value can't be empty or null",
      "location": "movie.id",
      "level": "ERROR"
    }
  ]
}</pre></td></tr>
<tr><td>Código 404 (No encontrado): Indica que el recurso a actualizar no se encontró.</td><td><pre>{
  "messageDetails": [
    {
      "code": "error.not.found",
      "description": "User: 111111 not found",
      "location": "error.user.not.exist",
      "level": "ERROR"
    }
  ]
}</pre></td></tr>
<tr><td colspan="2">Para las solicitudes POST:</td></tr>
<tr><td>Código 201 (Creado): Este código se usa cuando una solicitud POST crea con éxito un nuevo recurso.</td><td></td></tr>
</table>

<table>
<tr><td>Código 400 (Solicitud incorrecta): Se utiliza para notificar errores en la solicitud POST, devolviendo un mensaje.</td><td><pre>{
  "messageDetails": [
    {
      "code": "error.format.invalid",
      "description": "size is incorrect, the value should be between 1 and 5",
      "location": "requestEvaluationMetric.foldNumber",
      "level": "ERROR"
    }
  ]
}</pre></td></tr>
<tr><td colspan="2">Para las solicitudes DELETE:</td></tr>
<tr><td>Código 204 (Sin contenido): Indica que la solicitud DELETE se completó con éxito y que no hay contenido adicional para mostrar.</td><td></td></tr>
<tr><td>Código 400 (Solicitud incorrecta): Se emplea cuando la solicitud DELETE contiene datos incorrectos o estructura inapropiada.</td><td><pre>{
  "messageDetails": [
    {
      "code": "error.format.invalid",
      "description": "Value can't be empty or null",
      "location": "movie.id",
      "level": "ERROR"
    }
  ]
}</pre></td></tr>
<tr><td>Código 404 (No encontrado): Indica que el recurso a eliminar no se encontró.</td><td><pre>{
  "messageDetails": [
    {
      "code": "error.not.found",
      "description": "User: 111111 not found",
      "location": "error.user.not.exist",
      "level": "ERROR"
    }
  ]
}</pre></td></tr>
</table>

Tabla 7. Códigos de estado para los distintos endpoints.

Estos códigos de estado se seleccionaron cuidadosamente para garantizar que los clientes de la API reciban una respuesta adecuada a sus solicitudes y puedan tomar las acciones apropiadas en función de los resultados obtenidos. Por último antes de presentar los endpoints cabe mencionar la posibilidad de que el código de estado sea 500 para los endpoints encargados del modelo de recomendación, indicando algún error en el servidor producido por los datasets a partir de los cuales se crea el modelo.

Por otro lado, además de los microservicios presentados anteriormente, el frontal se comunica con otra API de un proveedor externo, tal y cómo es la API de IMDb [18]. Esta interfaz se ha incluido en el proyecto para obtener información relevante sobre películas, ya que permite el acceso a una amplia base de datos con una información precisa y detallada sobre películas, al igual que recursos cómo imágenes de las películas, incluso trailers.

Sin embargo, es importante mencionar que IMDb puede tener restricciones en cuanto al acceso a su API y la cantidad de solicitudes que puedes hacer, no obstante se ha elegido esta opción igualmente.

API IMDb				
METHOD	**ENDPOINT**	**ENTRADA**	**SALIDA**	**STATUS**
GET	{lang?}/API/Top250Movies/{apiKey}	**lang** : String **apiKey** : String	Top250Data	200
GET	{lang?}/API/MostPopularMovies/{apiKey}	**lang** : String **apiKey** : String	MostPopularData	200

Tabla 8. Endpoints de la API externa de IMDb.

Cabe mencionar que todos los endpoints están securizados con token por lo que en caso de no enviar tokens válidos en las peticiones se recibirían códigos http de estado 403 indicando que el servidor deniega la petición. Para poder obtener un token válido habrá qué loguearse, el cual es el único endpoint que no dispone de esta capa de seguridad.

5. IMPLEMENTACIÓN

La sección de implementación es un componente crucial en la memoria del proyecto, ya que es aquí donde se detallan las tecnologías y herramientas que forman parte de la aplicación. A lo largo de este capítulo, se explorarán las bases tecnológicas de nuestro proyecto, tanto de la parte del servidor (back-end) como de la parte del cliente (front-end). Además, presentaremos la arquitectura general que sustenta nuestra aplicación y discutiremos en profundidad el algoritmo de recomendación que conforma el proyecto.

Aquí, no sólo describiremos las tecnologías que hemos seleccionado, sino también explicaremos por qué las elegimos y cómo encajan en la estructura de la aplicación. En resumen, esta sección es esencial para comprender la implementación práctica de nuestra aplicación, desde la infraestructura subyacente hasta el algoritmo que brinda recomendaciones significativas a nuestros usuarios.

5.1. Tecnologías

En la implementación de nuestra aplicación, hemos aprovechado una serie de tecnologías y herramientas cuidadosamente seleccionadas para construir un sistema eficiente y efectivo. A continuación, detallamos las tecnologías clave que han dado forma a nuestra aplicación.

En primer lugar, sería conveniente hablar de la tecnología encargada de conformar el backend. Se ha optado por el uso de **Spring Boot** como marco de trabajo para desarrollar los microservicios. Esta elección no solo se elige por la experiencia previa si no porque además nos brinda una plataforma robusta y altamente escalable que facilita la creación de componentes autónomos y distribuidos necesarios para alcanzar el éxito de este proyecto.

Nuestros microservicios requieren interactuar con los datos, tanto para su almacenamiento como para su recuperación. En este contexto, hemos optado por emplear diversos sistemas de bases de datos, ya que cada microservicio desempeña un rol único. Por ejemplo, uno de los microservicios hace uso de **MongoDB**, una base de datos NoSQL, que ofrece flexibilidad y eficiencia en la gestión de datos. MongoDB se basa en documentos, lo que brinda una amplia versatilidad en la estructura de los datos. Además, cuenta con un sólido sistema de

indexación y consultas de alto rendimiento, elementos fundamentales en aplicaciones que deben extraer y analizar grandes volúmenes de datos de manera veloz para generar recomendaciones. MongoDB es especialmente adecuado para aplicaciones con extensos volúmenes de datos, ya que permite la distribución de datos en clústeres y la explotación de la replicación y el equilibrio de carga para garantizar la accesibilidad eficiente de los datos.

Pero además MongoDB nos ofrece un aspecto trascendental para la aplicación, a modo de servicio conocido cómo Atlas, qué consiste en un servicio global de bases de datos en la nube totalmente gratuito, para las necesidades que actualmente requerimos. Atlas permite a los desarrolladores almacenar datos como objetos similares a JSON que se asemejan a objetos en el código de la aplicación.

A su vez se necesita una tecnología que garantice un despliegue uniforme y fácil de nuestros microservicios. **Docker** es una tecnología revolucionaria que ha transformado la forma en que los desarrolladores construyen y despliegan aplicaciones en el contexto de la arquitectura de microservicios. Docker ofrece una serie de ventajas clave, como el aislamiento, la escalabilidad, la flexibilidad, la automatización y la gestión de versiones. Además, se utiliza ampliamente en diferentes usos: como el despliegue de aplicaciones, la integración continua y entrega continua, las pruebas de aplicaciones, la infraestructura como código y el despliegue de aplicaciones en la nube [19].

En el lado del frontend, hemos construido nuestra aplicación utilizando **Ionic** y Angular. Esto nos ha permitido crear una interfaz de usuario atractiva y receptiva que funciona de manera eficiente en dispositivos móviles, tanto para Android como para iOS. Esto ahorra tiempo y recursos en comparación con el desarrollo de aplicaciones nativas para cada plataforma. Asimismo, utiliza tecnologías web estándar como HTML, CSS y JavaScript, lo que facilita la adopción para desarrolladores web que ya están familiarizados con estas tecnologías, como es el caso.

En conjunto, estas tecnologías proporcionan una base sólida y coherente para nuestra aplicación, lo que nos permite ofrecer un sistema completo y altamente funcional a nuestros usuarios. Cada elección tecnológica se ha realizado cuidadosamente en función de las necesidades específicas de nuestro proyecto, lo que ha dado como resultado una aplicación bien estructurada y eficiente.

5.2. Arquitectura

En el contexto de este proyecto, la comprensión de la arquitectura general es de suma importancia. La arquitectura no solo proporciona un marco estructural para el desarrollo de la aplicación, sino que también determina su eficiencia, escalabilidad, seguridad y mantenibilidad. Una visión general de la arquitectura no solo es esencial para los desarrolladores y equipo técnico involucrado, sino que también es fundamental para las partes interesadas del proyecto.

Esta sección tiene como objetivo proporcionar una descripción clara y concisa de la arquitectura general que abarca tanto el backend como el frontend de la aplicación. Se explicarán los componentes clave de esta arquitectura, y la relación con las tecnologías mencionadas anteriormente. Además, destacaremos las consideraciones de seguridad, escalabilidad y las ventajas que esta arquitectura aporta al proyecto en su conjunto.

Para facilitar una comprensión común de la estructura general del proyecto se presenta una visión detallada de la arquitectura en la Ilustración 27.

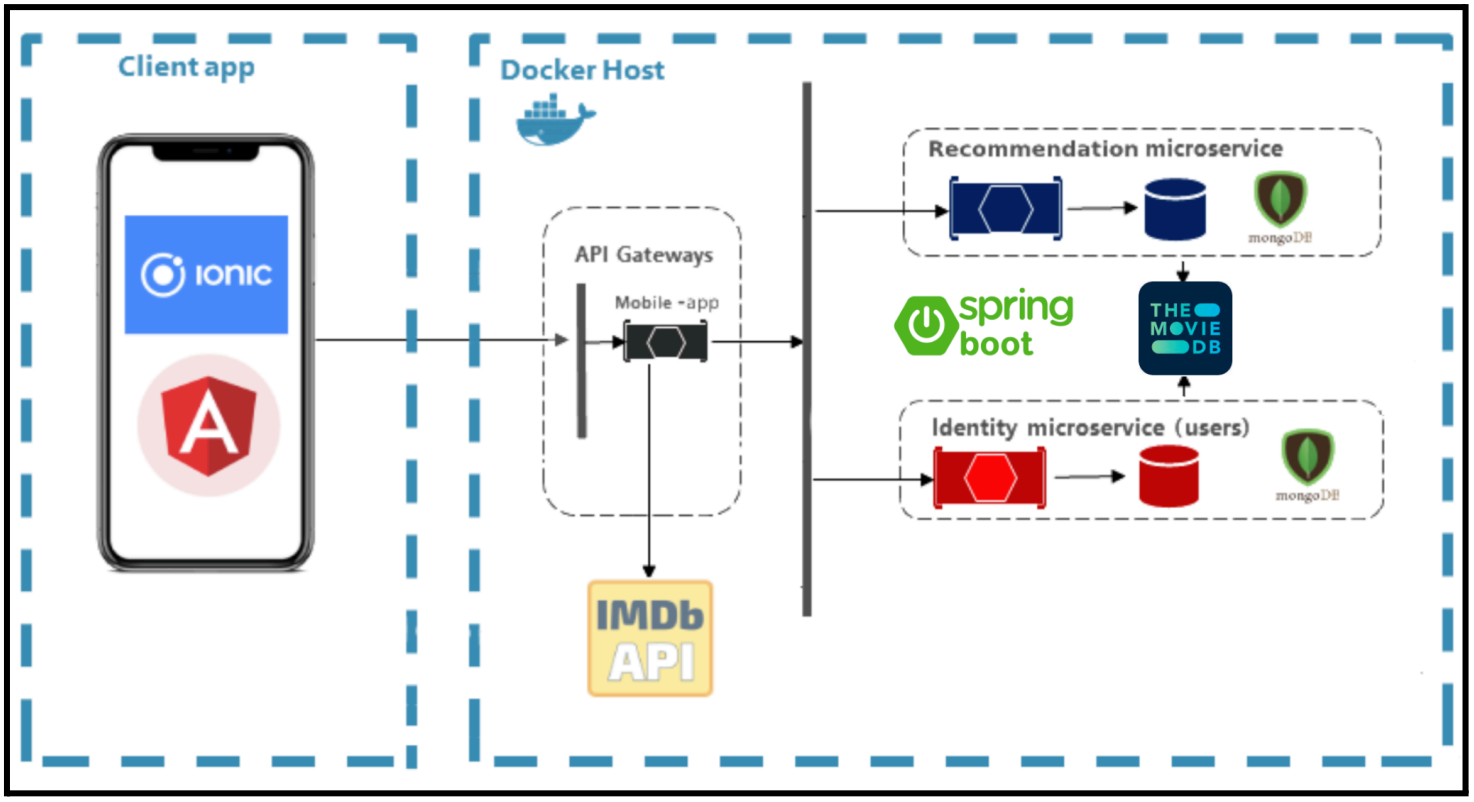

Ilustración 27. Arquitectura general de microservicio y app móvil.

En primer lugar, desgranamos la arquitectura llevada a cabo para el lado del servidor, qué no es otra que una arquitectura por capas, en concreto de 3 capas, muy comunes en aplicaciones de microservicios.

Esta arquitectura se adapta a las necesidades del negocio, con un enfoque sencillo de entender y cuya idea general pasa por tener 3 capas horizontales donde cada una agrupa subtareas que tienen una única responsabilidad y solo pueden acceder a la capa inmediatamente inferior. Para lograr esto, es muy importante que cada capa exponga su funcionalidad a través de abstracciones, de manera de ocultar la complejidad de la implementación a las demás, evitando cometer errores que puedan afectar al sistema [17].

Los microservicios tendrán por lo tanto una capa de presentación, que corresponde a la fachada o a las APIs que éste exponga como punto de entrada para los clientes, una capa de lógica de negocio donde se encuentra la funcionalidad central, y la capa de acceso a datos para comunicarse con las bases de datos o cualquier otro sistema de persistencia.

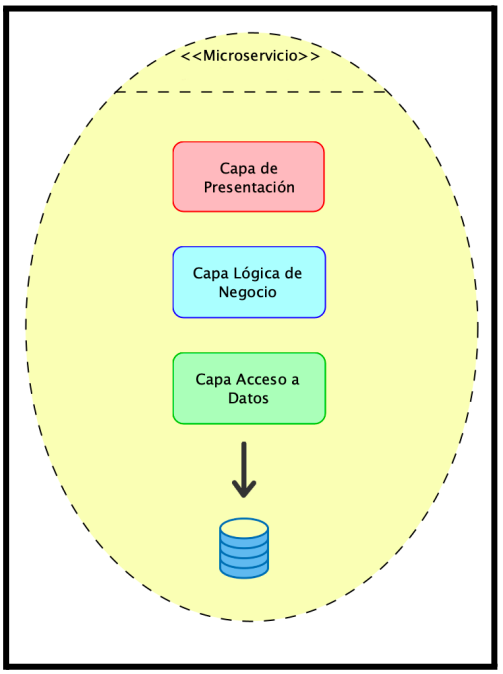

Ilustración 28. Arquitectura 3 Capas de los microservicios.

La capa de presentación sólo se preocupa por presentar la información, pero no le interesa de donde viene ella ni la lógica de negocio que hay detrás. En su lugar, solo sabe que existe una capa por debajo que le proporcionará la información que estará buscando. Por otra parte, bajo la misma idea, el módulo de negocio sólo

se encarga de aplicar todas las reglas de negocio y validaciones, pero no le interesa cómo recuperar, guardar o borrar los datos, ya que, para eso, existe una capa inferior de persistencia. Está, también conocida como de acceso a datos, es encargada de comunicarse con la base de datos, crear las instrucciones SQL para consultar, insertar, actualizar o borrar registros, y retornarlos en un formato independiente. Básicamente, gestionará las entidades y su persistencia. De esta forma, cada parte se preocupa por una cosa y no le interesa como hagan las demás para servirle los datos que requiere, logrando una desvinculación que ayuda a desacoplar el sistema [17].

Arquitectura 3 Capas	
Ventajas	**Inconvenientes**
Separación de responsabilidades: Cada capa tiene una responsabilidad bien definida, se encargará de ella y sólo de ella, simplificando esto la comprensión y mantenimiento del sistema.	**Complejidad**: Debemos preocuparnos por garantizar que las comunicaciones entre capas se ejecuten de manera correcta y liviana, sin que se conviertan en una complejidad extra.
Modularidad: Las capas se pueden desarrollar, probar y mantener de manera independiente, brindando una gran flexibilidad. Esto también permite limitar los errores, sobre todo, al introducir cambios.	**Acoplamiento**: Las capas están interconectadas y dependen unas de otras. Si no se piensa correctamente la estructura y la comunicación, puede generar acoplamiento y dificultar la modificación de una capa sin afectar a otras.
Reutilización de código: Con funcionalidades separadas, podemos reutilizar componentes en distintas partes del sistema.	**Cantidad de capas**: No menor, si tenemos muchas capas, el hecho de que cada una sólo pueda acceder a la inmediatamente inferior puede afectar un poco a la performance cuando la información deba viajar y pasar por cada una de ellas.

Tabla 9. Ventajas y desventajas de la arquitectura de 3 capas.

Finalmente la estructura de la arquitectura para la aplicación del servidor, también conocida cómo *scaffolding*, quedaría con las siguiente paquetería mostrada en la Ilustración 29, muy utilizada en desarrollo de microservicios en la actualidad.

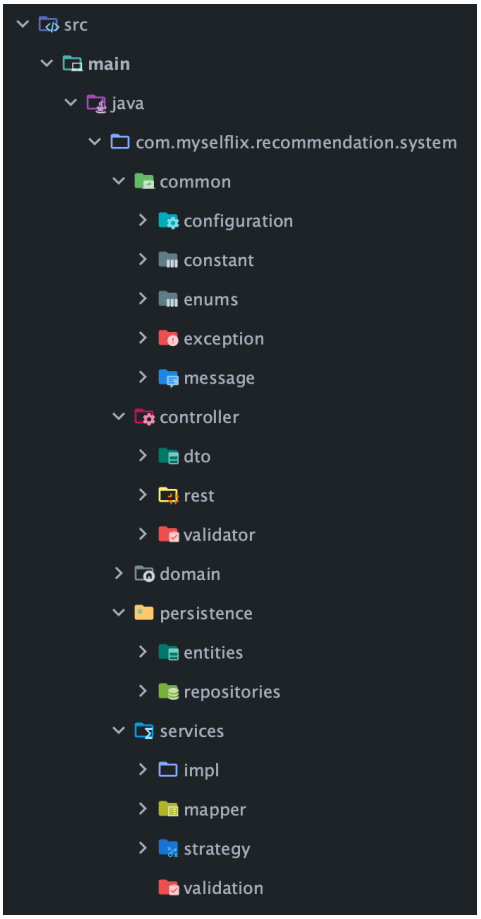

Ilustración 29. Estructura de paquetes del microservicio de Spring Boot.

Podemos observar la forma en la qué se encapsulan las 3 capas en los diferentes paquetes. Tendríamos la capa de negocio en el directorio **services**, la capa de acceso a datos en el directorio **persistence** y por último la capa de presentación en el directorio **controller**. Además tenemos otro tipo de paquetes, típicos en proyectos de microservicios con Spring Boot.

Por otro lado, también tenemos la arquitectura para el frontal. Dado que la tecnología que se utilizará para la aplicación del cliente es el framework ionic y este a su vez se basa en angular, utiliza el patrón conocido como Vista-Controlador.

En la siguiente ilustración se puede ver un esquema de la arquitectura completa que sigue Ionic y Angular:

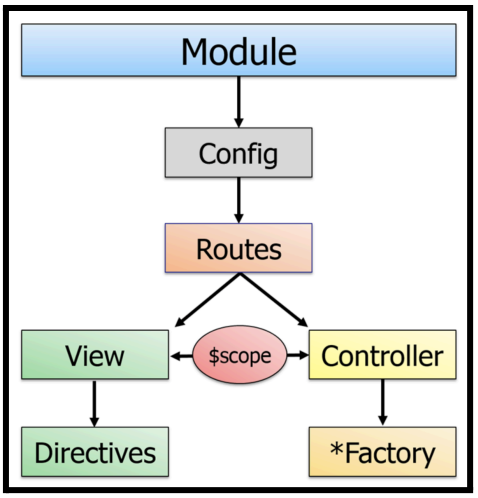

Ilustración 30. Arquitectura del framework Ionic con Angular.

En la arquitectura de una aplicación, se involucran varios tipos de componentes, más allá de las vistas y controladores. A continuación, explicaremos cada uno de ellos gradualmente, aunque los elementos que nos conciernen principalmente son las Vistas, Controladores, Servicios o Factories, así como la Configuración y las Rutas [20].

De manera intuitiva, las funciones de estos componentes en una aplicación con Ionic son las siguientes:

- Los **Controladores** se encargan de adquirir datos de uno o varios servicios o factories y los transmiten a una vista o plantilla a través de la variable *$scope*.
- **Vistas** o plantillas contienen la representación visual de una pantalla (o una parte de ella) y obtienen los datos que se mostrarán de la variable *$scope*.
- La **Configuración** y las **Rutas** de la aplicación permiten vincular los controladores con las vistas o plantillas correspondientes.
- Las **Directivas** son marcadores sobre los elementos del DOM que permiten crear y utilizar componentes con apariencia y comportamiento personalizado.

Por último la estructura de la arquitectura para el frontal quedaría cómo se muestra en la siguiente ilustración.

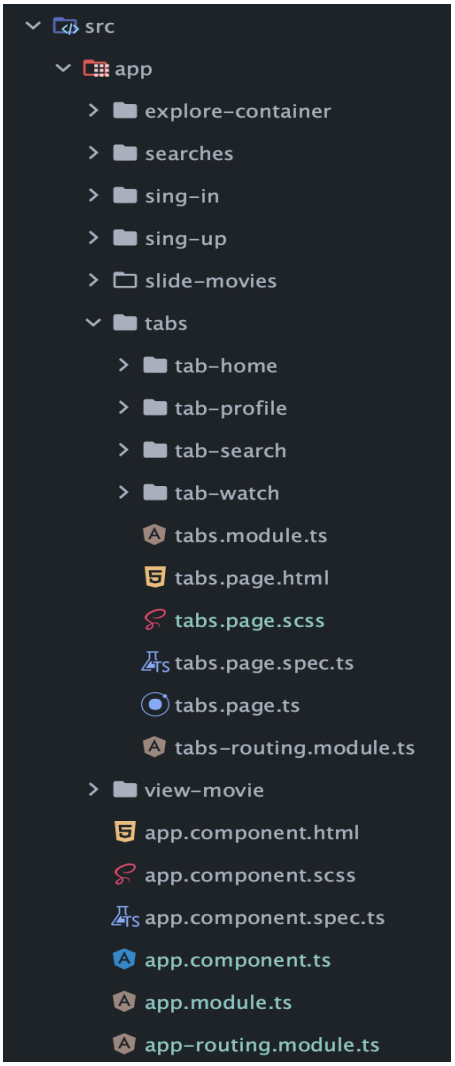

Ilustración 31. Estructura de paquetes de la aplicación de Ionic.

5.3. Algoritmo de Recomendación

El algoritmo qué entrena al modelo y por lo tanto define el sistema de recomendación, ha sido la parte más importante del desarrollo. Ahora se muestra el código encargado de entrenar el modelo en la Ilustración 32 y posteriormente se explicará éste.

```java
private SVDModel getTrainedModel(String ratingsDatasetPath, int K, int steps, double alpha, double beta) {
    loadDataMatrixAndIndexes(ratingsDatasetPath);
    RealMatrix R = createRatingMatrix();
    RealMatrix P = createFeaturesMatrix(R.getRowDimension(), K, K);
    RealMatrix Q = createFeaturesMatrix(K, R.getColumnDimension(), K);
    RealVector biasUser = new ArrayRealVector(R.getRowDimension());
    RealVector biasItem = new ArrayRealVector(R.getColumnDimension());
    double bias = calculateBias(R);

    for (int step=0; step < steps; step++) {
        log.info("Iteration {}", step);
        // Calculate gradient with alpha and beta parameter
        for (int i = 0; i < R.getRowDimension(); i++) {
            for (int j = 0; j < R.getColumnDimension(); j++) {
                if (R.getEntry(i, j) > 0) {
                    double real = R.getEntry(i, j);
                    double prediction = P.getRowVector(i).dotProduct(Q.getColumnVector(j))
                            + bias + biasUser.getEntry(i) + biasItem.getEntry(j);
                    double error = real - prediction;
                    biasUser.setEntry(i, value: biasUser.getEntry(i) + alpha * (error - beta * biasUser.getEntry(i)));
                    biasItem.setEntry(j, value: biasItem.getEntry(j) + alpha * (error - beta * biasItem.getEntry(j)));

                    for (int k = 0; k < K; k++) {
                        double p_ik = P.getEntry(i,k) + alpha * (2 * error * Q.getEntry(k,j) - beta * P.getEntry(i,k));
                        double q_kj = Q.getEntry(k,j) + alpha * (2 * error * P.getEntry(i,k) - beta * Q.getEntry(k,j));

                        P.setEntry(i,k, p_ik);
                        Q.setEntry(k,j, q_kj);
                    }
                }
            }
        }
    }
```

Ilustración 32: Algoritmo de creación del modelo de recomendación.

En este código comprobamos varios aspectos, entre ellos, como se le pasan como parámetros justo los qué conforman la configuración y de los cuales ya se ha hablado en otros apartados, como son alpha, beta, K (número de características) y steps (números de iteraciones).

Por otro lado, hay una serie de variables que toman valor de inicio, en este aspecto, se carga la matriz de valoraciones y los índices en primer lugar, ya sea por base de datos (creación modelo) o desde ficheros de test (evaluación), para posteriormente crear las matrices **R** (matriz con las valoraciones de los usuarios sobre películas), **P** (matriz de usuario) y **Q** (matriz de películas o ítems). La creación de P y Q se puede apreciar en la Ilustración 33.

```java
private RealMatrix createFeaturesMatrix(int rows, int columns, int featuresNumber) {
    Random random = new Random();
    double[][] matrix = new double[rows][columns];

    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < columns; j++) {
            matrix[i][j] = random.nextGaussian() / featuresNumber;
        }
    }
    return MatrixUtils.createRealMatrix(matrix);
}
```

Ilustración 33: Método de creación de matriz de características.

Hay que tener en cuenta, que tanto la matriz de características de usuario P como la matriz de características de items Q, necesitan crearse con unos valores iniciales y a continuación ir ajustándose con cada iteración de entrenamiento. Para que estos valores de inicio sean útiles para el modelo, se generan números aleatorios entre 0 y 1, dividido por el número de características.

Esto garantiza que el modelo no esté sesgado inicialmente hacia ningún conjunto específico de valores. En otras palabras, no asumimos ningún conocimiento previo sobre las preferencias de los usuarios o las características de los ítems, debido a que los sistemas de recomendación deben ser capaces de aprender y adaptarse a partir de datos reales.

Al dividir los valores aleatorios por el número de características, se normaliza la magnitud de los valores iniciales. Esto asegura que los valores no sean demasiado grandes ni demasiado pequeños, lo que podría afectar negativamente la convergencia del modelo.

Continuando con la explicación del código, el entrenamiento se repite tantas veces como iteraciones haya qué hacer. Y en cada iteración se recorre la matriz con las valoraciones, y dado que esta matriz estará vacía en determinados casos, omitimos las posiciones de la matriz qué no tengan valoración real.

Por último, el entrenamiento consiste en ajustar los valores de la matriz P y Q, para lo que se obtiene el error del cálculo de la predicción y el valor real dado por el usuario, utilizando este error junto con los demás parámetros para calcular los nuevos bias y los nuevos valores para las distintas posiciones de P y Q.

Para entenderlo y verlo de forma más visual se pondrán ejemplos con un volumen pequeño de datos de cómo funciona este entrenamiento. En primer lugar, imaginemos que tenemos una matriz de valoraciones de 5 usuarios x 5 películas.

movie_id user_id	0	1	2	3	4
0	5.0	0.0	3.5	4.0	0.0
1	0.0	3.5	0.0	0.0	0.0
2	0.0	2.5	0.0	0.0	0.0
3	0.0	0.0	3.5	4.5	2.0
4	3.5	0.0	0.0	3.0	0.0

Tabla 10. Matriz de valoraciones 5x5.

Además imaginemos que los parámetros de configuración son, 3 iteraciones, 2 características, 0.01 alpha y 0.2 beta. De esta manera, el algoritmo crearía las matrices de P y de Q con valores aleatorios iniciales y normalizados (entre 0 y 0.5 en este caso), siendo P una matriz (5x3) y Q (3x5):

P		
0.3	0.1	0.1
0.3	0.3	0.1
0.2	0.4	0.2
0.2	0.5	0.3
0.1	0.1	0.3

Q				
0.3	0.3	0.1	0.2	0.3
0.4	0.5	0.1	0.2	0.1
0.2	0.1	0.1	0.3	0.4

Tabla 11. Matrices de caraterísticas P y Q.

Es ahora cuando empieza el entrenamiento de estas matrices que irá ajustando cada valor de la matriz por medio de la fórmula mencionada en los fundamentos teóricos y visualizada en el fragmento de código anterior.

Dado que para la posición (0,0) existe una valoración inicial (5.0), calculamos la predicción, la cual se hará a partir del producto escalar de la primera fila de P (*P.getRowVector(0)*) por la primera columna de Q (*Q.getColumnVector(0)*), que dá 0.27, a falta de sumarle las bias correspondientes. Estas bias serían tres, una global que se define como la media de valoraciones y que en este caso es igual a 3.5, una a modo de vector para los usuarios y otra a modo de vector también para las películas. Estas dos últimas bias al ser la primera iteración serán igual a 0 y se irán calculando progresivamente a medida que se siga entrenando el modelo.

Por lo tanto la predicción para este primer caso sería de 3.77 y daría un error de -1.23, el cual será utilizado tanto para los cálculos de las bias de usuario y películas cómo para las matrices P y Q.

Al final en este primera iteración para la posición (0,0) se habrían ajustado las matrices de la siguiente manera, afectando en el caso de P a la primera fila y de Q a la primera columna.

P		
0.292	0.089	0.095
0.3	0.3	0.1
0.2	0.4	0.2
0.2	0.5	0.3
0.1	0.1	0.3

Q				
0.292	0.3	0.1	0.2	0.3
0.397	0.5	0.1	0.2	0.1
0.197	0.1	0.1	0.3	0.4

Tabla 12. Evoluciones de las matrices de caraterísticas P y Q.

A medida en que se fuera avanzando de manera gradual en los pasos del algoritmo, las tablas de usuarios y películas irían adaptándose hasta unas matrices finales que formarían el modelo de recomendación utilizado para la predicción de películas. Estas dos matrices son las que se almacenan en base de datos junto con las bias correspondientes y que facilitan predicciones en tiempo real para el usuario.

Al final, este algoritmo es utilizado tanto para la evaluación del modelo, cómo para la propia creación y guardado en base de datos del modelo, pero para cada una se utilizan distintos conjuntos de datos.

Para la evaluación se utiliza un dataset específico que se divide en 5 partes, siguiendo el método de 5-fold-cross-validation explicado más adelante y que ayuda al proceso de métricas. Tiene 100.836 valoraciones (1-5) de 610 usuarios sobre 9.742 películas. Por otro lado, para la creación formal del modelo tenemos otro dataset con mayor peso, coleccionado por el proyecto de investigación GroupLens de la Universidad de Minnesota con 1 millón de valoraciones de 6.000 usuarios sobre 4.000 películas [21].

5.4. Integración de Microservicios

En este apartado, exploraremos cómo se integran y coordinan los microservicios de nuestra aplicación a través del **API Gateway**. Mientras que en las secciones anteriores se detalló el diseño y funcionamiento de cada microservicio, ahora nos centraremos en cómo estas piezas se comunican para ofrecer una funcionalidad completa.

A continuación, se presentan fragmentos de código del API Gateway que expone el API REST y así poder comprobar su funcionamiento. Cuando un cliente envía una solicitud a esta API, el API Gateway la enruta al microservicio y este se encarga de procesar la petición y devolver la respuesta al API Gateway. Se puede ver más claramente en la Ilustración 34.

```java
@RestController
@AllArgsConstructor
@RequestMapping("/predictive-models")
public class PredictiveModelController {

    private final PredictiveModelService predictiveModelService;

    Javier Serrano
    @PostMapping()
    public ResponseEntity<?> createPredictiveModel(@Validated @RequestBody RequestModelParameterization parameterization) {
        Messages messages = predictiveModelService.savePredictiveModel(parameterization);
        return ResponseEntity.ok(messages);
    }
```

Ilustración 34: Clase controladora para el modelo predictivo.

El siguiente fragmento de código muestra cómo el API Gateway expone la API REST para la creación del modelo predictivo. La anotación *@RestController* indica que la clase ***PredictiveModelController*** es un controlador REST. La anotación *@RequestMapping("/predictive-models")* indica que el controlador expone una API

REST para los modelos predictivos y el método *createPredictiveModel()* llama a la capa de negocio a través del servicio ***PredictiveModelService*** mencionado en los diagramas de clase, guardan así el modelo predictivo.

Al final, esta mecánica comentada se repite en todos los controladores, a diferencia de los distintos tipos de mapeos de solicitud que se exponen en cada uno.

```java
@RestController
@AllArgsConstructor
@RequestMapping("/recommendations")
public class RecommendationController {

    private final RecommendationService recommendationService;

    Javier Serrano
    @GetMapping("/{id-user}")
    public ResponseEntity<?> getRecommendations(@PathVariable("id-user") long userId,
                                                @RequestParam("size")  int recommendationNumber) {
        return ResponseEntity.ok(recommendationService.getRecommendations(userId, recommendationNumber));
    }

    Javier Serrano
    @GetMapping("/{id-user}/{id-movie}")
    public ResponseEntity<?> getPredictiveRating(@PathVariable("id-user") long userId,
                                                 @PathVariable("id-movie") long movieId) {
        return ResponseEntity.ok(recommendationService.getPredictiveRating(userId, movieId));
    }
}
```

Ilustración 35: Clase controladora para las recomendaciones.

```java
@RestController
@AllArgsConstructor
@RequestMapping("/evaluations")
public class EvaluationMetricController {

    private final EvaluationService evaluationService;
    private final EvaluationMetricValidator validator;

    Javier Serrano
    @InitBinder(value = "evaluationValidator")
    public void initBinder(WebDataBinder binder) { binder.setValidator(validator); }

    Javier Serrano *
    @PostMapping(produces = MediaType.APPLICATION_JSON_VALUE)
    public ResponseEntity<?> getEvaluations(@Valid @RequestBody RequestEvaluationMetric evaluationMetric) {
        return ResponseEntity.ok(evaluationService.getEvaluations(
                evaluationMetric.getK(),
                evaluationMetric.getMetric(),
                evaluationMetric.getParameterization()));
    }

}
```

Ilustración 36: Clase controladora para la evaluación de métricas.

Cabe mencionar que es en estos puntos donde se hacen las validaciones de las solicitudes que se describen en los códigos de mensajes del apartado de API REST, verificando que el formato de los parámetros recibidos en la llamada es el correcto. En caso de no cumplir con los formatos, los propios validadores devuelven los mensajes correspondientes de error 400 especificando el problema

Para terminar con esta ejemplificación de cómo funciona el API Gateway, se proporciona un último fragmento de código que sirve de ejemplo para los demás, ya que la estructura y la idea es similar. Obsérvese las ilustraciones 37 y 38.

```java
@RestController
@AllArgsConstructor
@RequestMapping("/movies")
public class MovieController {
    private final MovieService movieService;

    Javier Serrano
    @GetMapping
    public ResponseEntity<?> getMovies() { return  ResponseEntity.ok(movieService.getMovies()); }

    Javier Serrano
    @GetMapping("/id/{id-movie}")
    public ResponseEntity<?> getMovie(@PathVariable("id-movie") Long id) {
        return ResponseEntity.ok(movieService.getMovie(id));
    }

    Javier Serrano
    @GetMapping("/title/{title}")
    public ResponseEntity<?> getMovie(@PathVariable("title") String title) {
        return ResponseEntity.ok(movieService.getMovies(title));
    }

    Javier Serrano
    @PostMapping
    public ResponseEntity<?> addMovie(@RequestBody Movie movie) {
        movieService.saveMovie(movie);

        URI location = ServletUriComponentsBuilder
                .fromCurrentRequest()
                .path("/{id}")
                .buildAndExpand(movie.getMovieId())
                .toUri();

        return ResponseEntity.created(location).body(movie);
    }
```

Ilustración 37: Clase controladora para las películas primera parte.

```
Javier Serrano
@PutMapping("/{id-movie}")
public ResponseEntity<?> updateMovie(@PathVariable("id-movie") Long id,
                                     @RequestBody Movie movie) {
    boolean existChanges = movieService.updateMovie(id, movie);
    return existChanges ?
            ResponseEntity.ok(MessagesConstants.MESSAGE_UPDATED_MOVIE) :
            ResponseEntity.noContent().build();
}

Javier Serrano
@DeleteMapping("/{id-movie}")
public ResponseEntity<?>  deleteMovie(@PathVariable("id-movie") Long id) {
    movieService.deleteMovie(id);
    return ResponseEntity.noContent().build();
}
```

Ilustración 38: Clase controladora para las películas segunda parte.

Se puede intuir que todos los métodos usan para devolver una respuesta la clase llamada ResponseEntity. Esta clase envolvente proporcionada por spring representa todas las respuestas HTTP: código de estado, encabezados y cuerpo. Como resultado, podemos usarla para configurar completamente la respuesta HTTP.

Por otro lado, nuestro frontal debe hacer llamadas a las diferentes APIs tal y como se mostraba en la arquitectura. Para realizar dichas peticiones se ha usado una biblioteca de cliente HTTP llamada Axios, que nos permite realizar solicitudes a un servidor y recibir respuestas fáciles de procesar.

A continuación se muestra en la Ilustración 39 como se hacen dichas peticiones a través de esta biblioteca. Para empezar se determinan los parámetros requeridos de la estructura de la llamada como opciones:

- El tipo de verbo HTTP, en este caso GET.
- La propia url. En el ejemplo, API de imdb con el top 100 de películas. En otros casos se necesitan path variables (**:**) o query params (**?**), qué se definirían en la url.
- La cabecera, en caso de ser necesaria.
- El body, en caso de ser necesario.

```
export class SlideTopMoviesPage implements OnInit {
  private rapidApiKey : string  = 'fae1357bc3msh8ad0535b5490e27p170aecjsn3eb143201c09';
  private readonly topMoviesNumber: number = 50;

  topMovies: any[] = [];

  no usages
  ngOnInit() : void  {
    this.fetchTopMovies() Promise<any>
      .then((data) : void  => {
        this.topMovies = data;
      }) Promise<void>
      .catch((error) : void  => {
        console.error('Error in obtaining the best movies:', error);
      });
  }

  1 usage
  async fetchTopMovies() : Promise<any>  {
    const options : {method: string, url: string, ...  = {
      method: 'GET',
      url: 'https://imdb-top-100-movies.p.rapidapi.com/',
      headers: {
        'X-RapidAPI-Key': this.rapidApiKey,
        'X-RapidAPI-Host': 'imdb-top-100-movies.p.rapidapi.com'
      }
    };

    try {
      const response : AxiosResponse<any, any>  = await axios.request(options);
      return response.data.slice(0, this.topMoviesNumber)
    } catch (error) {
      console.error(error);
      throw error;
    }
  }
```

Ilustración 39: Clase de componente en Ionic con llamada a AP IMDb.

Y por último, mediante el objeto axios importado desde la biblioteca, se hace una petición pasándole las opciones definidas anteriormente y de esta manera recibimos una respuesta que será utilizada por el frontal. En el ejemplo que se ha presentado, es una petición para obtener el top 100 de películas según imdb, lo cual ha demandado una apiKey correspondiente a una suscripción.

5.5. Seguridad de la aplicación

En este apartado se pretende hablar del aspecto de seguridad ya que es un componente crucial en el diseño y desarrollo de cualquier aplicación. En esta sección, se describen las medidas de seguridad implementadas para salvaguardar la integridad, autenticidad y confidencialidad de los datos y la experiencia del usuario, cumpliendo así con el requisito RF07.

El sistema de autenticación y autorización de la aplicación está implementado con ayuda de Spring Security, el cual provee de las suficientes herramientas para llevar a cabo esta tarea. Primeramente, se utiliza un sistema de tokens para gestionar la autenticación de usuarios y garantizar que solo aquellos con credenciales válidas tengan acceso a los recursos protegidos. En concreto se utilizan JWT conocidos cómo JSON Web Tokens. Estos tokens son un estándar que funciona como un mecanismo para poder propagar entre dos partes de forma segura, la identidad de un determinado usuario, junto con una serie de claims o privilegios.

Los principales puntos a destacar son:

- **Endpoint de Inicio de Sesión** (Login): El acceso a este endpoint permite a los usuarios autenticarse y obtener un token válido para acceder a recursos protegidos. Dado que es el punto de entrada para obtener un token será el único junto con el siguiente endpoint qué no necesitarán de éste para devolver un mensaje.
- **Endpoint de Refresco de Tokens**: Para mejorar la seguridad, se ha implementado un mecanismo que permite a los usuarios refrescar sus tokens. Esto ayuda a mantener sesiones activas sin comprometer la seguridad.

La configuración de seguridad se realiza de acuerdo con las mejores prácticas de Spring Security. Se han establecido roles y permisos específicos para garantizar que los usuarios solo tengan acceso a las partes de la aplicación para las que están autorizados. Por otro lado se ha deshabilitado la protección de *csrf* y *cors* qué ofrece spring security para los dos endpoints anteriores y así permitir devolver una respuesta ya que son los dos únicos endpoints que no deben estar securizados. Por

defecto spring security tiene habilitadas estas protecciones que previenen ante cierto tipo de de ataques habituales en aplicaciones web [22].

En esa misma línea, cabe destacar que la autenticación se hace por medio de consultas del usuario y la contraseña en base de datos, por lo que obviamente la contraseña previamente se guarda cifrada. Aspecto de no llevarse a cabo sería un claro ejemplo de malas prácticas. Este cifrado se hace mediante las herramientas que ofrece spring security cómo *PasswordEncorder.*

Para entender en mayor medida lo comentado anteriormente se presenta una Ilustración 40 que representa la estructura de un Json Web Token, con sus tres partes, header, payload y signature.

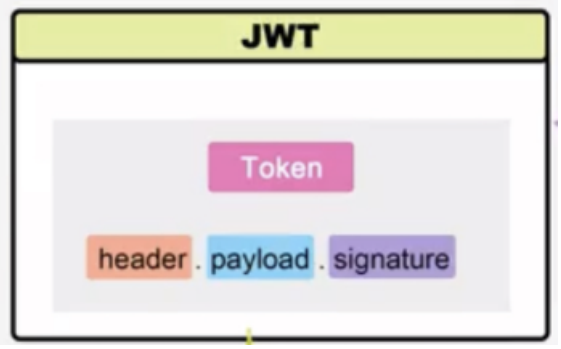

Ilustración 40. Gráfico representativo de las partes de un token.

El encabezado identifica qué algoritmo fue usado para generar la firma, el contenido contiene la información de los privilegios o claims del token, y la firma está calculada codificando el encabezamiento y el contenido en base64url, concatenando ambas partes con un punto como separador.

Para generar el token en el proyecto se hace de la siguiente manera, especificando el usuario, permisos, el algoritmo de firma y el tiempo que tarda en expirar el token

```
1 usage  Javier Serrano *
private String createToken(Map<String, Object> claims, String username) {

    return Jwts.builder()
            .setClaims(claims)
            .setSubject(username)
            .setIssuedAt(new Date(System.currentTimeMillis()))
            .setExpiration(Date.from(Instant.now().plusSeconds( secondsToAdd: 300)))
            .signWith(getSignKey(), SignatureAlgorithm.HS256).compact();
}
```

6. PRUEBAS

En esta sección se presentan diferentes pruebas llevadas a cabo durante la implementación del algoritmo de recomendación, tanto de rendimiento sobre la precisión de las recomendaciones cómo temporales conforme al tiempo qué tarda en entrenarse el modelo, debido a que son pruebas que ofrecen mucha información sobre la fiabilidad del modelo y el algoritmo de recomendación.

De hecho gracias a dichas pruebas se ha logrado hacer una evaluación de los resultados obtenidos y en base a estos, buscar nuevas mejoras y evoluciones hasta obtener un modelo entrenado lo suficientemente robusto como para cumplir con las expectativas de un sistema fiable y de calidad.

Cómo último subapartado, se mostrarán los diferentes resultados obtenidos del sistema de recomendación final después de su evolución. Estos resultados explicarán el porqué se ha decidido los distintos valores para los parámetros de entrenamiento del modelo.

6.1. Métricas de Evaluación del Modelo

Para evaluar las pruebas de rendimiento se utiliza el método de 5-fold-cross-validation [23] (training 80% - test 20%), ya que proporciona un buen equilibrio entre evaluación confiable y un costo computacional razonable teniendo en cuenta el conjunto de datos utilizado para las pruebas.

El método de 5-fold-cross-validation o validación cruzada 5-fold es una técnica comúnmente utilizada en el campo de la ciencia de datos y el aprendizaje automático para evaluar el rendimiento de un modelo de manera robusta y precisa. Esta técnica es especialmente útil cuando se tiene un conjunto de datos limitado y se desea obtener una estimación confiable del rendimiento del modelo.

Éste método consiste en primer lugar, en la partición de un conjunto de datos original, utilizado para entrenar el modelo, en 5 subparticiones. Este será un 80% de entrenamiento y un 20% de test, por lo qué 4 de estas particiones serán de entrenamiento y 1 partición servirá para el testeo.

La forma de llevarlo a cabo es mediante la realización de 4 iteraciones de entrenamiento del modelo con cada una de las particiones y compararlas con la partición de test. A partir de esa comparación se obtendrán diferentes métricas, las

cuales nos darán información relevante sobre el rendimiento y eficacia del sistema de recomendación implementado. El resultado final será una media de las métricas obtenidas en las 4 iteraciones.

Las métricas qué se utilizan para evaluar el sistema de recomendación son muy importantes para determinar la fiabilidad del sistema, en nuestro caso obtendremos las siguientes métricas de pruebas offline, las cuales suelen ser habituales para estos sistemas de recomendación:

- **RMSE** (Root Mean Square Error). Es una medida de la diferencia entre las predicciones del modelo y las observaciones reales y mide la raíz cuadrada del promedio de los errores al cuadrado entre las predicciones de un modelo y los valores reales en un conjunto de datos. Un valor de RMSE más bajo indica un mejor ajuste del modelo a los datos. RMSE cuantifica la magnitud promedio de los errores en las predicciones en la misma unidad que la variable objetivo.

$$\mathrm{RMSE} = \sqrt{\frac{1}{|\mathcal{T}|}\sum_{(u,i)\in\mathcal{T}}(\hat{r}_{ui} - r_{ui})^2}$$

- **MAE** (Mean Absolute Error). Es una medida de la magnitud promedio de los errores de predicción y mide el promedio de las diferencias absolutas entre las predicciones de un modelo y los valores reales en un conjunto de datos. Un MAE más bajo indica un mejor ajuste del modelo a los datos. Es menos sensible a valores atípicos en los datos que el RMSE, ya que no eleva al cuadrado los errores.

$$\frac{1}{n}\sum_{u,i}|p_{u,i} - r_{u,i}|$$

- **Precisión**. Es una métrica comúnmente utilizada en tareas de clasificación y recuperación de información que mide la proporción de elementos identificados como positivos que son verdaderamente positivos (TP) con respecto al número total de elementos identificados como positivos (verdaderos positivos - TP y falsos positivos - FP). Una alta precisión indica que la mayoría de las identificaciones positivas hechas por el modelo son correctas y confiables.

$$P = TP/(TP+FP)$$

- **Recall**. El recall es otra métrica utilizada en clasificación y recuperación de información que mide la proporción de elementos verdaderamente positivos (TP) que fueron identificados correctamente por el modelo con respecto al número total de elementos que son realmente positivos (verdaderos positivos - TP y falsos negativos - FN). Un alto recall indica que el modelo es efectivo para identificar la mayoría de los elementos positivos presentes en el conjunto de datos.

$$R = TP/(TP+FN)$$

- **Score-F**. También conocida cómo medida F es una medida de la precisión de una prueba. La puntuación de F1 es la media armónica de la precisión y la capacidad de recuperación. El F1 Score es especialmente útil cuando hay un desequilibrio entre las clases en un conjunto de datos.

$$F_1 = \frac{2RP}{R+P} = \frac{2TP}{2TP+FN+FP}$$

- **DGC** (Discounted Cumulative Gain). Es una métrica utilizada en sistemas de recomendación para evaluar la calidad de una lista ordenada de elementos recomendados. Cuantifica la utilidad acumulativa de los elementos recomendados, dando más peso a los elementos en posiciones superiores de la lista. Un DCG más alto indica una lista de recomendación de mayor calidad. Es útil para medir la relevancia y la posición de los elementos recomendados en una lista.

$$DGC(R) = \sum_i u(i)d(i) \qquad u(i) = r_{a,i} \qquad d(i) = \frac{1}{\min\left(1,\ \log_2 i\right)}$$

$$u(i) = \begin{cases} 1 \text{ if clicked} \\ 0 \text{ otherwise} \end{cases} \qquad d(i) = \frac{1}{2^{\frac{i-1}{\propto-1}}}$$

6.2. Evolución del Sistema de Recomendación

El algoritmo diseñado para entrenar al modelo y que conforma el sistema de recomendación en sí, ha ido sufriendo diferentes modificaciones hasta alcanzar unos resultados satisfactorios en las métricas de evaluación.

En primer lugar, se implementó un método de entrenamiento qué utilizaba una librería muy conocida de apache llamada ***Commons Math*** [24], la cual proporciona la clase *SingularValueDecomposition*. Esta clase es importante ya que a partir de una matriz de valoraciones, te permite obtener las matrices descompuestas de usuarios, ítems y los pesos.

Tras revisar el rendimiento de precisión de este primer intento, los resultados quedaron lejos de ser unos resultados apropiados, ya que las métricas indicaban que las recomendaciones no eran fiables. Por exponer un ejemplo, se obtenía un valor mayor que 1.2 en la métrica de MAE, con unos tiempos superiores a los 15 minutos para ejecutar la evaluación con un conjunto de datos de 100 películas, 5.564 usuarios y 338.355 valoraciones. Se puede observar con más detalle estos resultados en la siguiente Ilustración 40, qué muestra las métricas de respuesta tras enviar una solicitud POST con el programa Postman al sistema de recomendación.

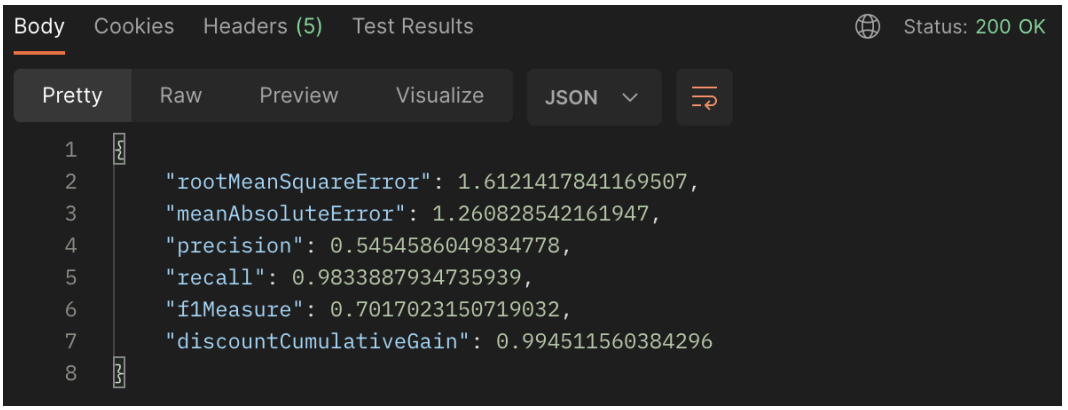

Ilustración 41. Resultados obtenidos mediante Postman del servicio de evaluación primerizo.

Tras este rendimiento se buscó otras alternativas para evolucionar el algoritmo. Fue muy importante para avanzar, el análisis del artículo qué explica el método de descenso por gradiente estocástico [9]. Con el cual se cambió el planteamiento del entrenamiento del modelo, entrando en juego nuevas fórmulas descritas en el apartado 1.5 de fundamentos teóricos y el uso de parámetros expuesto en el apartado 2.2 de descripción de solución propuesta.

Gracias a esto se empezó a mejorar los resultados, no obstante, sin ser lo suficientemente buenos cómo para dejarlos cómo definitivos, por lo qué se siguió investigando nuevas formas de mejorar el algoritmo. Fue cuando se observó qué las matrices de características tanto para el usuario cómo para los ítems, se

inicializaban para el primer entrenamiento con valores aleatorios entre 1 y 5, qué es el rango de puntuación qué tienen los conjuntos de datos para las valoraciones. Y esto hacía qué desde el principio se partiera de un modelo con muy malos resultados pero qué iba mejorando progresivamente con cada iteración. Con lo cual se realizó una modificación para qué las matrices de características inicialmente tuvieran valores aleatorios entre 0 y 1, divididos por el número de características a modo de normalizado.

Esto fue un gran éxito, ya qué supuso un gran avance en las métricas obtenidas, partiendo las primeras iteraciones de una mejor solución.

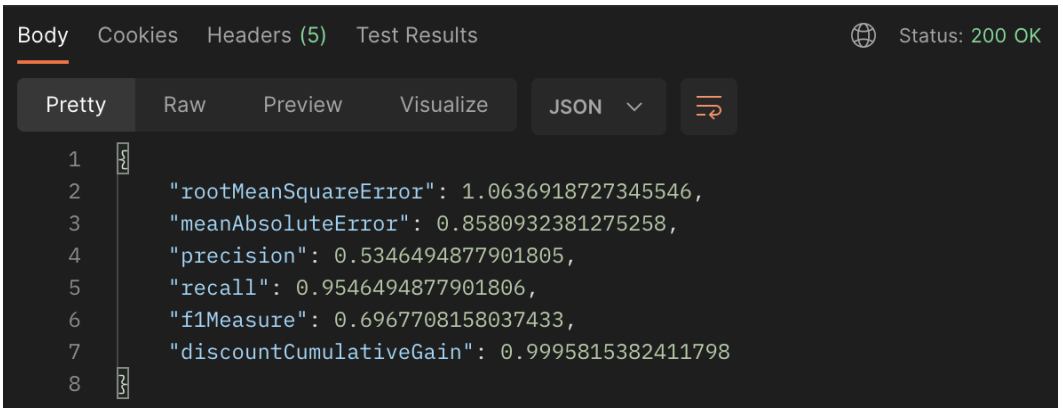

Ilustración 42. Resultados obtenidos mediante Postman del servicio de evaluación intermedio.

No satisfecho con eso, se buscó una última mejora qué proporcionará un modelo con resultados diferenciales acordes a un Trabajo Fin de Máster. Así qué por último se estudió la influencia y funcionamiento de las bias o interceptores, mencionado en el artículo Matrix factorization techniques for recommender system [5]. Incluyendo estos sesgos qué reflejan tanto la tendencia de los usuarios cómo de las películas, se mejoraron considerablemente dichos resultados.

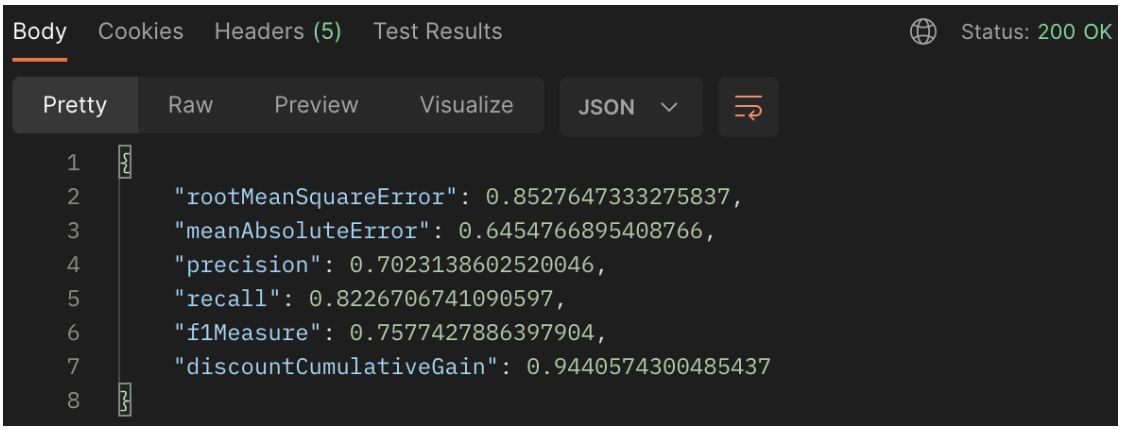

Ilustración 43. Resultados obtenidos mediante Postman del servicio de evaluación final.

De esta manera, se dió por resuelto el algoritmo qué creaba el modelo con el cual se hacen recomendaciones precisas y eficientes, siendo la fórmula de predicción el producto escalar de los vectores de características del usuario y de las películas, junto con la suma de las correspondientes bias o sesgos. Este algoritmo final es el que se presenta en los distintos puntos de la memoria, como en el apartado de fundamentos teóricos o incluso en el apartado que explica el algoritmo a nivel de código.

6.3. Resultados de Rendimiento del Modelo

Una vez definido el algoritmo que crea el modelo de recomendación, se han hecho varias pruebas de rendimiento que han determinado la precisión que tienen las recomendaciones ofrecidas por el sistema. Estas pruebas persiguen determinar la configuración y parametrización ideal para entrenar el modelo, tal y cómo se menciona en el apartado 2.2 de *descripción de la solución propuesta*.

De esta manera, el apartado expondrá el estudio de diferentes valores a estos parámetros, hasta encontrar los valores más adecuados con los que ultimar el modelo. A continuación se presenta una tabla con los resultados obtenidos de este estudio.

Parámetros	**Valores**									
Steps	10	10	10	20	25	25	25	30	40	**55**
f	10	10	10	20	25	25	25	25	28	**40**
Alpha	0.1	0.1	0.05	0.05	0.05	0.005	0.002	0.002	0.001	**0.001**
Beta	0.1	0.2	0.1	0.1	0.1	0.05	0.03	0.02	0.02	**0.02**
Métricas	**Resultados**									
RMSE	1.026	0.971	0.942	0.941	0.946	0.822	0.828	0.824	0.833	**0.816**
MAE	0.795	0.761	0.727	0.725	0.729	0.628	0.640	0.630	0.643	**0.626**
Precisión	0.651	0.658	0.673	0.675	0.676	0.705	0.697	0.706	0.696	**0.705**
Recall	0.713	0.722	0.763	0.760	0.759	0.837	0.843	0.840	0.841	**0.843**

Score-F	0.681	0.689	0.715	0.715	0.715	0.765	0.763	0.767	0.761	**0.768**
DGC	0.943	0.950	0.946	0.945	0.945	0.948	0.952	0.948	0.952	**0.949**
Tiempo	8”	8”	8”	22”	28”	28”	28”	32”	53”	**90”**

Tabla 13. Valores obtenidos de métricas para diferentes parámetros.

Al fin y al cabo, para encontrar la mejor configuración de parámetros para el conjunto de datos de valoraciones, hay qué hacer varias pruebas e ir afinando hasta determinar el valor ideal de cada parámetro. Para entender estas pruebas ha sido muy útil visualizarlo a través de gráficas, las cuales son herramientas esenciales para comprender y comunicar de manera efectiva cómo diferentes parámetros afectan al rendimiento de precisión del sistema.

A continuación se presentan primeramente los parámetros beta y alpha con distintos valores, posteriormente el número de iteraciones qué se entrena y el número de característica. El eje horizontal muestra los diferentes valores que han ido adoptando los parámetros en las pruebas y en el eje vertical el resultado de la métrica conocida cómo MAE, que cómo se comenta con anterioridad mide el promedio de las diferencias absolutas entre las predicciones de un modelo y los valores reales en un conjunto de datos.

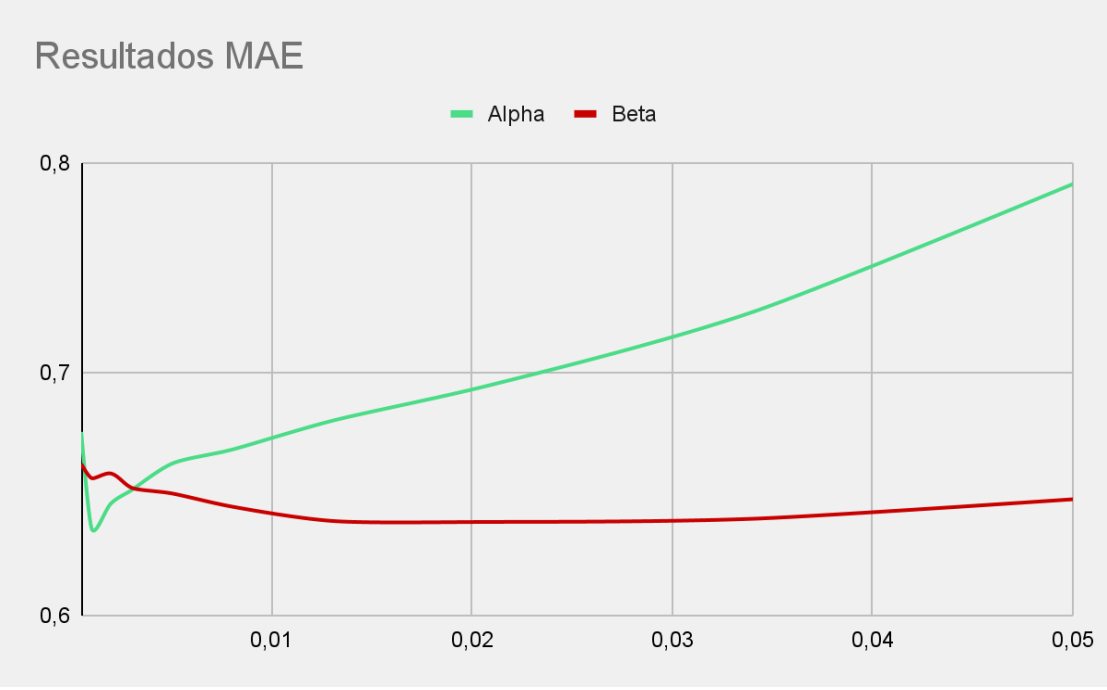

Ilustración 44. Gráfica con los resultados de las pruebas para alpha y beta.

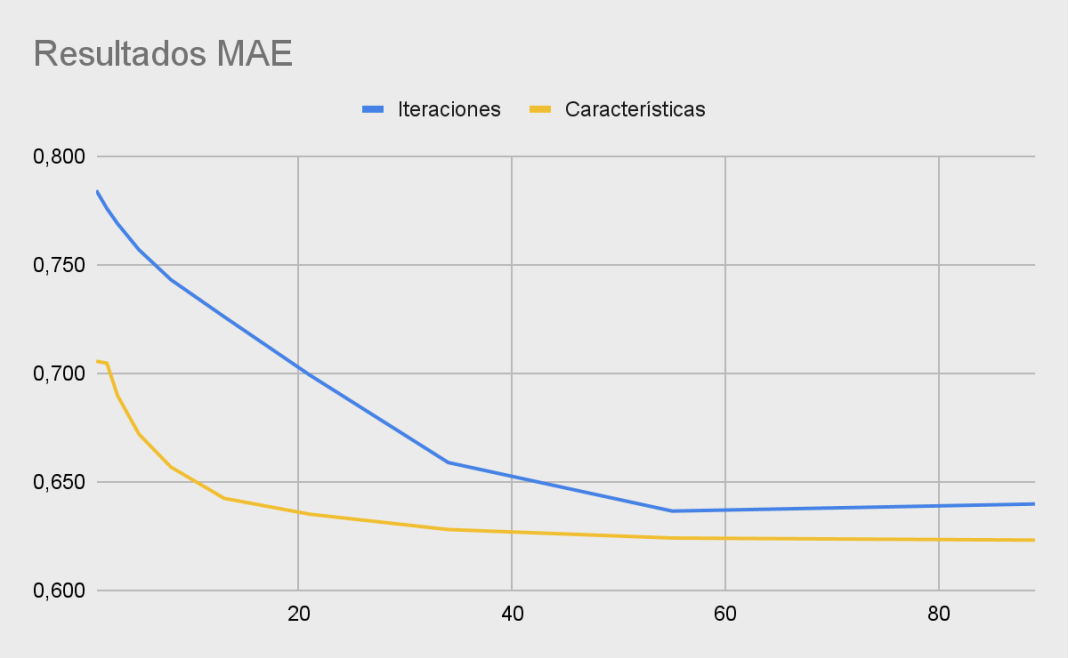

Ilustración 45. Gráfica con los resultados de las pruebas para steps y características.

Teniendo en cuenta que cuanto menor sea el resultado de la métrica MAE mejor es el modelo de recomendación, vamos a definir e interpretar estas gráficas en los siguientes puntos, para explicar la elección de los valores de los parámetros:

- Para **Alpha**, observamos que existe un punto de inflexión para el valor **0.001**, lo cual indica un cambio significativo en el rendimiento del modelo cuando nos salimos de éste, e incluso cuanto mayor es, más se incrementa el resultado de MAE.
- Para **Beta,** se puede comprobar una línea de tipo valle suave entre los valores 0.01 y 0.03, siendo el valor **0.02** donde el resultado es más bajo. Recordemos que esto significa que es el mejor resultado.
- Para las **Iteraciones** o lo que es lo mismo, las veces que se repite el entrenamiento, podemos ver una tendencia donde a mayor número de iteraciones mejor. No obstante, existe un punto concreto donde a partir de ahí se incrementa ligeramente. Por eso, el número de iteraciones elegidas es **55**.
- Para el número de **características**, vemos también que sigue una tendencia donde a mayor número de características mejor es el resultado, pero hay que decir, que esto aumenta considerablemente el tiempo que el sistema tarda en crear el modelo y realmente la mejora no es significativa, por lo que, haciendo

un balance entre la calidad del modelo y lo que tarda en crearse se ha elegido **40** características.

Por lo tanto, después de probar, investigar y comprender las pruebas realizadas nos quedamos finalmente con estos valores para la configuración de nuestro modelo, la cual nos aporta las mejores métricas por el momento. En la ilustración vemos todas las métricas obtenidas para esta parametrización final, además de la demora en el tiempo de 1 minuto y 30 segundos desde que se creó la petición de evaluación hasta que se recibe la respuesta. Tiempo qué engloba a crear el modelo de recomendación y calcular sus métricas, lo cual es increíble teniendo en cuenta que el conjunto de datos con el que se evalúa el sistema es conforme a 100 películas, 5.564 usuarios y 338.355 valoraciones.

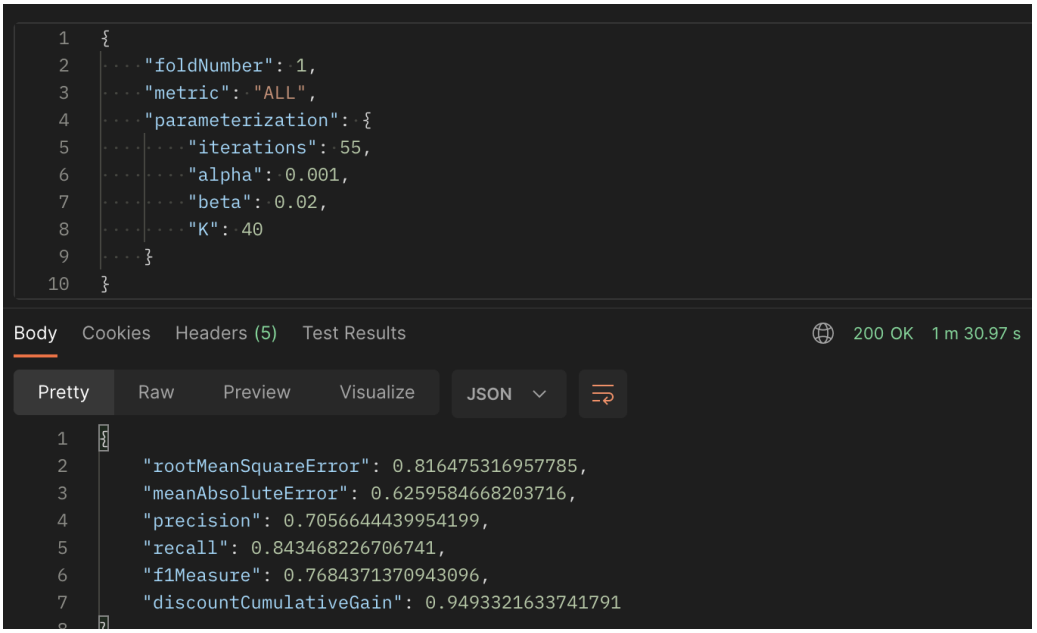

Ilustración 46. Solicitud GET del API REST de evaluaciones con resultados finales.

Cabe destacar que estos resultados son en base al conjunto de datos de los cuales se entrena el modelo, con lo cual para un conjunto de datos diferentes habría que realizar de nuevo este análisis y determinar qué parámetros son los mejores. Afortunadamente, el sistema de recomendación tal y como se ha creado, nos permite hacer tantas pruebas cómo queramos.

7. CONCLUSIONES

Este TFM propone una solución para que los usuarios que usen la aplicación no pierdan demasiado tiempo valorando la elección de que películas ver, aportando directamente recomendaciones de contenido con una alta probabilidad de que ese filtrado sea de su gusto.

Para el éxito de la solución, se han elegido una serie de tecnologías que han resultado ser adecuadas y sólidas para cumplir con los objetivos marcados al principio. Han permitido un desarrollo ágil y efectivo para los componentes necesarios de la aplicación, ya que proporcionaban módulos que facilitaban el desarrollo, cómo ha podido ser *spring-security* o *spring-boot-starter-web*, por no hablar de la sencillez para crear la interfaz con ionic creator, una herramienta de creación de prototipos de arrastrar y soltar para crear grandes aplicaciones utilizando Ionic, con sólo un clic del ratón. Por otro lado, elegir una base de datos no relacional de tipo documental cómo base de datos para las recomendaciones ha sido otro acierto, ya que ha proporcionado mucha velocidad para las operaciones de lectura y escritura teniendo en cuenta el volumen de datos tan alto.

No obstante, todo no es perfecto, pues estas opciones han podido conllevar algunos inconvenientes como por ejemplo la integración de todas las tecnologías en una arquitectura coherente, lo que ha llevado a un esfuerzo significativo para garantizar la comunicación eficaz entre los componentes. Otro punto importante es la posibilidad de que la base de datos de MongoDB no se administre adecuadamente, pudiendo resultar en redundancia y sobrecarga de datos, lo que afectaría al rendimiento.

Pero en líneas generales, estas decisiones han ayudado a cubrir los diferentes objetivos iniciales propuestos, especialmente el del sistema de recomendación, el cual era el objetivo principal.

No obstante, si hay un requisito inicial, el cual no ha dado tiempo a abordar pero que con la estructura del proyecto en sí no requiere un esfuerzo abordarlo en trabajos futuros. Este requisito se trata del RF07, que trataba del registro de usuario y a la autenticación de estos y pese a que la autenticación si se ha llegado a implementar, el registro por el contrario no.

Otro aspecto a comentar es el tipo de sistema de recomendación elegido, y lo cierto es que lo ideal hubiera sido haber construido un sistema de recomendación híbrido. De esta manera, podríamos haber aprovechado las ventajas de cada sistema, ofreciendo mayor flexibilidad para las recomendaciones. Sin embargo, esto hubiera supuesto excederse en el alcance final de un trabajo fin de máster, ya que se hubieran tenido que construir varios sistemas de recomendación en lugar de uno, además de la lógica implícita de decidir cuándo y en qué casos usar cada uno.

A pesar de todo esto, el sistema final de recomendación implementado nos ha brindado una precisión muy ventajosa siendo esta una de las bondades del sistema propuesto. Igualmente hemos obtenido una gran escalabilidad, siendo capaces de adaptarnos a un gran número de usuarios y datos, sin mencionar qué además se ha conseguido una gran eficiencia en términos de tiempo y recursos.

8. Trabajos Futuros

Este apartado describe las posibles mejoras de la aplicación. Para empezar, uno de estos posibles trabajos a futuro es convertir el sistema de recomendaciones en un sistema híbrido como se menciona en el anterior punto, lo qué implica agrupar varios sistemas de recomendación entre ellos el ya implementado de filtrado colaborativo, junto con otros. Esto enriquecería las recomendaciones y aportaría lo mejor de cada sistema.

Realmente se podría construir un sistema de recomendación basado en contenido que utilice los módulos ya desarrollados en el microservicio de recomendación. Así mismo, tendríamos un sistema híbrido de recomendación que utilizaría el filtrado colaborativo para recomendar productos en función de las preferencias de los usuarios similares, y el filtrado basado en contenido para recomendar productos similares a los que el usuario ha mostrado interés previamente. Incluso con la comunicación al API de IMDB encontramos un sistema extra de conocimiento que no hay siquiera que implementar, y tendríamos unas recomendaciones al nivel de las de Netflix, la poderosa empresa de entretenimiento de streaming, con el añadido de que serían recomendaciones existentes en otras plataformas de streaming.

También cabe la posibilidad de escalar el tipo de contenido que se recomienda, como por ejemplo las series. Actualmente solo se recomiendan películas, pero es habitual que junto a este tipo de recomendaciones aparezcan también las series. Gracias a la estructura del microservicio y el tipo de sistema de recomendación implementado, esto no supondría modificaciones en el ya existente, valiendo tanto para películas como para series. A pesar de ello, si habría que encontrar conjuntos de datos con valoraciones de series que sirvieran para crear el modelo de recomendación de series. Con lo cual, a partir de unos ligeros cambios en la interfaz de la aplicación y un nuevo conjunto de datos se podría añadir ágilmente esta nueva funcionalidad, gracias sin duda a la arquitectura conjunta del proyecto.

Por lo que se refiere a mejoras o trabajos futuros, también se podría proponer un tercer punto que sería interesante introducir, y es qué vivimos en la era de las redes sociales, con lo que podría ser sugerente hacer un enfoque de red social a

esta aplicación, permitiendo seguir y que te sigan a otros usuarios, viendo sus recomendaciones propias.

Por último, otra idea para evolucionar el sistema puede estar en la creación de un nuevo microservicio que se encargue de hacer las llamadas a las APIs externas. Esto tiene un motivo y es que cada petición que se hace, como por ejemplo a la API de IMdb, tiene un precio y se pueden ahorrar muchas peticiones a estas APIs si se acompaña a este nuevo microservicio de una caché que evite que se hagan llamadas repetidas y descontrolas que aumenten el gasto. Para este proyecto sin ir más lejos, se ha partido de una suscripción gratuita pero que solo permite 100 solicitudes al mes, lo cual es realmente insuficiente si se quisiera comercializar la aplicación. De este modo, con el nuevo microservicio encargado de hacer estas llamadas, cuando la aplicación pidiera información de una película se haría una única llamada para esa película y no varias por cada usuario.

En definitiva, se pueden incluir varias evoluciones de la aplicación que la hagan más completa y las tecnologías escogidas junto con la arquitectura diseñada facilitarían la integración de estas.

9. APÉNDICES

9.1. Material Complementario

Para este subapartado exploraremos brevemente el material complementario y adicional a la memoria que se entrega en un fichero zip con todo lo generado e importante durante el desarrollo del TFM. Para ello se presenta la estructura de este fichero comprimido en la Ilustración 46.

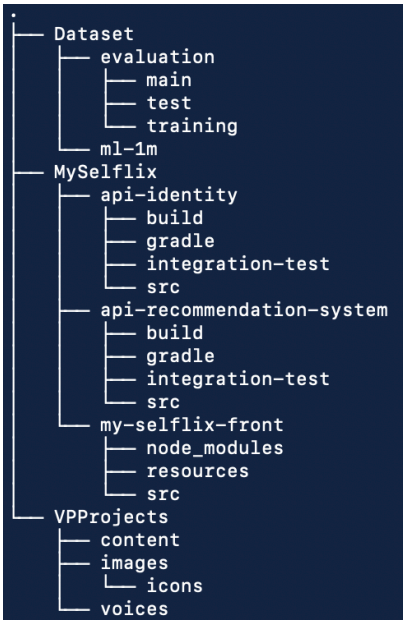

Ilustración 47. Estructura de directorios en 3 niveles del material complementario.

Contiene la aplicación móvil en la carpeta MySelflix compuesta por los dos microservicios y el propio frontal. En cada microservicio se puede obtener colecciones de postman con los diferentes endpoints que proveen las apis, situados en la carpeta integration-test.

Además se han incluido otras fuentes con todo el material correspondiente a visual paradigm que se encuentra en la carpeta llamada VPProjects. Ahí se encuentra un fichero de nombre index.html que navega por los diferentes diagramas y modelos de diseño.

De igual modo también existe un fichero docker-compose con la configuración de las imágenes de docker que se detalla en el siguiente subapartado.

Finalmente también se aportan los dataset utilizados tanto para la evaluación como para la creación del modelo predictivo (ml-1m) en una carpeta llamada Dataset.

9.2. Instalación y Configuración del Sistema

Seguidamente pasamos a explicar cómo lanzar la aplicación a partir del material suministrado. Cada microservicio que comienza con la nomenclatura api tiene en su interior un fichero .jar que se encuentra en la ruta /build/libs. Estos ficheros se generan al lanzar el comando *gradle clean build* en un terminal.

En el mundo de DevOps, para desplegar servicios en Docker, se utilizan archivos y configuraciones. El proceso es simple, ya que Docker ofrece Dockerfile para crear imágenes de contenedores. Una imagen es una plantilla que Docker utiliza para crear contenedores. Por lo tanto, cada microservicio tiene un archivo Dockerfile con tres propiedades básicas que permiten ejecutar la aplicación.

En otras palabras, para desplegar un servicio en Docker, necesitamos crear una imagen de contenedor. Una imagen de contenedor es un paquete de código, dependencias e instrucciones para ejecutar una aplicación. Podemos crear una imagen de contenedor usando un archivo llamado Dockerfile. Un Dockerfile es un archivo de texto que contiene instrucciones para crear una imagen de contenedor.

```
FROM adoptium/openjdk:17-jre
ADD target/api-recommendations-system-1.1.0.jar app.jar
ENTRYPOINT ["java", "-Djava.security.egd=file:/dev/./urandom", "-jar", "/app.jar"]
```

Ilustración 48. Fichero Dockerfile para el microservicio de recomendación.

Las tres propiedades básicas de un Dockerfile son:

- La propiedad FROM es la imagen base que vamos a tomar, constituida por una distribución de Linux extremadamente ligera y un jdk para ejecutar aplicaciones Java.
- ADD nos servirá para crear un directorio en el que alojaremos el ejecutable de cada una de nuestras aplicaciones.
- Y ENTRYPOINT será el comando que se ejecute cuando se levante el contenedor con esta imagen, el cuál ejecutará nuestra aplicación usando el fichero .jar.

Docker también ofrece la herramienta docker-compose, que nos ahorra ejecutar un comando para levantar cada contenedor, permitiendo centralizar la configuración de despliegue, lo cuál es mucho más práctico.

La sintaxis de este fichero es bastante sencilla, y podemos alojarlo donde queramos, ya que lo único que necesitamos son las imágenes creadas anteriormente, que ya están subidas a nuestro repositorio de imágenes en local. El fichero aportado cómo material complementario se llama docker-compose.yml, y luce cómo en la siguiente ilustración.

```yaml
version: '3.8'
services:
  recommendation-microservice:
    image: recommendationmicroservice
    restart: always
    container_name: recommendationmicroservice
    networks:
      - 'docker-netflix-network'
  identity-microservice:
    image: identitymicroservice
    restart: always
    container_name: identitymicroservice
    networks:
      - 'docker-netflix-network'
networks:
  dockernetflix-network:
```

Ilustración 49. Fichero docker compose con la configuración.

Por último, sólo queda desplegar los contenedores que se hará por medio del comando "***docker-compose up -d***".

Con esto ya estarían disponibles los microservicios a través del Gateway con lo cual solo queda desplegar la aplicación frontal. Para ello es necesario tener la versión 16 de Angular previamente instalada, si no es el caso instalar con el comando "***npm install -g @angular/cli***". Igualmente es necesario tener la versión 6 de Ionic y de la misma manera instalamos con "***npm install -g @ionic/cli***". Hay que matizar que para poder hacer uso del comando npm debe tener ***node.js*** instalado, en caso de ser también necesario se puede instalar desde su página oficial o desde un terminal con el comando "***brew install node***" (para macOS).

Con el fin de terminar de levantar toda la aplicación, ya solo tienes que dirigirte al directorio raíz *my-selflix-front* en un terminal y lanzar el comando "***ionic serve***" para el navegador. Tras esto, directamente se levantará la aplicación y podrá ser accedida mediante la ruta **http://localhost:4200/** a partir de un navegador.

Otra opción es lanzar el comando "***ionic cordova prepare ios***" para probar en ios o incluso para android con "***ionic cordova prepare android***". Para ello seguir instrucciones [25][26].

9.3. Manuales de Usuario

En el subapartado de manuales de usuario vamos a ver cómo el usuario debe interactuar con la interfaz de la aplicación. Esta interfaz ha seguido los pasos marcados en el storyboard, con lo cual encontraremos 4 pestañas principales para moverse dentro de la aplicación. La primera pestaña y más importante considerando la temática del proyecto es la de inicio, y en ella se presentan las recomendaciones al usuario.

Desgranando esta pestaña principal encontramos las recomendaciones segmentadas en diferentes categorías, de las cuales primero aparecen las recomendaciones que el sistema recomienda al usuario, seguidas posteriormente de un top 50 de películas según IMdb y un top semanal de las 10 películas más populares del momento según IMdb.

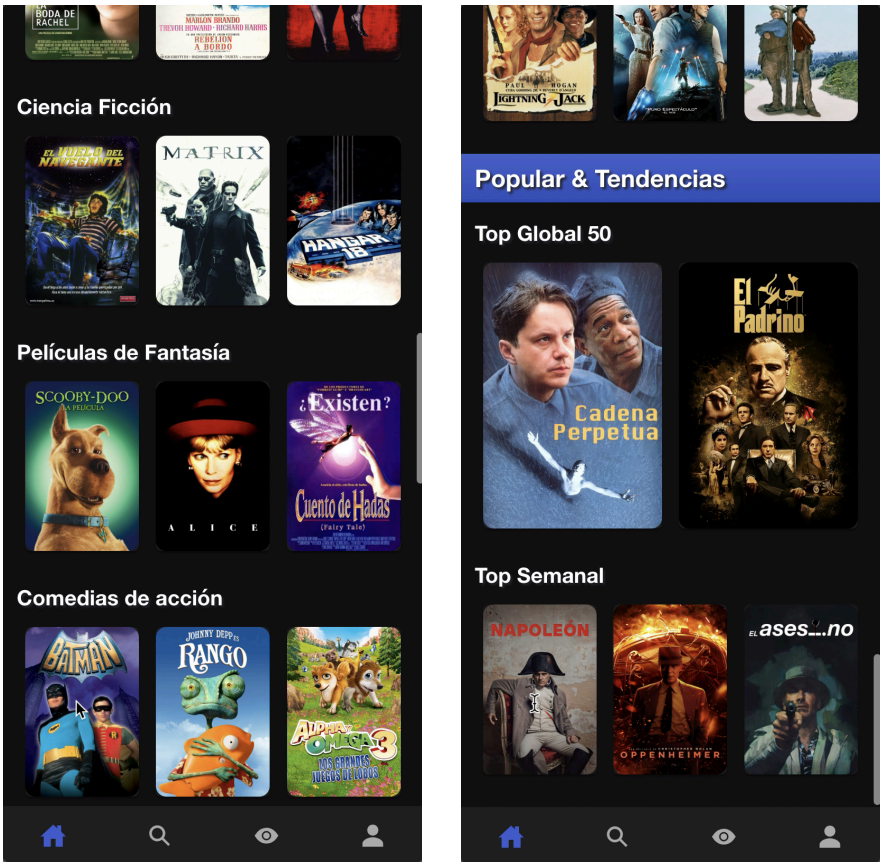

Ilustración 50: Capturas de la aplicación en la pestaña de inicio.

Para moverse por la página de inicio hay que deslizarse hacia arriba y abajo o hacia izquierda o derecha en las distintas categorías, ya que se ofrecen cinco recomendaciones por categoría, con lo que deslizas tu dedo hacia un sentido se desplaza la lista de recomendaciones. Además pinchando en cualquiera de las películas que aparecen, entramos en una página que visualiza la película con información más detallada y con posibles acciones sobre la película que se explicará más adelante.

Igualmente tenemos la pestaña de la lista de películas a ver, que agrupa y almacena el conjunto de películas que el usuario ha añadido para ver. La lista muestra distinta información sobre cada película cómo el propio título o la imagen de identificativa de dicha película. Por último si se pulsa algún ítem de la lista nos abre una página para visualizar la película con información detallada.

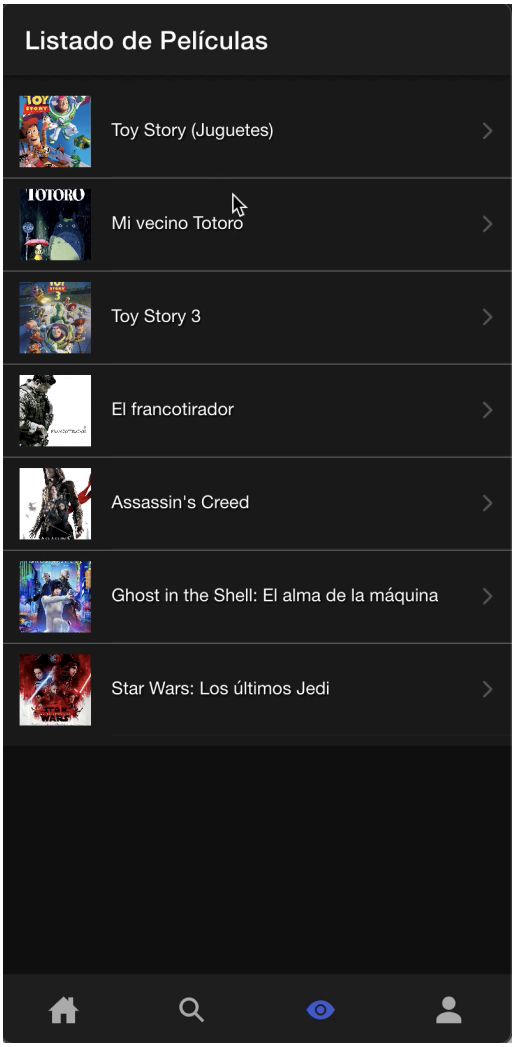

Ilustración 51: Captura de la aplicación en la pestaña de listado de películas a ver.

Con respecto a la búsqueda de películas (accesible a partir de la pestaña de la lupa) se pueden buscar las diferentes películas para obtener información y hacer seguimiento cómo se puede comprobar en la Ilustración 51.

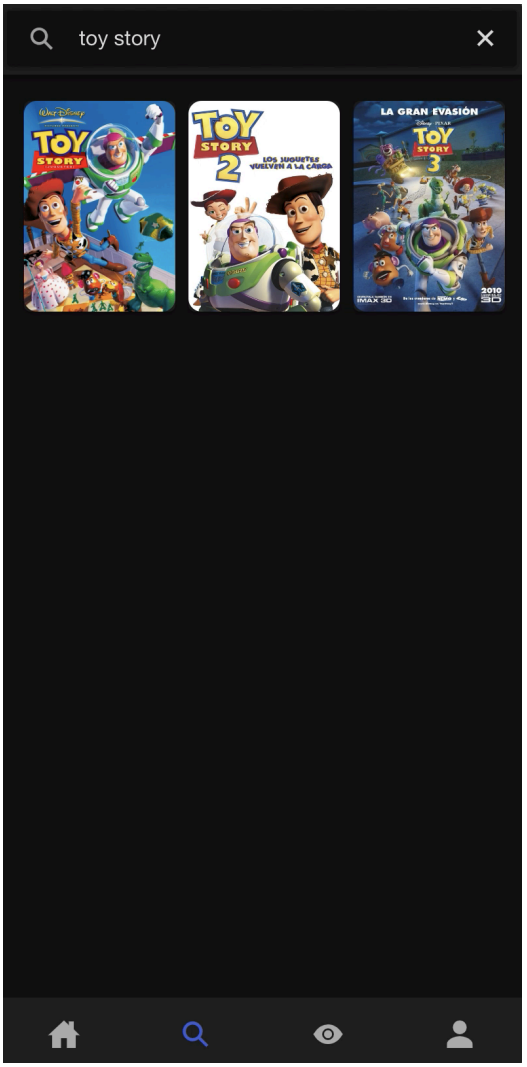

Ilustración 52: Captura de la aplicación de la pestaña de búsquedas.

Si nos fijamos, en todas estas capturas de pantalla vemos cómo se muestran diferentes películas, pero si pinchamos en cualquiera de ellas, obtendremos una información más detallada de cada una, siguiendo la apariencia de la Ilustración 52.

En esta muestra se puede apreciar información como la descripción de la película, el año de publicación, los géneros concretos que abarca y muy importante, el porcentaje de coincidencia que tendrá la película para el usuario con respecto a sus gustos, siempre y cuando esta película no haya sido valorada en cuyo caso no muestra el porcentaje si no la propia valoración del 1 al 5 que el usuario le dió.

Además visualizamos tres botones con diferentes funcionalidades, cómo el de añadir al listado de películas a ver, marcar cómo películas vista o valorar película si no se valorado antes, respectivamente por ese orden. Incluso cabe la opción que la película visualizada tenga el trailer embebido de youtube o las posibles plataformas donde se puede visualizar la película de forma gratuita, alquilada o comprada.

Por último cabe mencionar que dispone de un botón en la parte superior izquierda para volver hacia atrás.

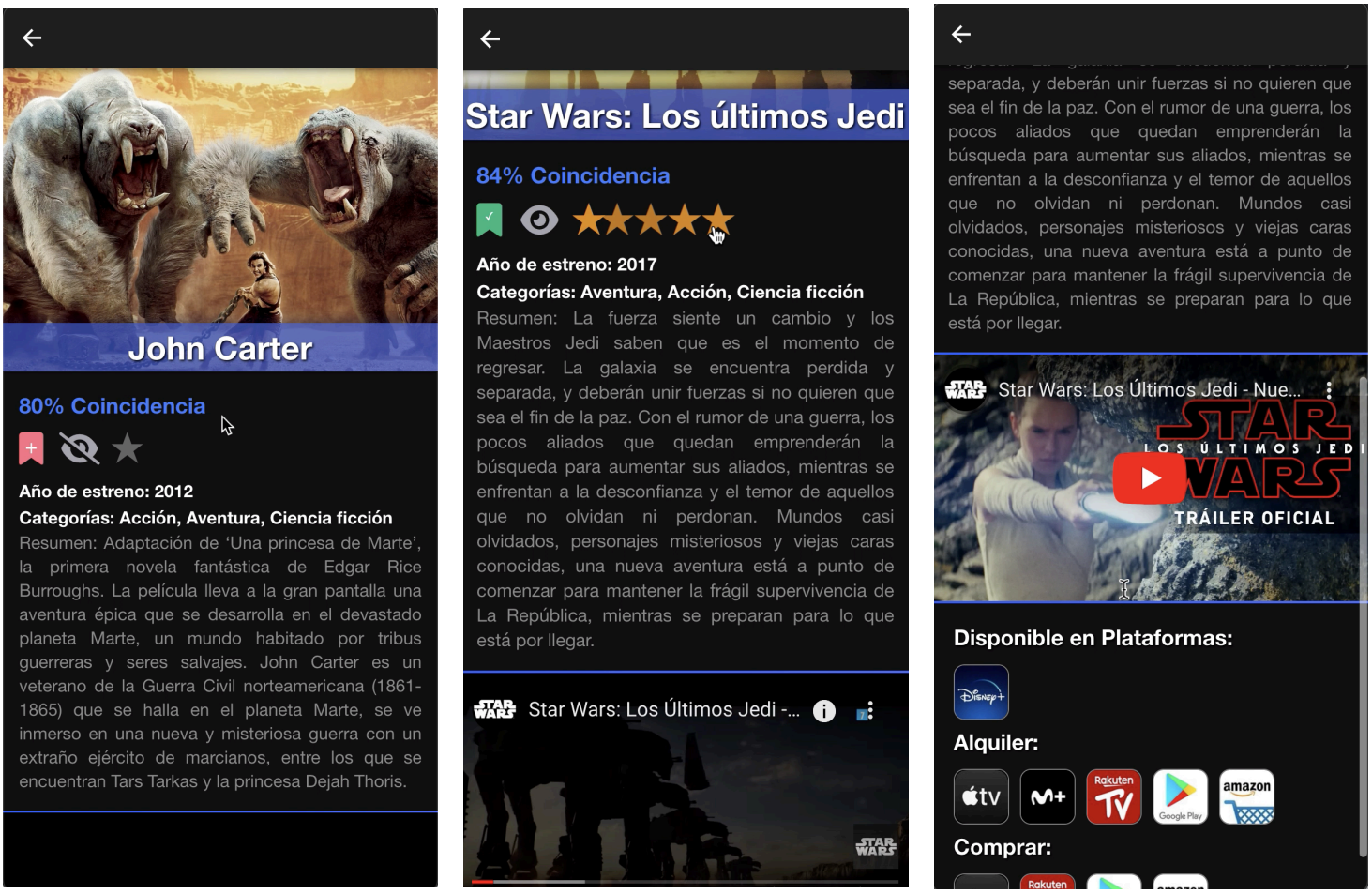

Ilustración 53: Capturas de Información detallada sobre la película.

También es muy importante para el sistema el hecho de valorar películas, ya que a medida que se genera más valoraciones, más precisas podrán ser las predicciones. Para ello encontramos otra ilustración diferente (Ilustración 52), donde se presenta una imagen de la película a valorar, y conjunto de estrellas para hacer su puntuación o un botón de omitir en caso de que el usuario no haya visto la película o no sepa dar una correcta valoración sobre esta.

Ilustración 54: Captura de valoración de películas.

Para terminar con las pestañas o tabs de la aplicación explicaremos el perfil de usuario, donde encontramos un primer botón que nos lleva a la página de valorar películas o el botón para cambiar el idioma de la aplicación, en concreto al Español, Inglés o al Francés. Lo más interesante del cambio de idioma, no es solo la conversión de los literales mostrados en las páginas de un lenguaje a otro, si no que va más allá. Cuando cambiamos de un idioma a otro, este cambio se refleja igualmente en los audios de los videos de youtube que aparecen a modo de trailer y de las plataformas donde se puede ver la película, ya que dependiendo del país la disponibilidad de las películas en las plataformas será diferente. Para el Francés el país será Francia, para el Español es España y para el Inglés EEUU. Puede observar este punto en la Ilustración 54.

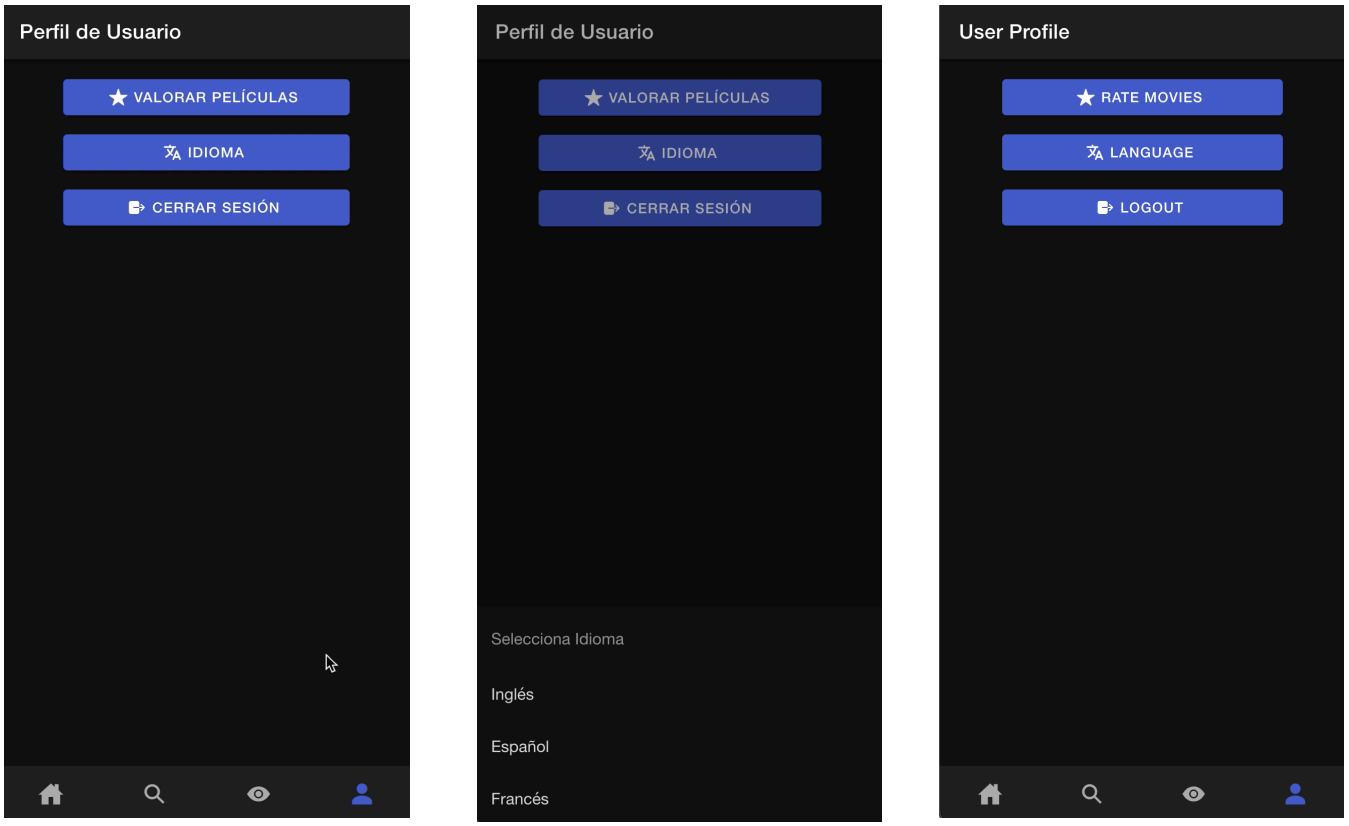

Ilustración 55: Capturas de la pestaña de perfil de usuario.

Para terminar con este manual se presentan las distintas capturas para loguearse el cual plantea un escenario familiar a la hora de loguearse, ya que sigue el patrón y las características de cualquier autenticación.

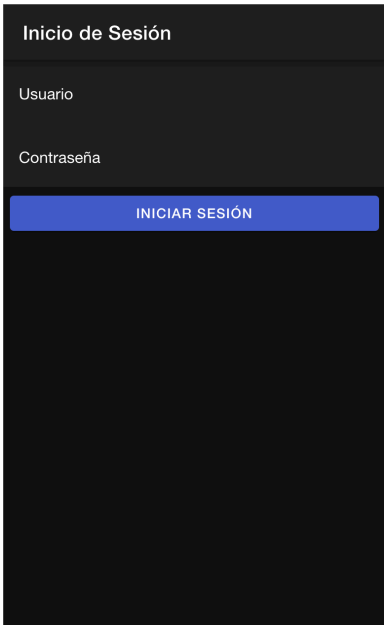

Ilustración 55: Captura de autentificación de la aplicación.

10. Bibliografía

[1] Simon Kemp (2022). *Digital Report España 2022*. [En Línea]. Available: https://wearesocial.com/es/blog/2022/02/digital-report-espana-2022-nueve-de-cada-diez-espanoles-usan-las-redes-sociales-y-pasan-cerca-de-dos-horas-al-dia-en-ellas/. [Último acceso: 1 Enero 2024].

[2] M. J. Barranco, L.G. Pérez, L. Martínez (2006). *Un Sistema de Recomendación Basado en Conocimiento con Información Lingüística Multigranular*. Universidad de Jaén, pp. 647.

[3] López, V. F. (2013). *Técnicas eficientes para la recomendación de productos basadas en filtrado colaborativo.* Doctoral dissertation, Universidade da Coruña.

[4] Herlocker, J. L., Konstan, J. A., Terveen, L. G., & Riedl, J. T. (2004). *Evaluating collaborative filtering recommender systems*. ACM Transactions on Information Systems (TOIS), 22(1), 5-53.

[5] Koren, Y., Bell, R., & Volinsky, C. (2009). *Matrix factorization techniques for recommender systems*. Computer, 42(8), 30-37.

[6] John Daniel A. B. (2013). *Agile methodologies in the development of applications for mobile devices*. Revista de Tecnología, Journal Technology, pp. 111-124.

[7] Scrum: Revisión de las Iteraciones. [En Línea]. Available: http://modeladowebapp.blogspot.com/2015/04/scrum-revision-de-las-iteraciones.html [Último acceso: 1 Enero 2024].

[8] Guía de principiantes para Scrum en 7 pasos. [En Línea]. Available: https://medium.com/forecast-en-español/guía-de-principiantes-para-scrum-7-pasos-127e30cfd585 [Último acceso: 1 Enero 2024].

[9] Simon Funk. *Stochastic gradient descent optimization of Equation 2* [En Línea]. Available: https://sifter.org/simon/journal/20061211.html [Último acceso: 1 Enero 2024].

[10] Simon Bennet, Ray Farmer, Steve McRobb (2006). *Análisis y diseño orientado a objetos de sistemas usando UML*. Editorial McGraw-Hill.

[11] Agile User Story Mapping Tool [En Línea]. Available: https://www.visual-paradigm.com/features/agile-user-story-mapping-tool/ [Último acceso: 1 Enero 2024].

[12] Intellij IDEA Ultimate edition. [En Línea]. Available: https://www.jetbrains.com/idea/var/buy/#commercial?billing=monthly [Último acceso: 1 Enero 2024].

[13] Visual Paragim store. [En Línea]. Available: https://www.visual-paradigm.com/shop/vp.jsp

[Último acceso: 1 Enero 2024].

[14] Boletín del Estado (2018), *XVII Convenio colectivo estatal de empresas de consultoría y estudios de mercado y de la opinión pública*. [En Línea]. Available: https://www.boe.es/boe/dias/2018/03/06/pdfs/BOE-A-2018-3156.pdf
[Último acceso: 1 Enero 2024].

[15] Ivar Jacobson, Martin Griss, Patrik Jonsson. Pearson, (2000). *Modelado y diseño de sistemas con UML*.

[16] Descripción de REST API [En línea]. Available: https://www.redhat.com/es/topics/api/what-is-a-rest-api
[Último acceso: 1 Enero 2024].

[17] Morisetti, Juan (2023). *Entendiendo los microservicios: La arquitectura que usan las grandes empresas del mundo* (p. 31).

[18] Documentación API IMDb [En línea]. Available: https://imdb-api.com/api/#Title-header
[Último acceso: 1 Enero 2024].

[19] Pablo Fernández. *Docker y microservicios*. [En línea]. Available: https://www.hiberus.com/crecemos-contigo/docker-y-microservicios/
[Último acceso: 1 Enero 2024].

[20] Arquitectura de una aplicación con Ionic [En línea]. Available: https://ajgallego.gitbook.io/ionic/arquitectura
[Último acceso: 1 Enero 2024].

[21] Dataset MovieLens 1M. [En Línea]. Available: https://grouplens.org/datasets/movielens/1m/
[Último acceso: 1 Enero 2024].

[22] Seguridad web: una visión general de SOP,CORS y CSRF. [En Línea]. Available: https://maddevs.io/blog/web-security-an-overview-of-sop-cors-and-csrf/#csrf
[Último acceso: 1 Enero 2024].

[23] A Gentle Introduction to k-fold Cross-Validation. [En Línea]. Available: https://machinelearningmastery.com/k-fold-cross-validation/
[Último acceso: 1 Enero 2024].

[24] API de la librería matemática de apache Commons Math. [En Línea]. Available: https://commons.apache.org/proper/commons-math/javadocs/api-3.6.1/index.html.
[Último acceso: 1 Enero 2024].

[25] iOS Development. [En Línea]. Available: https://ionicframework.com/docs/v6/developing/ios
[Último acceso: 1 Enero 2024].

[26] Android Development. [En Línea]. Available: https://ionicframework.com/docs/v6/developing/android [Último acceso: 1 Enero 2024].