ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
Grado de Ingeniería Informática
Mención en Sistemas de Información

DESARROLLO DE UN SISTEMA DE RECOMENDACIÓN MÚSICAL

DEVELOPMENT OF A MUSIC RECOMMENDER SYSTEM

Realizado por
Javier Vázquez García
Tutorizado por
Carlos Rossi Jiménez
Departamento
Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, SEPTIEMBRE DE 2018

Fecha defensa:
El Secretario del Tribunal

Resumen

Español

Los sistemas de recomendación han surgido y evolucionado de forma paralela al desarrollo de la web, nutriéndose de información de distintas fuentes con objeto de realizar un análisis exhaustivo para ser capaces de elaborar y presentar una recomendación personalizada a usuarios individuales. Este trabajo estudia diferentes artículos y trabajos de investigación sobre sistemas de recomendación, además de desarrollar una aplicación con un sistema de recomendación para música orientado a facilitar la creación de listas de reproducción generadas automáticamente y adaptada al contexto de cada usuario. Para la aplicación a desarrollar se ha elegido una arquitectura web basada en tecnologías Java. La plataforma consta de una aplicación web para la gestión de artistas, álbumes, canciones, géneros y listas de reproducción. Además, permitirá a los expertos configurar el motor de recomendación. La aplicación cubrirá la funcionalidad para los perfiles de usuario, administrador y experto (musical), y se basa en la gestión del propio perfil de usuario, en la consulta de canciones, artistas, álbumes, géneros y listas de reproducción en el sistema, además de mostrar las recomendaciones solicitadas por el usuario en base a una serie de parámetros concretos, usando para ello el motor de recomendación basado en implicaciones. También aborda una fase de evaluación. El trabajo se ha realizado con técnicas y prácticas ágiles de gestión y desarrollo de proyectos, así como con técnicas de análisis y diseño.

Palabras claves: Aplicación web, Sistemas de recomendación musical, Motor de recomendación, Java

English

The recommendation systems have arisen and evolved in parallel to the development of the web, drawing on information from different sources to perform a thorough analysis to be able to develop and submit a personalized recommendation for individual users. This project studies different articles and research work on recommendation systems, in addition to developing an application with a recommendation system for music oriented to facilitate the creation of automatically generated playlists adapted to the context of each user. For the application developed, a web architecture based on Java technologies has been chosen. The platform consists of a web application for the management of artists, albums, songs, genres and playlists. In addition, it will allow experts to configure the recommendation engine. The application will cover the functionality for profiles of type user, administrator and expert (musical), and is based on the management of the user profile itself, in the consultation of songs, artists, albums, genres and playlists in the system, in addition to show the recommendations requested by the user based on a series of specific parameters, using the recommendation engine based on implications. It also addresses an evaluation phase. The work has been done with agile management techniques and project development, as well as analysis and design techniques.

Keywords: Web application, Music recommendation systems, Recommendation engine, Java

Índice

Capítulo 1 – Introducción

Este es un Trabajo de Final de Grado desarrollado en la línea de sistemas de recomendación. A menudo, los clientes de servicios musicales (véase Spotify, Apple Music, Deezer, Google Play Music, etc.) se sienten abrumados ante la gran cantidad de posibles canciones, artistas y listas de reproducciones entre las que pueden elegir, ocurriendo este fenómeno incluso dentro de una misma aplicación, creándose la necesidad de un sistema que les ayude a seleccionar qué canciones escuchar, o, al menos, a acotar significativamente el número de resultados de las búsquedas.

Partiendo de la base que el tiempo de los usuarios es muy valioso y escaso, cada vez que eligen pasar varios minutos de su vida escuchando una canción, están tomando una decisión que puede conllevar una imagen negativa del sistema de recomendación si este no sugiere algo adecuado. Por ello, es importante mejorar estos sistemas de recomendación para que hagan al usuario sugerencias cada vez más precisas.

En la actualidad este tipo de sistemas tienen mucho margen de mejora, siendo esta una de las principales razones para realizar este trabajo. En este sentido, se construye una herramienta ágil y sencilla de usar, consistente en una aplicación que incorpora un sistema de recomendación para música.

Fundamentalmente, aparte de la típica gestión de artistas, álbumes, canciones y listas de reproducciones, la aplicación incorpora un sistema de recomendación orientado a facilitar la creación de listas de reproducción basadas en el contexto en el que se encuentra el usuario. En este tipo de recomendación, las listas de reproducción recomendadas son “curadas” o refinadas a partir de conocimiento de expertos humanos.

Para obtener dichas recomendaciones, el sistema de recomendación utiliza un motor basado en implicaciones, desarrollado por el grupo de investigación SICUMA [14]. El motor de recomendación utilizado para este proyecto esta implementado como prototipo de investigación y se ha integrado y adaptado para ser utilizado en este sistema de recomendación.

Para el desarrollo de la aplicación, se ha llevado a cabo una fase de análisis y diseño, donde se han realizado la especificación de requisitos, el diseño de diferentes diagramas, la creación de maquetas de usuario y recolección de datos, entre otras tareas, además del desarrollo en sí y la realización de diferentes pruebas.

Por otra parte, se ha realizado una investigación acerca del estado del arte actual de los sistemas de recomendación, analizando cuando son las principales aproximaciones y las características de cada una de ellas.

Capítulo 2 – Estado del arte

En este capítulo se analiza el estado actual de la tecnología relacionada con los aspectos principales de este trabajo fin de grado. En primer lugar, se presentan los servicios musicales on-line más relevantes, y posteriormente se analizan los avances actuales en cuanto a sistemas de recomendación musical.

2.1 Servicios musicales

Los servicios de contenidos musicales son aquellos que nos permiten reproducir productos musicales remotamente. Normalmente son canciones, las cuales el usuario final reproduce desde la herramienta del proveedor del servicio, ya sea una versión web o una aplicación para móvil u ordenador. Estos servicios son una alternativa con numerosas ventajas a la tradicional descarga de archivos, un proceso en el cual el usuario final obtiene el archivo completo para escucharlo. En el caso de los servicios musicales en línea, un usuario final puede usar el reproductor multimedia del servicio para comenzar a reproducir la canción antes de que se haya transmitido todo el archivo.

Estos servicios para escuchar audio digital están disponibles en diversas plataformas, desde un teléfono inteligente, una tablet, un ordenador de escritorio o un sistema de entretenimiento doméstico, como una televisión inteligente.

Un requisito indispensable común a todos ellos es que funcionan a través de Internet, aunque algunos servicios dan la posibilidad de hacer una copia offline para cuando no se disponga de conexión.

En general, este tipo de servicios cuentan con millones de canciones, podcasts y videos, además de otros contenidos de artistas de todo el mundo.
Los más populares y usados internacionalmente son Spotify, Apple Music, Tidal, Amazon Music Unlimited y Deezer, entre otros.

El mayor servicio por número de usuarios por el momento es Spotify [7], con casi 160 millones. Para profundizar sobre el tema veamos de forma general que es lo que ofrece esta plataforma en concreto. Para empezar, las funciones básicas, como escuchar música, son totalmente gratis, pero también se tiene la opción de mejorar la cuenta con Spotify Premium, cuya mayor ventaja es la eliminación de anuncios. De cualquiera de las dos maneras se puede, entre otras opciones:

- Elegir lo que se quiere escuchar manualmente con Explorar. Podemos observar como se muestra esta pantalla en la Figura 1 : Explorar, en Spotify.
- Gestionar diversas listas de reproducción.
- Ver lo que escuchan amigos, artistas y famosos.
- Crear emisoras de radio basadas en los gustos del usuario.
- Recibir recomendaciones en funciones personalizadas, como Descubrimiento semanal, Radar de Novedades y Daily Mix, basado en el historial de escucha, entre otros parámetros. [8]

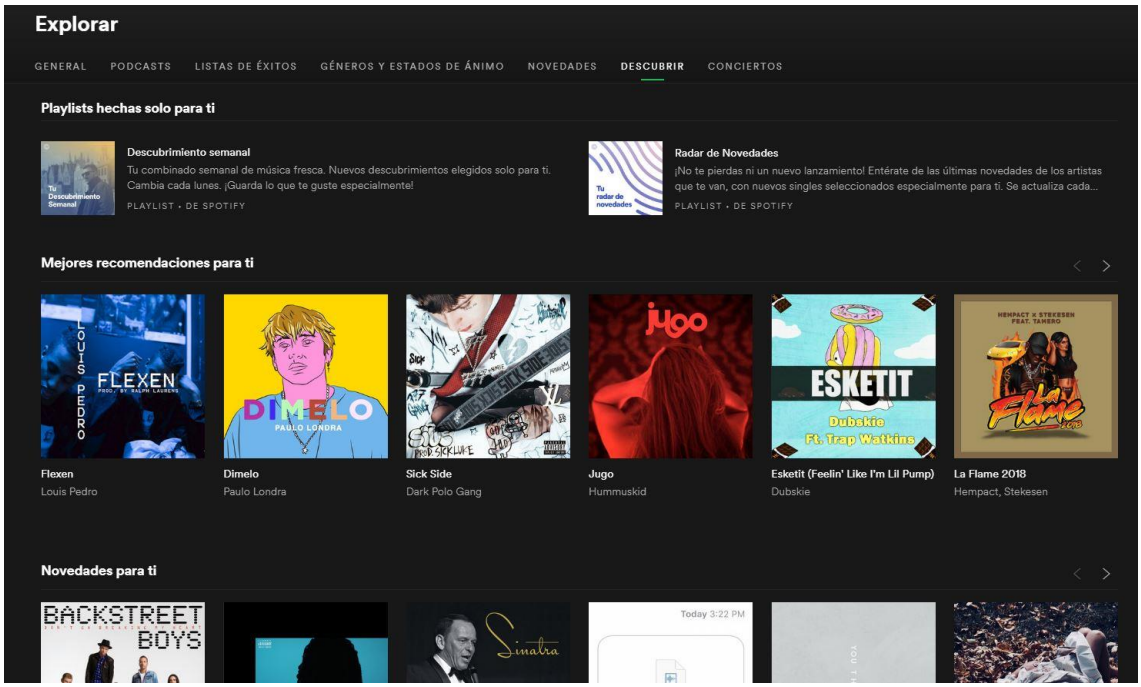

Figura 1 : Explorar, en Spotify

Esta última opción es una de las que nos interesa resaltar, pues está intrínsecamente relacionada con el objetivo de este trabajo de final de grado.

Otro de los servicios más populares es Apple Music, el cual es un servicio de reproducción de música y video desarrollado por Apple Inc. Los usuarios seleccionan música para transmitirla a su dispositivo bajo demanda, pudiendo escuchar también listas de reproducción existentes y creadas por el mismo. El servicio también incluye la estación de radio de Internet Beats 1 (Figura 2), que transmite en vivo a más de 100 países las 24 horas del día.

Figura 2 : Estación de radio Beats 1, de Apple Music

Originalmente creado como un servicio de música, Apple Music comenzó a expandirse en su faceta de plataforma de video en 2016.

Apple Music ha ganado rápidamente popularidad tras su lanzamiento a finales de junio de 2015, superando los 10 millones de suscriptores después de solo seis meses. Actualmente tiene alrededor de 40 millones de suscriptores.

Otro competidor es Tidal, un servicio musical, para el que se necesita una suscripción, que combina audio sin pérdida de calidad y videos musicales de alta definición con una editorial selecta. El servicio tiene más de 40 millones de pistas y 150 mil videos musicales, ofreciendo además la posibilidad de escuchar listas de reproducción existentes y creadas por el usuario.

Los vídeos y, sobre todo, la emisión de actuaciones en directo es una de las principales diferencias de Tidal con el resto de las plataformas, así como el contenido en exclusiva.

Por otro lado, Amazon Music Unlimited es un servicio de música de suscripción premium que cuenta con un catálogo de más de 50 millones de canciones, siendo este el más extenso de los servicios analizados. Además, tiene ofertas para aquellos usuarios que ya son parte de Amazon Prime.

Por último, cabe destacar a Deezer. Es el único servicio musical junto a Spotify que ofrece un servicio gratuito a cambio de anuncios, siendo estos eliminados en la modalidad de pago.

En la siguiente tabla podemos observar las características principales de cada uno de estos servicios:

	Catalogo canciones	Usuarios versión de pago	Usuarios versión gratuita	Precio Suscripción	Contenido extra
Spotify	30 millones	71 millones	86 millones	9,99 euros	Podcasts, vídeos
Apple Music	45 millones	40 millones	No	9,99 euros	Radio, vídeos, documentales
Tidal	40 millones	4.2 millones	No	9,99 euros	Vídeos, actuaciones en vivo, contenido en exclusiva
Amazon Music	50 millones	16 millones	No	9,99 euros	No
Deezer	40 millones	9 millones	3 millones	9,99 euros	Vídeos

Figura 3 : Tabla servicios musicales

También cabe destacar que todas ofrecen servicio offline[1], recomendaciones personalizadas de canciones y listas de reproducción, compatibilidad con un gran número de dispositivos y ausencia de publicidad en el servicio de pago

2.2 Sistemas de recomendación

Ante la gran cantidad de posibles canciones, artistas y listas de reproducciones entre las que pueden elegir los usuarios, surge la necesidad de un sistema de recomendación que ayude a seleccionar qué canciones escuchar, o, al menos, a acotar significativamente el número de resultados de las búsquedas. Partiendo de la base que el tiempo de los usuarios es muy valioso y escaso, cada vez que eligen pasar varios minutos de su vida escuchando una canción, están tomando una decisión que puede conllevar una imagen negativa del sistema de recomendación, y por ende del sistema de música en línea, si este no sugiere algo adecuado. Por ello, es importante mejorar estos sistemas de recomendación para que hagan al usuario sugerencias cada vez más precisas.

Los sistemas de recomendación han surgido y evolucionado de forma paralela al desarrollo de la web, nutriéndose de información de distintas fuentes con objeto de realizar un análisis exhaustivo, y así ser capaces de elaborar y presentar una recomendación personalizada a usuarios individuales.

Para tener una buena comprensión de cuál es el estado de arte actual en cuanto a recomendación musical, debemos repasar una serie de conceptos básicos.

Una lista de reproducción es un conjunto de canciones ordenadas, con la intención de adecuarse a su escucha en una situación o contexto concreto, como por ejemplo para salir a correr o para una fiesta. En general, se usa este tipo de listas cada vez que se utiliza un servicio de música en línea, en vez de seleccionar las canciones de una en una.

Gracias a la disponibilidad existente de metadatos sobre canciones, podemos basarnos en estos a la hora de automatizar la creación de listas de reproducción. Hay diversas maneras de afrontar este problema, las cuales varían fundamentalmente en qué tipo de datos y algoritmos utilizan.

Las listas de reproducción tienen en común que su construcción está determinada por el conjunto de canciones disponibles y las características del objetivo de la lista de reproducción. Estas características pueden ser el género, el estado de ánimo, el tempo de las canciones, la popularidad, etc.

Para automatizar esta tarea, una herramienta de generación de listas de reproducción debe incluir un algoritmo que evalúe los objetivos previstos y las características de la lista de reproducción que se va a generar. También necesitamos un algoritmo que determine si una canción es adecuada para la lista de reproducción que vamos a realizar.

[1] Servicio offline: Se refiere a la capacidad de guardar canciones de forma local en tu dispositivo para reproducirlas sin conexión a internet desde la aplicación.

Esta tarea no es baladí, pues hay que tener en cuenta numerosos factores y características. Por ejemplo, el gusto y el contexto de los oyentes son dos factores determinantes en la calidad de una lista de reproducción para un usuario determinado, ya que dos usuarios pueden considerar la misma lista de reproducción de manera totalmente dispar, aun siendo escuchada en un mismo contexto. Por tanto, debemos identificar correctamente todos los factores y características que se necesitan para generar una lista de reproducción adecuada para el usuario.

Geoffray Bonnin y Dietmar Jannach [2] resumen el problema de generación de una lista de reproducción de la siguiente manera:

“Dado un grupo de canciones, una base de datos de conocimiento, y algunas características objetivo de la lista de reproducción a crear, se quiere obtener una búsqueda de canciones que cumplan las características del objetivo de la mejor manera posible”.

Otro punto a reseñar es que el generador de listas de reproducción debería ser capaz de adaptar dinámicamente sus recomendaciones a lo que le apetece escuchar al usuario en un momento determinado. Para ello, se incorporan opciones en el reproductor para omitir una canción o para dar un “Me gusta”, entre otras. Por ejemplo, cuando en Spotify un usuario esta escuchado la “radio de canción” o “radio de artista”, que no es más que una lista de reproducción basada en una canción o en un artista, al escuchar una canción de esa lista tiene a su alcance dos opciones: “Me gusta” y “No me gusta”. Al pulsar cualquiera de ellas se tiene en cuenta la opción seleccionada para las siguientes canciones a generar en la lista de reproducción y para las futuras listas de reproducciones a generar; es decir, se añade la información a esa base de datos de conocimiento que hemos mencionado anteriormente.

Por otra parte, si se analiza pormenorizadamente los registros de escucha se obtiene una gran cantidad de información sobre el gusto y las preferencias de un usuario concreto. Se puede inferir la frecuencia de escucha de una determinada canción, la asiduidad de ciertas acciones, como por ejemplo cambiar de canción, o las anteriormente mencionadas “Me gusta” o “No me gusta”, entre otras.
Una vez analizada y procesada dicha información, permite saber la idoneidad de una pista dentro de una lista de reproducción o si se ha reproducido una canción en el contexto adecuado.
Todo esto forma un valioso recurso para la correcta generación personalizada de listas de reproducción.

Los servicios musicales permiten a los usuarios crear sus propias listas de reproducción y, además, seguir la que otros usuarios han compartido. Las que tienen un número significativo de seguidores se puede presuponer que obedecen ciertos criterios de calidad y homogeneidad, como el género musical, el estado de ánimo, etc. Estas sutiles relaciones y patrones se pueden usar para generar una lista de reproducción personalizada.

Una vez analizado estos conceptos básicos, pasamos a analizar cuáles son las aproximaciones más relevantes propuestas en cuanto a sistemas de recomendación musicales.

J. Aucouturier y F. Pachet [3], entre otros, proponen métodos basados en el contenido, los cuales extraen las características directamente de la señal de audio, calculando la similitud entre canciones basándose en las características, y recomendando las canciones que son similares a los usuarios. Por ejemplo, en Spotify, al seleccionar en una canción la acción “Ir a radio de la canción” crea una lista de reproducción de canciones con características similares a la seleccionada.

G. Adomavicius y A. Tuzhilin[1], entre otros, han propuesto métodos basados en el contexto, es decir, métodos que se basan en la ubicación, día, hora, etc. A la hora de realizar recomendaciones, ya que los usuarios prefieren diferentes canciones, y por ende listas de reproducción, para los diferentes momentos del día: trabajar, estudiar, entrenar, relajarse de dormir, etc.

J. Wang y A.P. de Vries [4], entre otros investigadores, han propuesto métodos de filtrado colaborativo, los cuales usan comentarios explícitos o retroalimentación implícita para rastrear los hábitos de escucha del usuario.

Ke Ji, Runyuan Sun, Wenhao Shu y Xiang Li [5] han propuesto un modelo dinámico basado en un modelo de Markov incrustado, el cual tiene en cuenta el cambio que experimentan los usuarios con el paso del tiempo.
En concreto tienen en cuenta tres rasgos que afectan al interés del usuario con el paso del tiempo; el efecto a largo plazo, el efecto a corto plazo y el efecto en cada sesión. Integrando estos tres efectos les permite hacer un seguimiento de los cambios de intereses musicales en un usuario conforme pasa el tiempo y hacer recomendaciones precisas, obteniendo mejores resultados que otras aproximaciones basadas en los métodos de Markov.

Kuang Mao, Gang Chen, Yuxing Hu y Luming [6] han propuesto un sistema de recomendación musical usando un modelo de calidad basado en grafos.
Para ello, primero necesitan calcular qué canción es más atractiva dadas dos canciones cualesquiera, lo cual realizan basándose en la calificación de los usuarios, aunque siempre teniendo en cuenta la fiabilidad de la persona que emite el voto. Además, las recomendaciones de aquellos usuarios con mayor reputación se ponderan de forma mayor.
Una vez que se han descubierto estas relaciones de preferencia entre canciones, se crea un grafo de preferencias para modelarlas con el objetivo de poder ser utilizadas adecuadamente. Para cuantificar si una canción es adecuada para la recomendación, usan una medida de probabilidad. Una vez obtenida dicha probabilidad para cada canción, con un algoritmo de ranking de canciones, realizan una recomendación de canciones.

Además de mejorar los resultados con respecto a otros algoritmos, una de las mayores ventajas de este sistema es que no existe el problema de arranque en frío[2].

En la siguiente tabla podemos observar las diversas aproximaciones mencionadas de forma resumida:

[2] Arranque en frio: Esta expresión se refiere al problema que ocurre cuando un sistema no puede extraer inferencias para los usuarios o temas sobre los que aún no ha reunido suficiente información.

Técnica de recomendacion	**Autores**	**Características**	**Observaciones**
Métodos basados en el contenido	J. Aucouturier, F. Pachet	Extraen la información de la señal de audio	Generan listas de reproducción basadas en un género o en una canción
Métodos basados en el contexto	G.Adomavicius, A. Tuzhilin	Se basan en la ubicación, hora, actividad, etc.	Generan listas de reproducción basadas en el contexto del usuario (hacer deporte, relajarse, etc.)
Métodos de filtrado colaborativo	J. Wang, A.P. de Vries, M.J.T. Reinders,	Se basan en las recomendaciones de otros usuarios, además en las propias acciones del usuario	Generan listas de reproducción basadas en canciones populares para otros usuarios con gustos similares
Modelo de Markov incrustado	Ke Ji, Runyuan Sun, Wenhao Shu y Xiang Li	Tiene en cuenta el cambio que experimentan los usuarios con el paso del tiempo.	Generan listas de reproducción adecuadas para un usuario que evolucionan en el tiempo, adaptándose a sus cambios
Modelo de calidad basado en grafos	Kuang Mao, Gang Chen, Yuxing Hu y Luming	Establece relaciones de preferencia entre canciones modelándolas en un grafo	Generan listas de reproducción con recomendaciones muy precisas.

Figura 4 : Tabla servicios musicales

Capítulo 3 - Descripción general del sistema

El sistema consiste en una aplicación con un sistema de recomendación para músca orientado a facilitar la creación de listas de reproducción generadas automáticamente y adaptadas a los gustos de cada usuario.

Fundamentalmente, aparte de la típica gestión de artistas, álbumes, canciones, géneros y listas de reproducciones, la aplicación incorporará un sistema de recomendación orientado a facilitar la creación de “curated playlists”.

Las denominadas “curated playlists” son listas de reproducción afinadas por un experto musical con un objetivo concreto.

Se contemplan tres perfiles de usuario en la aplicación. Tenemos el perfil de usuario registrado (en otras palabras, el usuario que utilizará la aplicación para oir música y gestionar sus playlists), el experto musical y, por último, el perfil de administrador del sistema.

A continuación, se detallan a alto nivel las funcionalidades disponibles en el sistema y a qué perfiles le corresponde cada funcionalidad.

La gestión de artistas permitirá añadir un nuevo artista al sistema por parte del administrador, al igual que eliminarlo, además de ver, modificar y eliminar todas las características de un artista. Un usuario registrado podrá ver los artistas disponibles, y los álbumes y canciones asociados al mismo.

En cuanto a la gestión de álbumes, permitirá añadir un nuevo álbum al sistema por parte del administrador, al igual que eliminarlo, además de ver, modificar y eliminar todas las características de un álbum. Un usuario registrado podrá ver los álbumes disponibles, y los artistas y canciones asociados al mismo.

Respecto a la gestión de canciones, el sistema permitirá añadir una nueva canción al sistema por parte del administrador, al igual que eliminarla, además de ver, modificar y eliminar todas las características de una canción. Un usuario registrado podrá ver los artistas disponibles, y los álbumes y canciones asociados al mismo.

Sobre la gestión de listas de reproducción, el sistema permitirá a un usuario registrado crear una lista de reproducción, al igual que eliminarla, acceder a la misma para ver su contenido, añadir y eliminar canciones, y modificar el orden de canciones en la lista de reproducción.

Asimismo, el sistema ofrecerá una opción al usuario para crear una lista de reproducción de forma automática, basándose en ciertos criterios introducidos. Para ello, el sistema se valdrá del motor de recomendación, cuyo funcionamiento se describe más adelante.

El administrador del sistema, además de las funcionalidades anteriormente descritas, podrá acceder a las relativas a la gestión del sistema, como por ejemplo la gestión de usuarios.

En cuanto al perfil de experto musical, tendrá acceso al sistema de recomendación, pudiendo realizar ajustes mediante la gestión de implicaciones (entendidas como reglas) y, por lo tanto, siendo capaz de alterar las sugerencias que proporciona el motor de recomendación. Además, tendrá las mismas capacidades que el administrador a la hora de gestionar los artistas, álbumes y canciones.

Para la aplicación a desarrollar se ha elegido una arquitectura web basada en tecnologías Java. La plataforma constará de una aplicación web para la gestión de artistas, álbumes, canciones, géneros y listas de reproducción. Además, permitirá a los expertos configurar el motor de recomendación. La aplicación, además del perfil de usuario registrado., cubrirá la funcionalidad para los perfiles de administración y experto (musical). Ofrecerá funcionalidades como la gestión del propio perfil de usuario, la consulta de canciones, artistas, álbumes, géneros y listas de reproducción en el sistema, además de mostrar las recomendaciones solicitadas por el usuario en base a una serie de parámetros concretos, usando para ello el motor de recomendación basado en implicaciones.

Un usuario deberá registrarse al iniciar sesión por primera vez; se le mostrará un formulario donde se debe introducir el correo electrónico, el nombre, un ID de usuario y una contraseña.

El sistema de recomendación utilizará un motor de recomendación, desarrollado por el grupo de investigación SICUMA [14], el cual esta implementado como prototipo de investigación y se ha integrado y adaptado para este TFG. El sistema desarrollado se comunica con el motor de recomendación a través de una API.

Mediante la API del motor de recomendación se gestionan tres tipos de entidades principales y dos secundarias.

Las entidades principales son las que realmente están involucradas en el sistema de recomendación y son:

- Items: Elementos o recursos a recomendar por el sistema.

- Rules: Reglas o implicaciones del sistema.

- Profiles: Perfiles o contextos del sistema.

Las entidades secundarias son las que sirven para dar acceso a la API o para definir alguna funcionalidad adicional:

- Users: Usuarios que tienen acceso a la API.

- Track: Log o huellas de las acciones que se realizan con la API

Con la operación getRecommendations vamos a poder obtener recomendaciones del motor de recomendación teniendo en cuenta:

- El perfil de usuario / contexto.

- El conjunto de recursos a tener en cuenta (opcional).

- El conjunto de reglas a aplicar (opcional)

El motor de recomendación está basado en implicaciones. Una implicación es un concepto fundamental en la lógica, que describe la relación entre las declaraciones que son verdaderas cuando una afirmación se sigue lógicamente de una o más afirmaciones. Un argumento lógico válido es aquel en el cual la conclusión está implicada por las premisas, porque la conclusión es la consecuencia de las premisas. La implicación es necesaria y formal, a modo de ejemplos que explican con pruebas formales y modelos de interpretación.

La implicación relaciona una causa con un efecto, y en la lógica proposicional se puede escribir formalmente como: A → B, siendo A una causa o conjunto de causas, y B es el efecto o conjunto de efectos, que resultan de esas causas. En otras palabras, B es una conclusión lógica de A. En nuestro sistema, se define a A como el antecedente y a B como el consecuente.

En el sistema de recomendación, cada uno de los ítems son los elementos a recomendar por el sistema, las reglas representan las implicaciones del sistema y los perfiles son los contextos. En nuestro caso, los ítems son canciones, las cuales tienen diferentes atributos descriptivos que se evalúan entre 0.0 a 1.0.

Para cada uno de los contextos se determinan unas reglas (implicaciones), las cuales indican al motor de recomendación que ítems son adecuados para un contexto determinado. Un contexto es un conjunto de circunstancias que rodean una situación y sin las cuales no se puede comprender correctamente. Trasladado a nuestro sistema, un contexto puede describir el estado de ánimo de un usuario, el momento del día en el que se encuentra, la actividad que esté realizando en un momento determinado, etc. Deberá ser un experto el que introduzca en el sistema los contextos relevantes que ayuden a recomendar música de forma eficaz.

Las reglas (implicaciones) que se aplican a cada uno de los contextos se establecen tomando en cuenta diferentes factores, y serán los expertos musicales los encargados de retocarlas para que las recomendaciones sean las más adecuadas a cada situación. Para obtener dichas recomendaciones, un usuario debe describir su contexto, valorando un contexto concreto entre 0.0 y 1.0, significando esto cuanto representa ese contexto su situación actual. Una vez descrito el contexto, en base a las implicaciones introducidas en el sistema para ese contexto determinado, se le recomendaran los ítems que son adecuados para su contexto actual.

Un ejemplo concreto sería un usuario al azar, que se encuentra en un contexto de actividad física intensa y, por ende, describe así su situación, y al cual se le recomienda una serie de canciones con unas características comunes, por ejemplo, con un elevado valor en el atributo “energía”. La razón por la que se le recomienda estos ítems es porque en el sistema se ha introducido una regla para dicho contexto, la cual especifica que para ese contexto todas las canciones

deben de tener como mínimo un valor 0.8 en el atributo energía, y, por lo tanto, se logra que el usuario solo obtenga canciones que cumplan dicha condición.

Capítulo 4 - Catálogo de requisitos

A lo largo de este capítulo se hace un análisis exhaustivo de los requisitos de la aplicación. En el mismo se describe todas las funcionalidades que presenta la aplicación y la utilidad de estas.

4.1 Requisitos funcionales

RF01 - Registrarse en la aplicación.

- Al pulsar el botón “Regístrate”, se mostrará un formulario donde se debe introducir un ID de usuario, el nombre, un correo electrónico y una contraseña. Estos dos últimos serán los que se utilizarán para iniciar sesión.

RF02 - Iniciar sesión en la aplicación.

- Se compararán los datos de inicio de sesión en la base de datos para conceder el acceso o no. Estos serán: correo electrónico como ‘Email’ y contraseña como ‘Contraseña’.

RF03 - Cerrar sesión en la aplicación.

- Al pulsar el botón “Cerrar sesión” se cerrará la sesión la aplicación, siendo necesario volver a introducir las credenciales la siguiente vez que se abra la aplicación.

RF04 - CRUD[3] lista de reproducción.

- Al pulsar el botón “Crear lista de reproducción”, el sistema pedirá al usuario un nombre. Una vez introducido, se creará una lista de reproducción vacía con el nombre que se ha escrito.

- Al pulsar en una lista de reproducción, se accederá a la misma y podremos ver cuáles son las canciones que la forman.

- Al pulsar el botón “Modificar lista de reproducción”, se podrá añadir o quitar canciones a la lista de reproducción seleccionada. También se podrá modificar el orden de las canciones en la lista de reproducción.

- Al pulsar el botón “Eliminar lista de reproducción”, se eliminará la lista de reproducción seleccionada del sistema.

RF05 - CRD lista de reproducción automática.

[3] CRUD: Del inglés, *Create Read Update Delete,* lo cual significa “Crear, Leer, Actualizar y Borrar”.

- Al pulsar el botón “Crear lista de reproducción automática”, el sistema pedirá al usuario un nombre y que seleccione el contexto en el que se encuentra (running, relajación, etc.). Una vez introducido, se creará una lista de reproducción automática, basada en el contexto introducido, con el nombre que se ha escrito.

- Al pulsar en una lista de reproducción, se accederá a la misma y podremos ver cuáles son las canciones que la forman.

- Al pulsar el botón “Eliminar lista de reproducción”, se eliminará la lista de reproducción seleccionada del sistema.

RF06 - Cambiar contraseña

- Dentro de la aplicación, pulsando el botón “Cambiar contraseña”, se permitirá al usuario introducir una nueva contraseña que reemplazará a la actual contraseña.

RF07 - Recordar contraseña

- Al pulsar el botón “Recordar contraseña”, un usuario registrado al introducir el correo que usó para registrarse se le envía un correo con una nueva contraseña generada de forma aleatoria.

RF08 - Consultar canciones

- Al pulsar el botón “Canciones” podremos ver las canciones que han sido añadidas en nuestras listas de reproducción. También habrá un campo donde podremos realizar una búsqueda al introducir el nombre de una canción.

RF09 - Gestionar perfil

- Al pulsar en el botón “Mi perfil” tendremos acceso a diferentes aspectos de nuestra cuenta.

RF10 - CRUD Usuarios

- Será posible por parte del administrador el crear, ver, modificar y eliminar usuarios en el sistema.

RF11 - Valorar canciones

- Un usuario podrá valorar cada una de las canciones disponibles en el sistema en un rango de 1 a 5, siendo 1 la peor puntación y 5 la mejor.

RF12 - CRUD canciones

- El administrador podrá añadir una nueva canción al sistema

- El administrador podrá ver y revisar todas las canciones que hay en el sistema

- El administrador podrá modificar las características de una canción, como por ejemplo el nombre o el género musical, entre otras.

- El administrador podrá eliminar una canción del sistema.

RF13 - CRUD álbumes

- El administrador podrá añadir un nuevo álbum al sistema

- El administrador podrá ver y revisar todos los álbumes que hay en el sistema

- El administrador podrá modificar las características de un álbum, como por ejemplo el nombre o el género musical, entre otras.

- El administrador podrá eliminar un álbum del sistema.

RF14 - CRUD artista

- El administrador podrá añadir un nuevo artista al sistema

- El administrador podrá ver y revisar todos los artistas que hay en el sistema

- El administrador podrá modificar las características de un artista, como por ejemplo el nombre o el género musical, entre otras.

- El administrador podrá eliminar un artista del sistema.

RF15 - Configurar motor de recomendación

- El perfil de experto musical y el de administrador del sistema podrán ajustar los parámetros del sistema de recomendación, siendo capaz de alterar las recomendaciones de este, entre otras capacidades.

4.2 Requisitos no funcionales

RNF01 - Usabilidad.

- Una interfaz sencilla que permita a un usuario nuevo tras 15 minutos de uso a navegar en la misma sin problemas.

RNF02 - Facilidad de mantenimiento.

- La aplicación, si ha pasado más de 48 horas desde la última comprobación, comprobará en segundo plano cuando se ejecute si hay alguna actualización disponible y procederá a instalar la misma en caso afirmativo.

RNF03 - Confiabilidad.

- Los fallos graves no deben ocurrir más de cinco veces al mes en el sistema.

RNF04 - Escabilidad.

- Que permita soportar 1000 usuarios y al menos 100 de ellos conectados de forma simultánea.

RFN05 – Tiempo de respuesta

- El tiempo de respuesta al ejecutar cualquier acción no debe superar los 5 segundos

RFN06 – Seguridad

- Ninguna persona debe ser capaz de acceder a la cuenta de un usuario determinado, ni a la información de la misma, a no ser que disponga del usuario y la contraseña de dicho usuario.

RFN07 – Instalación

- La instalación de la aplicación no debe tardar más de 10 minutos desde que comienza el proceso, independientemente del dispositivo utilizado.

RFN08 – Tratamiento de los datos

- Los datos personales serán tratados de acuerdo a Ley Orgánica de Protección de Datos de Carácter Personal (LOPD).

4.3 Requisitos de información

RI01 - Información sobre un usuario

- Dispondremos de un correo electrónico, un ID de usuario, una contraseña y la fecha en la que se dio de alta en el servicio.

  En cuanto datos personales, dispondremos de nombre, apellidos y fecha de nacimiento.

RI02 - Información sobre una canción

- Dispondremos del nombre de la canción, la duración de la misma, el género, el artista, el álbum al que pertenece y el tempo[4].

RI03 - Información sobre un álbum

- Dispondremos del nombre del álbum, el género y el artista.

RI04 - Información sobre un artista

- Dispondremos del nombre del artista, el género y el nombre de sus álbumes.

RI05 - Información sobre una lista de reproducción

- Dispondremos del nombre de la lista de reproducción, las canciones que pertenecen a la misma, el número de canciones y la duración total.

RI06 - Información sobre las canciones escuchadas

- Tendremos un registro por cada usuario donde dispondremos del nombre de cada una de las canciones escuchadas, la hora exacta en la que fue reproducida, la fecha y la lista de reproducción a la que pertenece dicha canción.

[4] El tempo se mide en BPM (*Beats Per Minute*). Esto nos permitirá saber en qué contexto puede ser reproducida una canción. Por ejemplo, una canción con un elevado número de BPM es adecuada para hacer deporte.

Capítulo 5 – Casos de uso

Para realizar los diferentes diagramas, hemos usado UML (Unified Modeling Language) [22], el cual es un lenguaje para el modelado de sistemas software. Dicho lenguaje es un estándar que nos permite de forma gráfica definir un sistema, detallar sus aspectos, y documentar y construir el mismo. Podemos diferenciar entre tres tipos de diagramas UML: de estructura, de comportamiento y de interacción. Estos tipos de diagramas muestran diferentes aspectos del sistema representado.

Para ello, nos hemos valido de MagicDraw [21], la cual es una herramienta gráfica para el diseño y análisis de UML, que se utiliza para el modelado, documentación, construcción y mantenimiento de sistemas software orientados a objetos. Gracias a esta herramienta es posible realizar diagramas de casos de uso, diagramas de interacción, de clases, de estado, de despliegue, etc., de forma rápida y sencilla.

A continuación, se presentan los casos de uso para todos los actores de la aplicación.

5.1 Casos de uso - Usuario

Para el actor ‘Usuario’, los casos de uso son los siguientes:

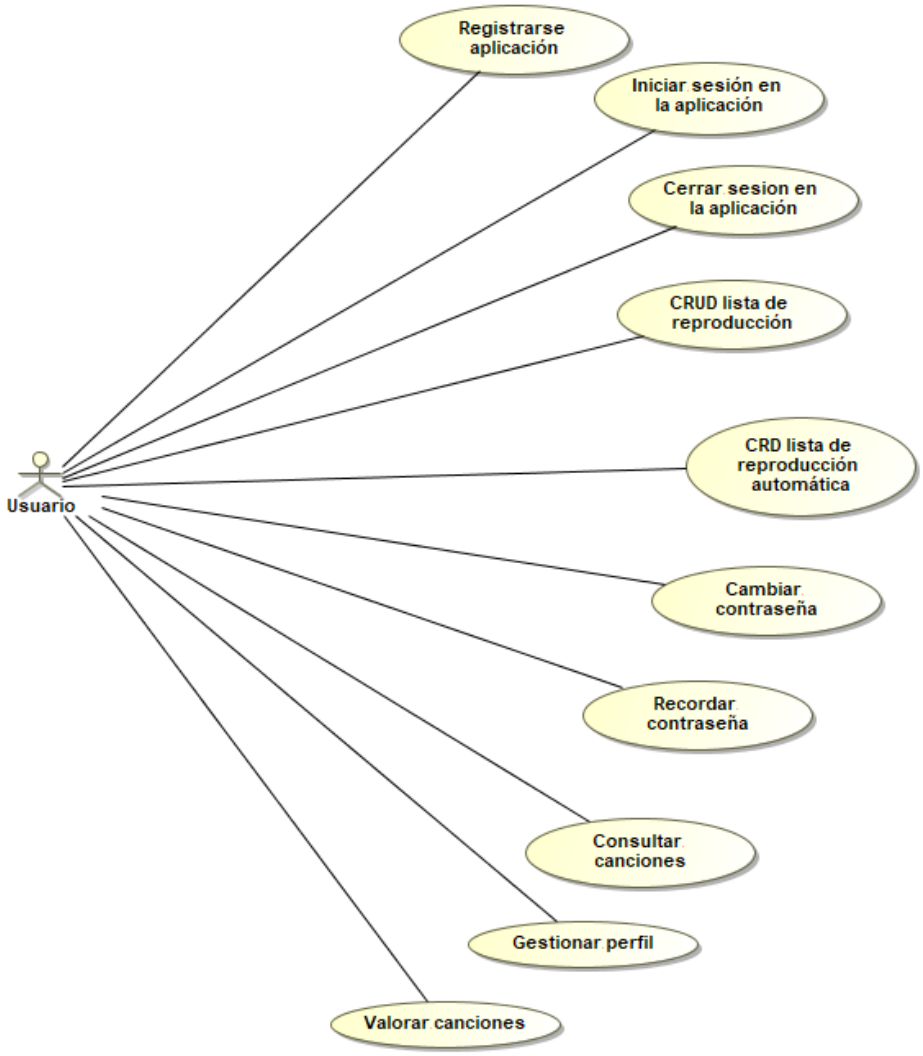

Figura : Casos de uso para el actor “Usuario”

5.2 Casos de uso – Administrador y Experto musical

Para los actores ‘Administrador’ y ‘Experto musical’ son los siguientes:

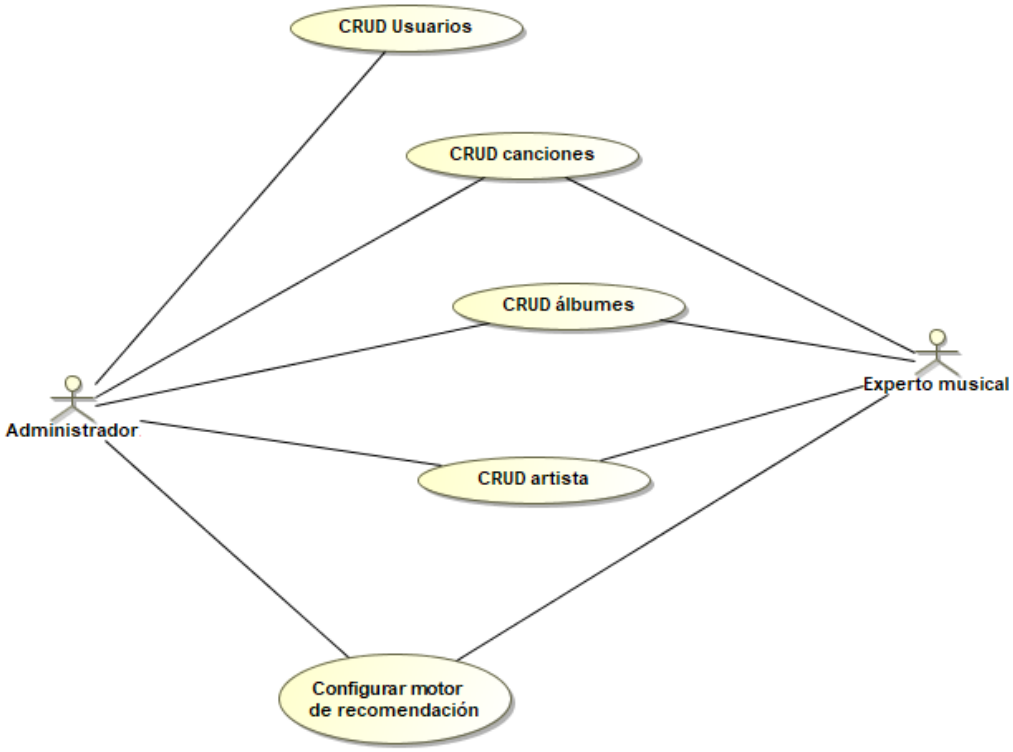

Figura 5 : Casos de uso para el actor “Administrador” y “Experto Musical”

5.3 Jerarquía de actores

La jerarquía de actores es la siguiente:

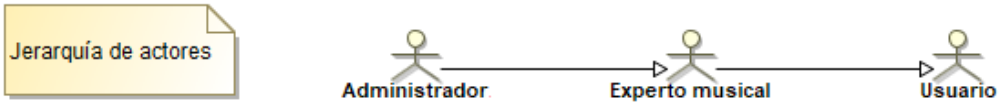

Figura 6 : Jerarquía de actores

5.4 Matriz de trazabilidad

El registro y mantenimiento de la trazabilidad de los requisitos es fundamental para poder realizar los análisis de impacto de las peticiones de cambios en los requisitos que se producen durante el transcurso de un proyecto y que deben ser gestionadas en el proceso de Gestión de Requisitos. No registrar las trazas en el momento de creación de nuevos productos de desarrollo suele provocar que posteriormente no se registren las dependencias o que se registren de manera incompleta debido a posibles olvidos por parte de los participantes responsables.

En nuestro proyecto, la matriz de trazabilidad es la siguiente:

	CS01 - Registrarse aplicación	CS02 - Iniciar sesión en la aplicación	CS03 - Cerrar sesión en la ap	CS04 - CRUD lista de reproducción	CS05 - CRD lista de reproducción a	CS06 - Cambiar contraseña	CS07- Recordar contraseña	CS08 - Consultar canciones	CS09 - Gestionar perfil	CS10 - Valorar canciones	CS11 - CRUD Usuarios	CS12 - CRUD Canciones	CS13 - CRUD Álbume	CS14 - CRUD Artista	CS15 - Configurar motor de recomendación
RF01 - Registrarse en la aplicación	X														
RF02 - Iniciar sesión en la aplicación		X				X	X								
RF03 - Cerrar sesión en la aplicación			X												
RF04 - CRUD lista de reproducción				X											
RF05 - CRD lista de reproducción automática					X										
RF06 - Cambiar contraseña						X									
RF07 - Recordar contraseña							X								
RF08 - Consultar canciones								X					X		
RF09 - Gestionar perfil									X						
RF10 - CRUD Usuarios												X			
RF11 - Valorar canciones										X					
RF12 - CRUD canciones													X		
RF13 - CRUD álbumes														X	
RF14 - CRUD artista															X
RF15 - Configurar motor de recomendación															X

Figura 7 : Matriz de trazabilidad

Capítulo 6 – Diagramas de estructura y modelo físico de datos

El objetivo de estos diagramas es identificar clases, relaciones, atributos y operaciones. Las clases agrupan o abstraen objetos con atributos, relaciones y operaciones comunes.

6.1 Diagrama de clases

En este diagrama se describe la estructura del sistema, mostrando las clases, sus atributos, operaciones, y las relaciones entre los objetos

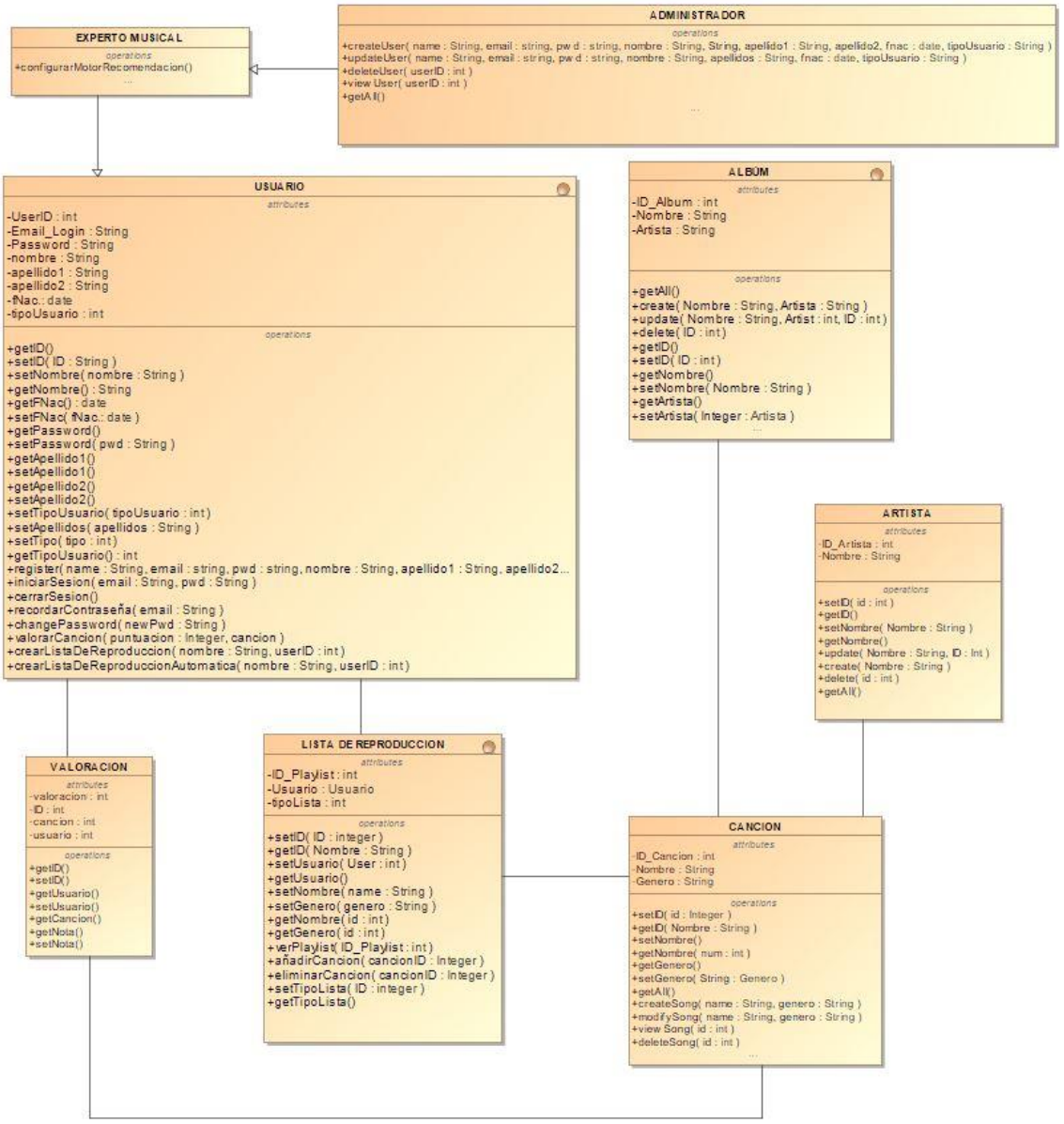

Figura 8 : Diagrama de clases.

6.2 Diagrama de clases de control

Las clases de control son los controles donde se hace la conexión entre los datos y la interfaz, donde el usuario introduce los datos y realiza las acciones que el sistema permite.

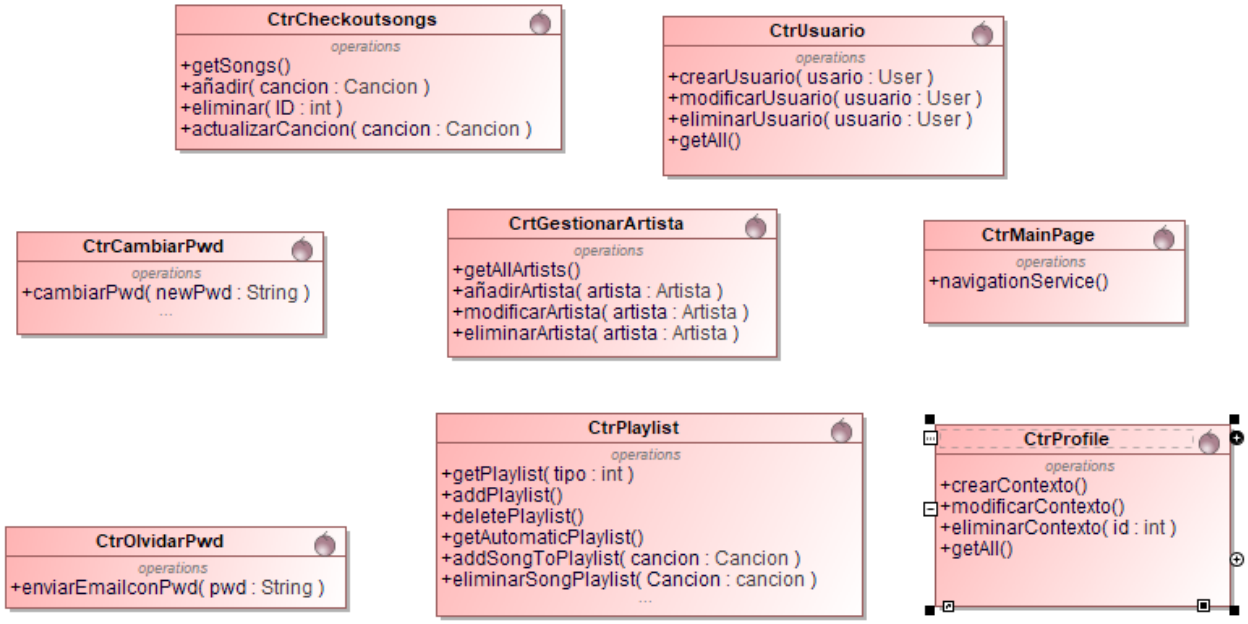

Figura 9 : Diagrama de clases de control

6.3 Diagrama de clases de interfaz

Las clases de interfaz son las que controlan los diferentes ficheros HTML [23] y la página web.

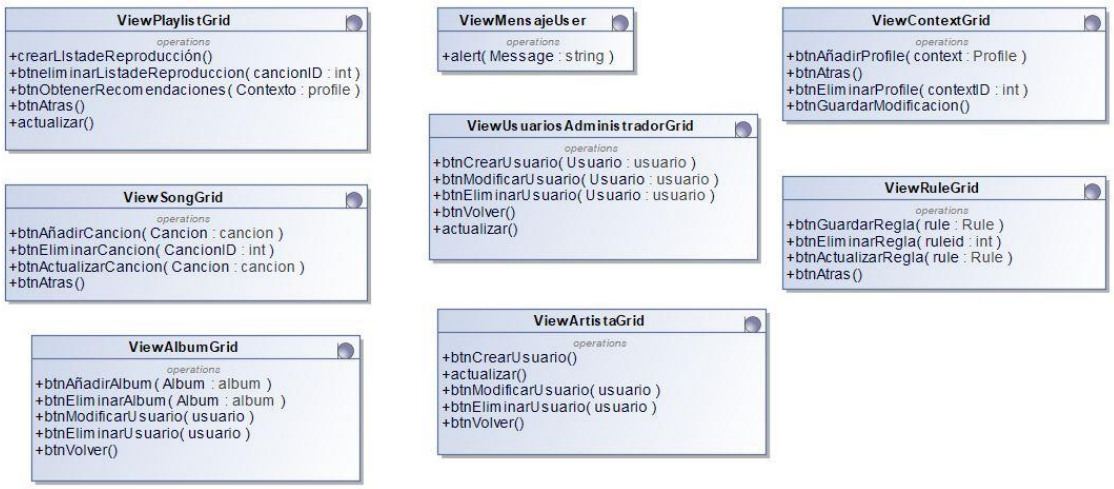

Figura 10 : Diagrama de clases de interfaz.

6.4 Diagramas de secuencias

Los diagramas de secuencia muestran de forma explícita la secuencia de los mensajes intercambiados por los objetos, describiendo así el comportamiento dinámico del sistema. Se han realizado dos diagramas de secuencia para ilustrar algunos de los procesos más importantes del sistema.

El siguiente diagrama de secuencia describe como se obtienen recomendaciones en el sistema:

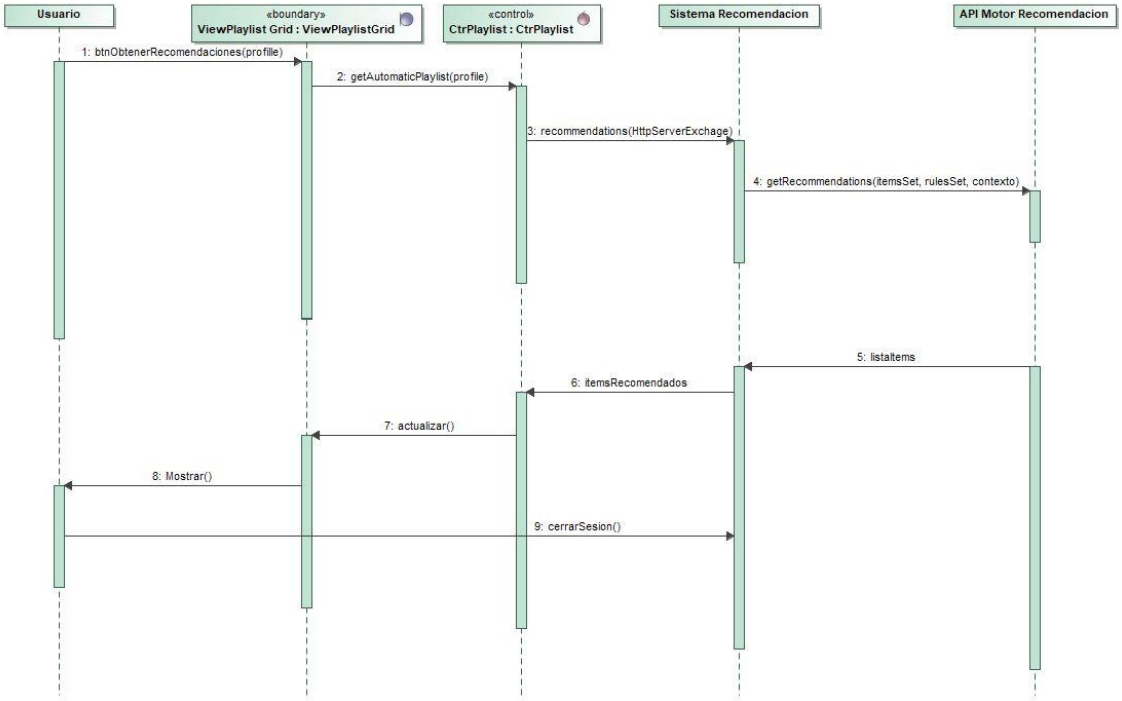

Figura 11 : Diagrama de secuencia para obtener recomendaciones

El siguiente diagrama de secuencia muestra el proceso para añadir una canción al sistema de recomendación. Cada canción que añadimos en el sistema también debe ser introducida en la API para poder ser recomendada.

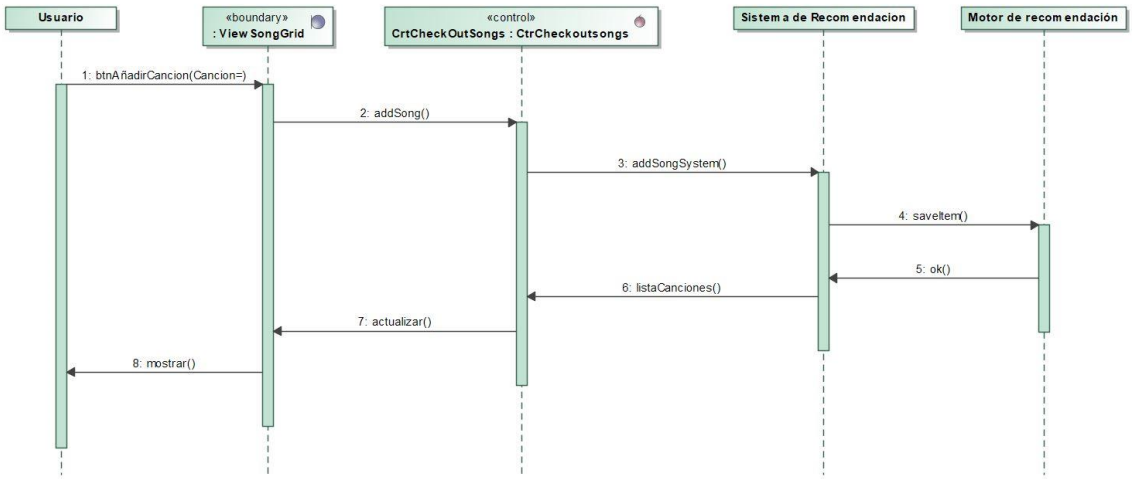

Figura 12 : Añadir canción en el sistema

6.5 Modelo físico de datos

Un modelo físico de datos es un modelo específico de bases de datos que representa objetos de datos relacionales (por ejemplo, tablas, columnas, claves principales y claves externas) y sus relaciones.

En la Figura 13 se puede observar el modelo físico de datos de nuestro sistema de recomendación.

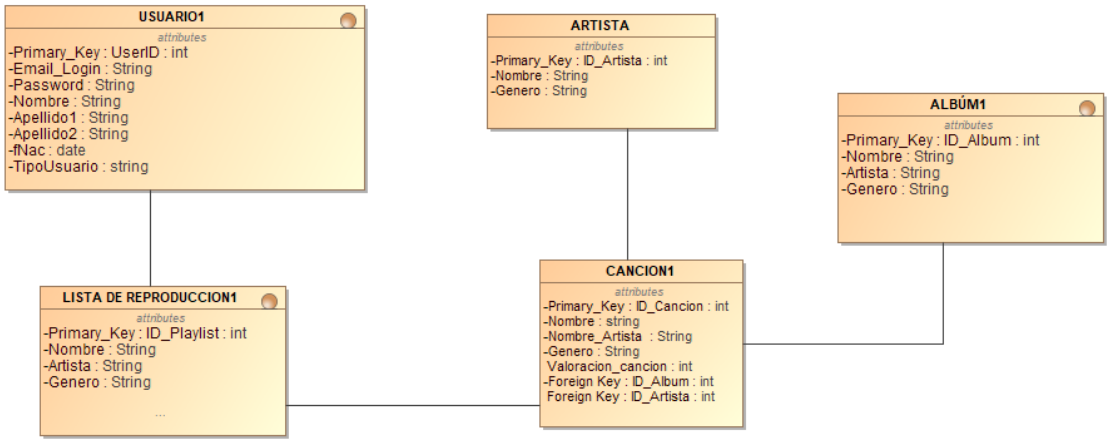

Figura 13 : Modelo Físico de Datos

Capítulo 7 – Maquetas

Para la realización de este capítulo, se ha utilizado el programa Balsamiq [24], que usa en diferentes asignaturas de la carrera, y que destaca por lo intuitivo y sencillo de su uso. Este programa permite hacer maquetas de cómo será la aplicación, permitiendo tener un referente en las fases posteriores a la hora de crear la aplicación, y, específicamente, a la hora de crear el Front-End.

A continuación, en la figura 15, 16 y 17 se muestran las maquetas que cubre las funcionalidades de registrarse en el sistema, iniciar sesión y la función de recuperar contraseña respectivamente. Aunque sea simple, se puede ver la idea básica tras estas tres pantallas. Sumado a esto, también resulta inmediatamente intuitivo lo que estamos visualizando, con títulos grandes que no dan lugar a equivocación.

´

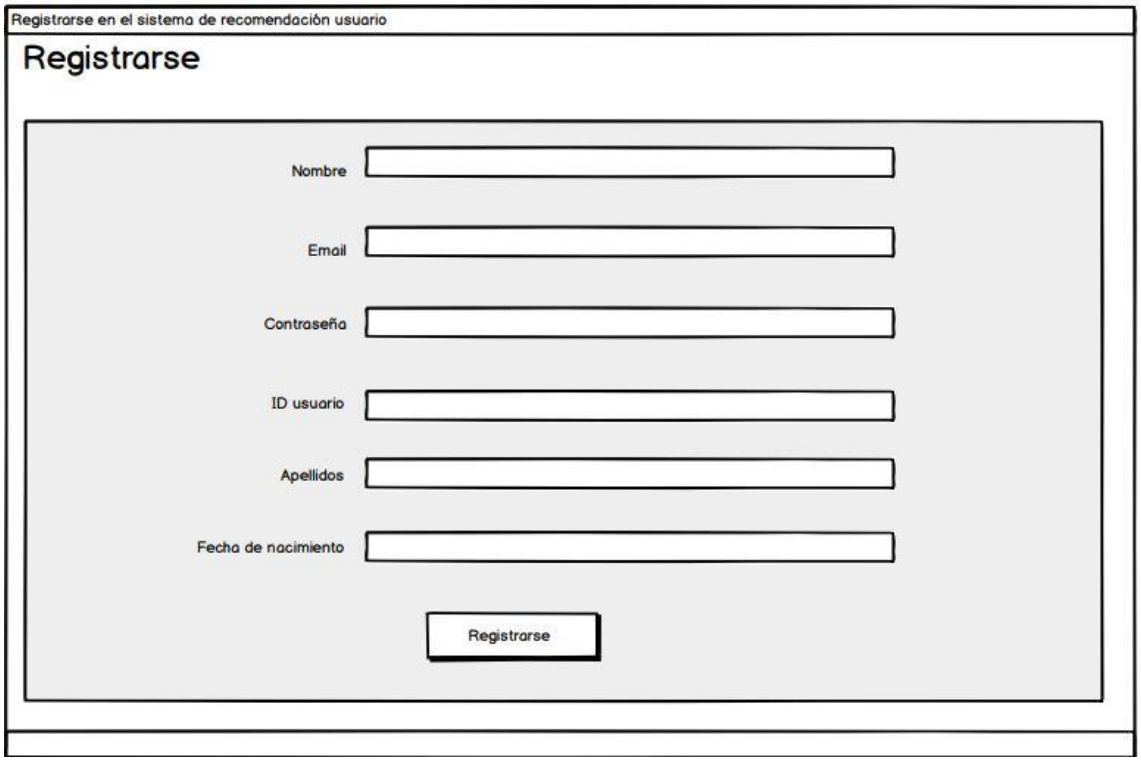

Figura 14 : Registrarse en el sistema

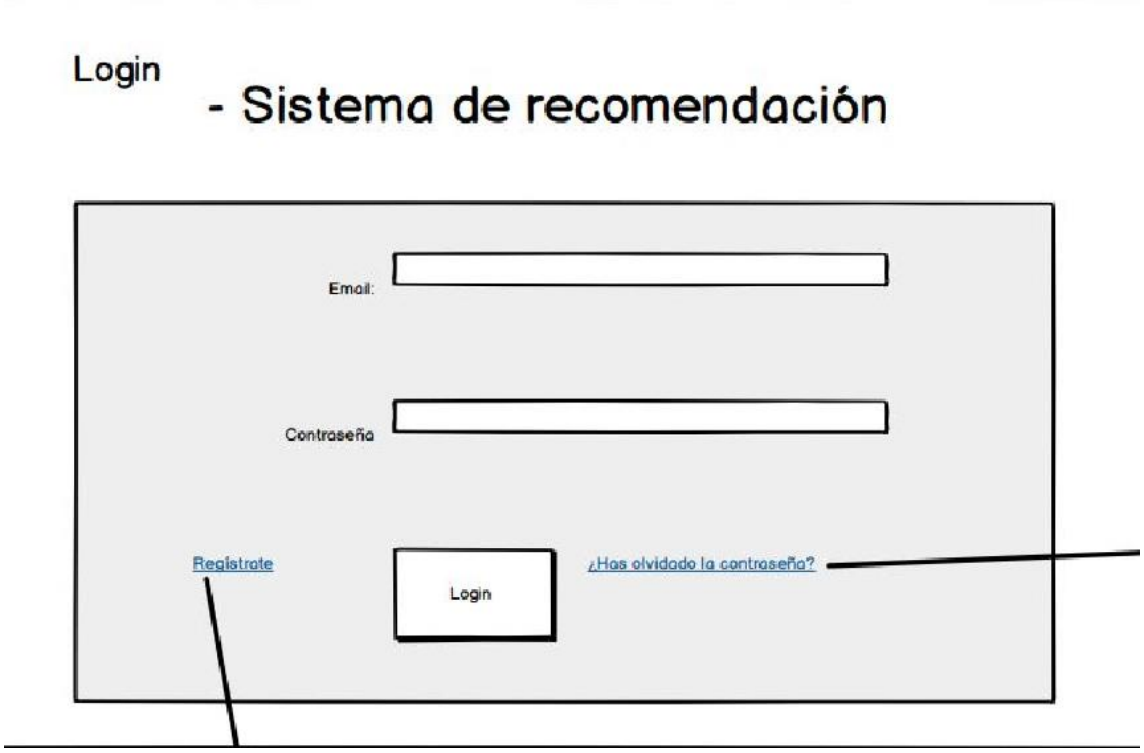

Figura 15 : Maqueta del Login

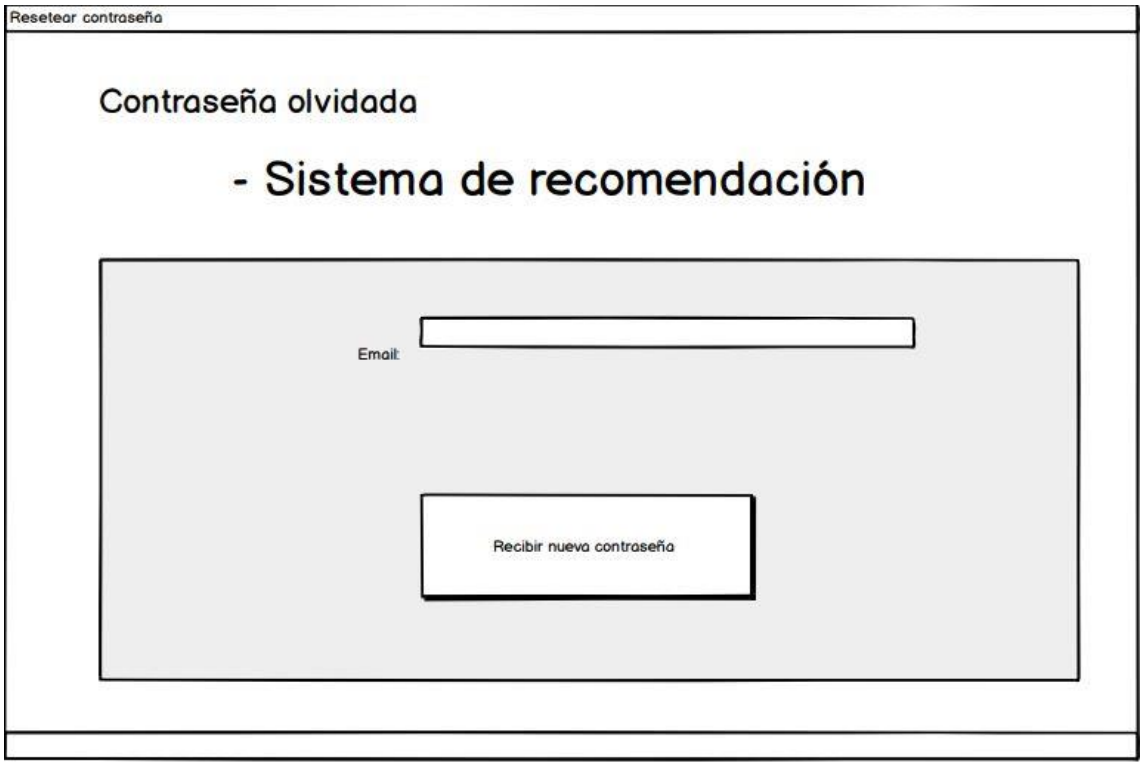

Figura 16 : Recuperar Contraseña

Para un usuario, tras iniciar sesión, entrará en la página principal, la cual consta de diferentes botones para cada una de las acciones disponibles para realizar.

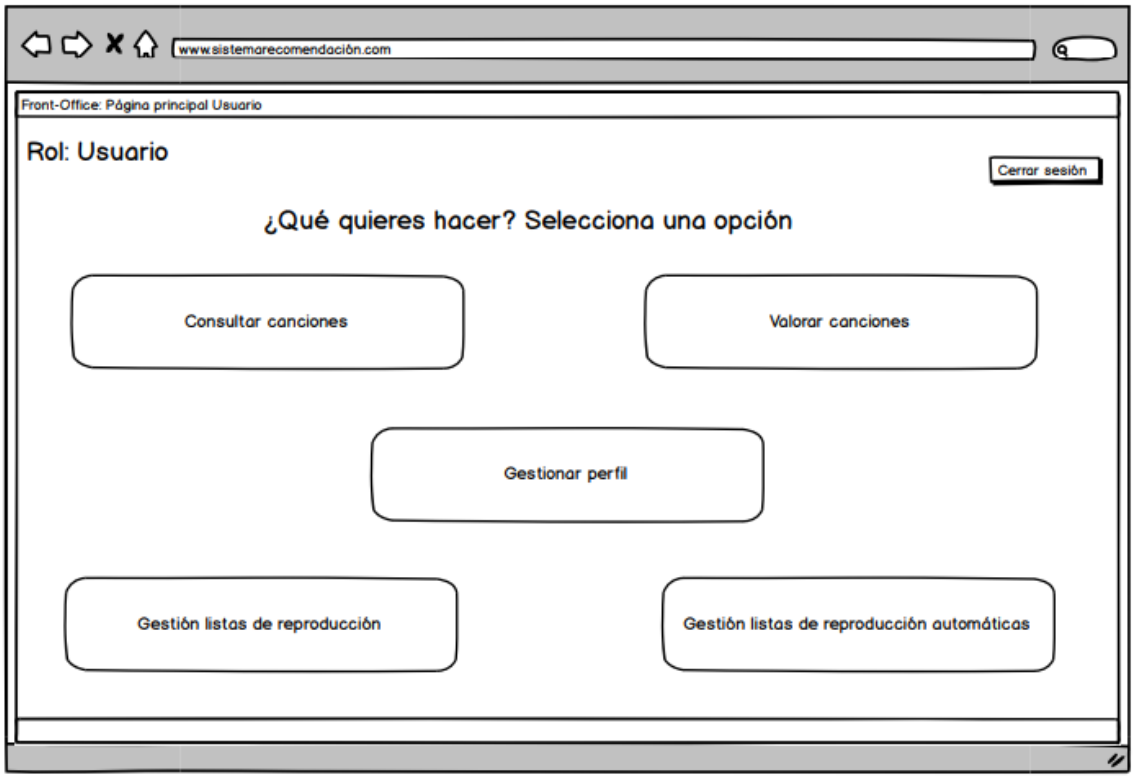

Figura 17 : Pantalla de inicio para un usuario

Para un experto musical, tras iniciar sesión, entrara en su página principal, la cual consta de diferentes botones para cada una de las acciones disponibles para realizar, como podemos observar en la Figura 18.

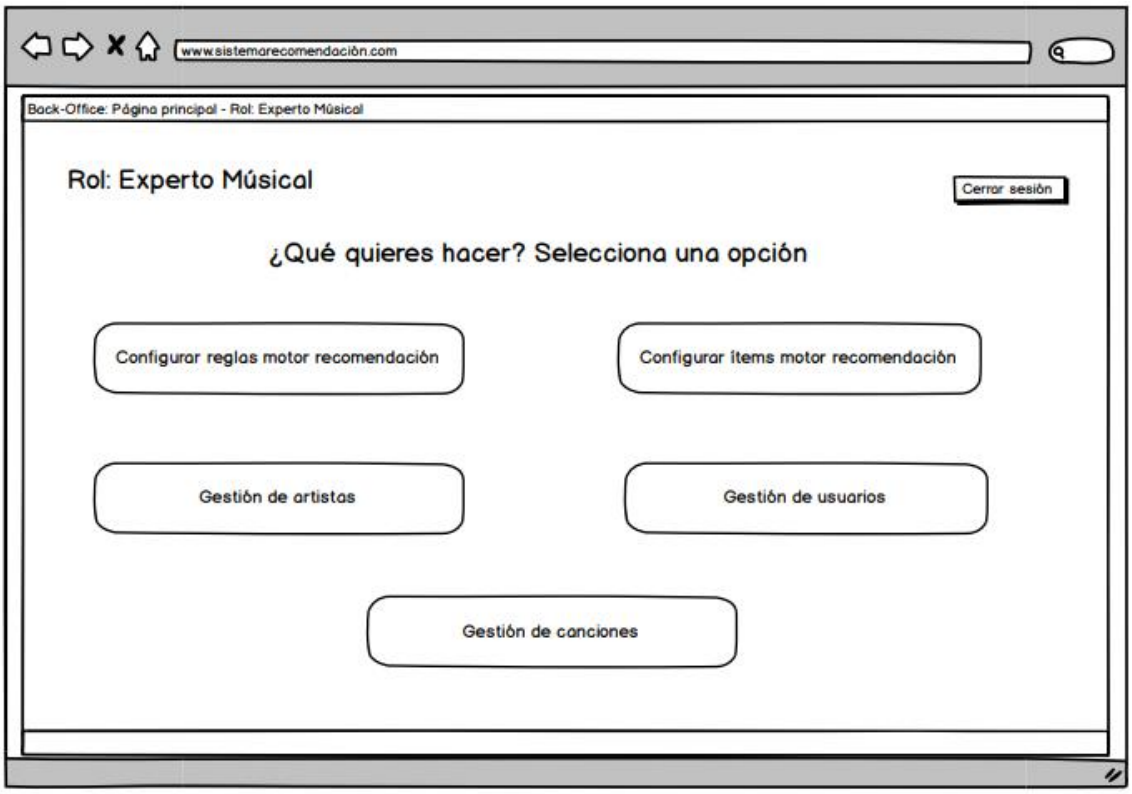

Figura 18 : Pantalla de inicio para un usuario

Para ilustrar una de las funcionalidades de un usuario, la de ver la lista de canciones, se realizó la siguiente maqueta:

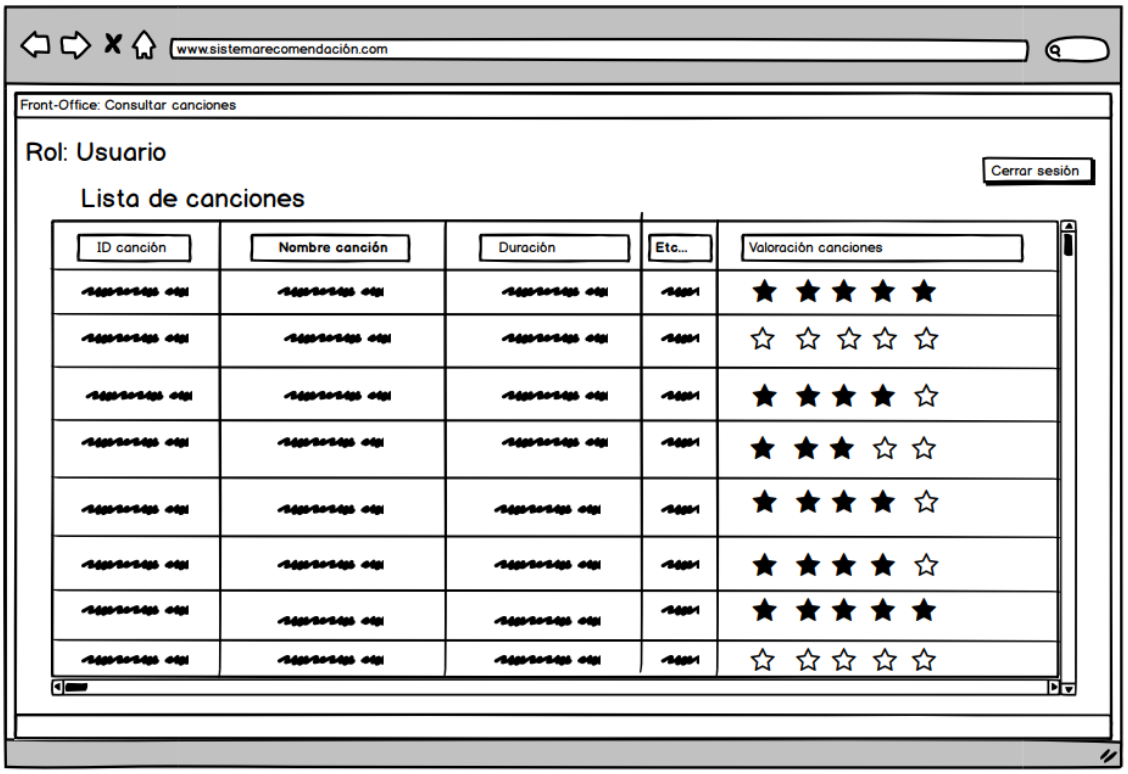

Figura 19 : Ver lista de canciones disponibles en el sistema

Capítulo 8 – Arquitectura de la aplicación

Se ha realizado un modelo de la arquitectura de la aplicación. Posteriormente, se explicará cada una de las tecnologías usadas para la realización de este proyecto. A grandes rasgos, los usuarios, a través de diversos dispositivos interactúan con el Front-End de nuestra aplicación, el cual se comunica con el back-end, el cual es el verdadero núcleo del proyecto. Esta se encarga de comunicarse con el motor de recomendación, utilizando para ello la API del grupo de investigación SICUMA. El back-end, además de funcionar como servidor, el cual esta empotrado en el mismo, también se encarga de comunicarse con la base de datos, la cual está contenida en Docker para garantizar la portabilidad del proyecto.

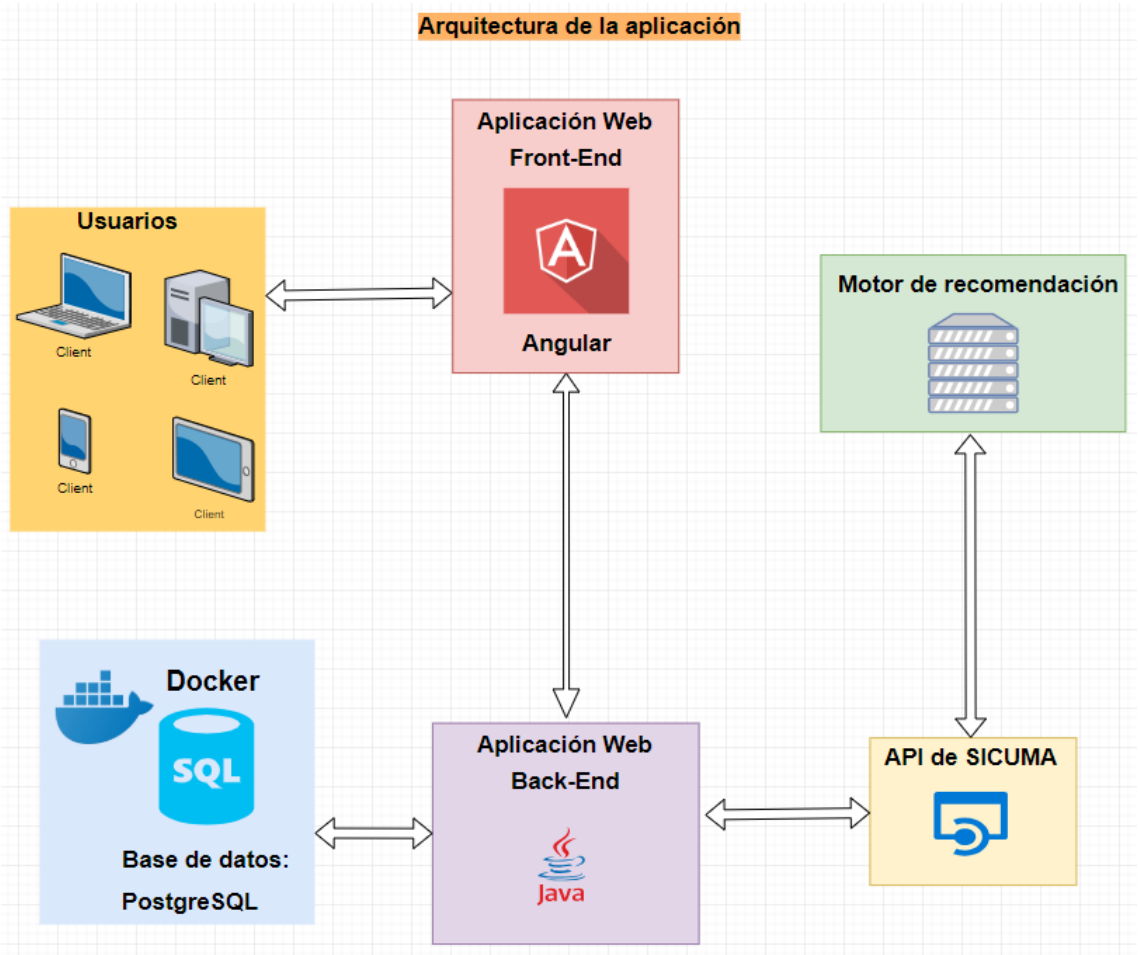

Figura 20 : Arquitectura de la aplicación

Aparte de las diferentes tecnologías y programas mencionados anteriormente, se han utilizado:

- **Java**: Es un lenguaje de programación de propósito general, concurrente y orientado a objetos. Su intención es permitir que los desarrolladores de aplicaciones escriban el programa una vez y lo ejecuten en cualquier dispositivo. [19]

- **Angular**: Es un framework de desarrollo para JavaScript. La finalidad de Angular es facilitarnos el desarrollo de aplicaciones web y además darnos herramientas para trabajar con los elementos de una web de una manera sencilla y óptima. Nos permite separar completamente el front-end y el back-end en una aplicación web. [17]

- **HTML**: Del inglés HyperText Markup Language (lenguaje de marcas de hipertexto), hace referencia al lenguaje de marcado para la elaboración de páginas web. [23]

- **JavaScript**: Es un lenguaje de programación interpretado, es decir, no se compila previamente a su ejecución, sino que se traducen las instrucciones a medida que van siendo necesarias. Se utiliza principalmente en su forma del lado del cliente (client-side), implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas. [25]

- **Bootstrap**: Framework de código abierto para diseño de sitios y aplicaciones web. Contiene plantillas de diseño con tipografía, formularios, botones y más basado en HTML y CSS, así como extensiones de JavaScript adicionales. [9]

- **PostgreSQL**: Es un sistema de gestión de bases de datos relacional orientado a objetos y libre, publicado bajo la licencia PostgreSQL. Hemos utilizado este sistema por su sencillez y potencia, cumpliendo de sobra todas las funcionalidades requeridas para el proyecto. [10]

- **Docker**: Es un proyecto de código abierto capaz de automatizar el despliegue de aplicaciones dentro de contenedores de software. Los contenedores son paquetes de elementos que permiten ejecutar una aplicación determinada en cualquier sistema operativo, siempre que tenga el procesador capacidad de virtualización. Esto nos permite flexibilidad y portabilidad en donde la aplicación se puede ejecutar, ya sea en nuestros servidores, la nube pública, nube privada, etc. Es por eso por lo que se ha elegido para empotrar nuestra base de datos, por la portabilidad que ofrece. [16]

- **Maven**: Es un marco central donde hay acceso a complementos Maven. En los complementos son donde se realiza gran parte de la acción real; los complementos se utilizan para: crear archivos jar, crear archivos war, compilar código, código de prueba unitaria, crear documentación del proyecto, etc. En nuestro caso hemos utilizado un plugin de Maven para realizar las migraciones y otro para compilar el proyecto. [12]

- **Eclipse y IntelliJ IDEA**: Plataformas de software compuestas por un conjunto de herramientas de programación multiplataforma para desarrollar. Estas plataformas se ha utilizado como entorno de desarrollo integrado (del inglés IDE) para el desarrollo de programas informáticos. [10, 20]

Capítulo 9 - Implementación del sistema de recomendación: Back-end

9.1 Introducción

A continuación, se detalla cómo se ha realizado la implementación del sistema de recomendación. La aplicación tiene dos partes bien diferenciadas; por una parte, está el back-end, que se encarga de comunicarse con el motor de recomendación mediante la API del grupo de investigación SICUMA, y que de hecho actúa como una segunda API de cara al front-end. El back-end se encarga de guardar la información necesaria en la base de datos, manteniendo la integridad de los datos entre la información que hay en la API del motor de recomendación y la que existe en nuestra base de datos.

Los detalles más relevantes del back-end son los siguientes:

9.2 Estructura del proyecto de back-end

Se ha utilizado como IDE Eclipse Java Photon, pero sobre todo IntelliJ IDEA 2018, ya que este último es mucho más fiable a la hora de compilar.

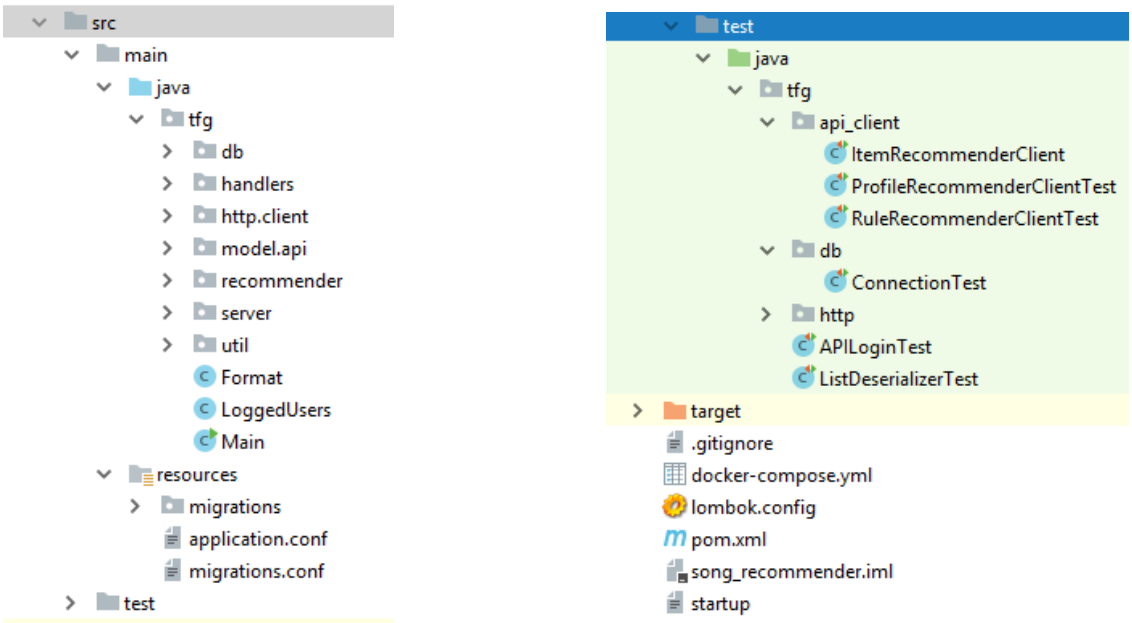

Figura 21 : Estructura del proyecto en el IDE

9.3 Back-end: Conexión con la API

En el paquete del proyecto http.client, están todas las clases necesarias para conectar con la API del grupo de investigación SICUMA. La más importante es la clase HttpClient, que realiza las peticiones http directamente a la API utilizando la librería de java OkHttp. Esta clase añade automáticamente el token del login a todas las peticiones, automatizando dicho proceso y, además, añade datos que se repiten en todas las peticiones, como por ejemplo el nombre de usuario y el sistema.

```
public String get(Endpoint endpoint, List<Tuple2<String, String>> parameters) {
    return call(addAuth(RequestBuilder.get(endpoint, parameters)));
  }

  public String post(Endpoint endpoint, List<Tuple2<String, String>> parameters)
{
    return call(addAuth(RequestBuilder.post(endpoint, parameters)));
  }

private Request addAuth(Request request) {
    return request.newBuilder().addHeader("Authorization", "Bearer " +
APILogin.instance().token()).build();
  }
```

Figura 22 : Métodos de ejemplo de la clase HttpClient

También está la clase RequestBuilder, la cual se encarga de recibir listas de tuplas de vavr y convertirlas en requests http de HttpClient. Las tuplas de vavr son simplemente pares con los datos nombre del parámetro y valor del parámetro. Todos estos parámetros se añaden al request que luego se realizará a la API.

Por otra parte, la clase Endpoint simplemente contiene todos los endpoints de la API del grupo de investigación SICUMA, a los cuales vamos a realizar las peticiones utilizando objetos de la clase URL. Esta separa una URL en varios elementos diferentes, como por ejemplo esquema, dirección y ruta completa.

```
COPY_GENERIC_RULES_SET(URL.of("https", "galactus.uma.es",
"sicumarecommender/recommender/copyGenericRulesSet")),
  COPY_RULES_SET(URL.of("https", "galactus.uma.es",
"sicumarecommender/recommender/copyRulesSet")),
  DELETE_ITEM(URL.of("https", "galactus.uma.es",
"sicumarecommender/recommender/deleteItem")),
  DELETE_PROFILE(URL.of("https", "galactus.uma.es",
"sicumarecommender/recommender/deleteProfile")),
  DELETE_RULE(URL.of("https", "galactus.uma.es",
"sicumarecommender/recommender/deleteRule")),
  GET_DISTINCT_ATTRIBUTES(URL.of("https", "galactus.uma.es",
"sicumarecommender/recommender/getDistinctAttributes")),
  GET_DISTINCT_PROFILE_ATTRIBUTES(URL.of("https", "galactus.uma.es",
"sicumarecommender/recommender/getDistinctProfileAttributes")),
  GET_DISTINCT_RULES(URL.of("https", "galactus.uma.es",
"sicumarecommender/recommender/getDistinctRules")),
  GET_ITEM(URL.of("https", "galactus.uma.es",
"sicumarecommender/recommender/getItem")),
  GET_ITEMS(URL.of("https", "galactus.uma.es",
"sicumarecommender/recommender/getItems")),
  GET_PROFILE(URL.of("https", "galactus.uma.es",
"sicumarecommender/recommender/getProfile")),
  GET_PROFILES(URL.of("https", "galactus.uma.es",
"sicumarecommender/recommender/getProfiles")),
  GET_RECOMMENDATIONS(URL.of("https", "galactus.uma.es",
"sicumarecommender/recommender/getRecommendations")),
```

Figura 23 : Ejemplo de la clase Endpoint en el paquete HttpClient

Es de especial relevancia el paquete recommender, donde tenemos los datos de conexión con la API en la clase AuthData, y una clase API Login que utiliza la memoimacion de guava para refrescar el token que la API nos da para permitirnos el acceso. La memoimacion funciona de la siguiente manera: el token se genera de forma lazy, es decir, únicamente cuando es necesario. Una vez generado, se conserva durante 24 horas y, después de las 24 horas, si se vuelve a intentar utilizar, se refresca el token.

```
public class APILogin {
  private static final APILogin INSTANCE = new APILogin();
  private final Supplier<String> tokenSupplier =
Suppliers.memoizeWithExpiration(this::refreshToken, 24,
TimeUnit.HOURS);

  private APILogin() { }

  public static APILogin instance() {
    return INSTANCE;
  }

  public String token() {
    return tokenSupplier.get();
  }

  private String refreshToken() {
    return HttpClient.instance().login();
  }
}
```

Figura 24 : Clase APILogin, incluida en el paquete recommender

La clase RecommenderClient es la fachada que une todas las clases anteriores en una sola clase, imitando todos los métodos de la API del grupo de investigación SICUMA, pero convirtiéndolos a métodos de Java. Todo lo que necesitamos para conectarnos a la API se realiza de forma transparente al usuario cuando utiliza estos métodos.

```
public static Profile saveProfile(@NotNull String profileAttribute,
                                  @NotNull String ruleDesc,
                                  @NotNull String userDesc,
                                  @NotNull Double value) {
  String response = HttpClient.instance().post(Endpoint.SAVE_PROFILE,
addSystem(List.of(
      Tuple.of("profileattribute", profileAttribute),
      Tuple.of("ruledesc", ruleDesc),
      Tuple.of("userdesc", userDesc),
      Tuple.of("value", value.toString())
  )));
  return Deserializer.apply(response, Profile.class);
}

public static Rule saveRule(@NotNull String attribute,
                            @NotNull RulePart part,
                            @NotNull String ruleDesc,
                            String rulesSet,
                            @NotNull Double value) {

  String response = HttpClient.instance().post(Endpoint.SAVE_RULE,
addSystem(List.of(
      Tuple.of("attribute", attribute),
      Tuple.of("part", part.getString()),
      Tuple.of("ruledesc", ruleDesc),
      Tuple.of("rulesset", rulesSet),
      Tuple.of("value", value.toString())
  )));
```

Figura 25 : Algunos métodos de la clase RecommenderClient

9.3 Back-end: Clases valor

Para poder conectarnos con la API de SICUMA necesitamos convertir los objetos JSON [5] que nos devulve está a objetos POJO [6] java.

Esto se ha conseguido utilizando la librería Lombok y se puede ver su uso en las clases de model.api. Ahí se observan varias clases, como por ejemplo Action, Item o Recommendation, que se corresponden con objetos JSON que devuelve la API.

Para realizar estos objetos únicamente se ha declarado las propiedades, se les ha añadido anotaciones, las cuales sirven para mapear cada atributo con una propiedad JSON y un atributo @Value (de la librería Lombok), que automáticamente convierte el objeto en un POJO añadiéndole getters, setters, equals y HashCode.

[5] JSON: Es un acrónimo de JavaScript Object Notation, «notación de objeto de JavaScript». Básicamente es un formato de texto ligero para el intercambio de datos.
[6] POJO: Es un acrónimo de Plain Old Java Object. Es una sigla utilizada por programadores Java para enfatizar el uso de clases simples y que no dependen de un framework en especial.

```
@Value
public class Profile {
  @JsonProperty("_id")
  private final String id;

  @JsonProperty("system")
  private final String system;

  @JsonProperty("ruledesc")
  private final String ruleDescriptor;

  @JsonProperty("profileattribute")
  private final String profileAttribute;
```

Figura 26 : Algunos métodos de la clase RecommenderClient

9.4 Back-end: Login de usuarios

Respecto a las sesiones de los diferentes usuarios del sistema, todo está contenido en la clase LoggedUsers del paquete util. El contenido de esta clase es simple, pero al mismo tiempo complejo, ya que controla todo lo relativo al seguimiento de que usuarios están logueados en el sistema y cuáles son sus respectivos tokens de acceso. En esta clase se guardan dos mapas, uno con un timestamp del momento del login asociado al token y otra con un objeto usuario asociado al token. Por lo tanto, podemos saber a qué usuario corresponde un token y si está logueado. Cuando llamamos a cualquiera de los método de la clase, se llama al metodo removeOldEntries(), el cual se encarga de borrar todos los tokens con una antigüedad superior a 24 horas. Para ello, se sirve del timestamp para saber cuáles debe borrar de las listas.

```
public synchronized Usuario getUser(String token) {
    removeOldEntries();
    return users.get(token);
  }

public synchronized String log(Usuario usuario) {
    removeOldEntries();
    String token = UUID.randomUUID().toString();
    timestamps.put(token, System.currentTimeMillis());
    users.put(token, usuario);
    return token;
 }

  synchronized private void removeOldEntries() {
    for (Map.Entry<String, Long> entry : timestamps.entrySet())
      if (elapsedTime(entry.getValue(), System.currentTimeMillis()) >
TIME_TO_REMOVE) {
        timestamps.remove(entry.getKey());
        users.remove(entry.getKey());
      }
  }

  private long elapsedTime(long start, long end) {
    return end - start;
  }
}
```

Figura 27 : Algunos métodos de la clase RecommenderClient

9.5 Back-end: Undertow y el servidor

Esta es, probablemente, la parte más compleja de la aplicación. La aplicación contiene un servidor empotrado que nos permite lanzarla directamente desde el IDE o la consola sin necesidad de utilizar servidores de aplicación o ningún tipo de infraestructura extra. Este servidor empotrado se llama SimpleServer y se puede ver como se utiliza en la clase Main.

```
  public static void main(String[] args) {
    Router router = new V1Router();

    SimpleServer server = new SimpleServer(9011, "0.0.0.0", router);

    server.start();

    System.out.println("Server running");
  }
```

Figura 28 : Ejemplo de llamada a SimpleServer en la clase Main

Para poder utilizar simple server necesitamos pasarle un Router. Por ello tenemos la clase V1Router. Esta clase, V1Router, simplemente contiene una lista con todas las rutas que corresponden a los diferentes endpoints, a los cuales un usuario puede acceder desde el front-end. Estas rutas se forman de la siguiente manera; con la dirección de la ruta, el método que se usa para consultarlas, ya sea Post o Get, y, finalmente, una función, que es el código que se ejecuta cada vez que se llama a ese método.

```
Route.of(Methods.POST, API_VERSION + "/login", Users::login),
Route.of(Methods.POST, API_VERSION + "/create", Users::create),
Route.of(Methods.POST, API_VERSION + "/change/password",
Users::changePassword),
Route.of(Methods.POST, API_VERSION + "/profile/read", Users::read),
Route.of(Methods.POST, API_VERSION + "/profile/update",
Users::updateOwn),
Route.of(Methods.POST, API_VERSION + "/recover", Users::recoverPass)
```

Figura 29 : Parte de la clase V1Router

Por otra parte, el código que se ejecuta al llamar a cada método está en las clases que pertenecen al paquete handlers. Se caracterizan por ser secuencias de instrucciones muy repetitivas, con operaciones como insertar en la base de datos o consultarla. En todos los métodos se realiza siempre el mismo proceso:

Primero se reciben los parámetros de la petición. Esto se repite en todos los handlers, excepto en los que se consultan por get. Para ello se utiliza la primera línea de cada método:

```
JsonNode node = Exchanges.readBodyAsNode(exchange);
```

Figura 30 : Primera línea de cada uno de los métodos de la clase Profiles

Exchange es un objeto que Undertow siempre envía en las funciones, y el cual contiene tanto la información necesaria para el request http como la información necesaria para la respuesta.

La respuesta se devuelve en todos los casos y siempre en la última línea, con dos métodos diferentes; Exchange.OK para devolver un ok, el cual sirve para saber si la petición que hemos lanzado realmente se ha realizado correctamente. El segundo caso, es un Exchange.Serve, el cual nos permite devolver un string cualquiera al usuario. Normalmente devolvemos un JSON, excepto en algunos casos particulares, como por ejemplo el login, en el que solo devolvemos un string con el contenido del token de login.

```
public static void create(HttpServerExchange exchange) {

  JsonNode node = Exchanges.readBodyAsNode(exchange);

  CheckUser.forType(node.get("token").asText(), 2);

  String name = node.get("nombre").asText();

  JOOQ.dsl.insertInto(CANCION, CANCION.NOMBRE, CANCION.ALBUM,
CANCION.GENERO)
      .values(
          name,
          node.get("album").asInt(),
          node.get("genero").asText()
      ).execute();

  RecommenderClient.saveItem(ACOUSTICNESS, "CancionesTest4", name,
node.get(ACOUSTICNESS).doubleValue());
  RecommenderClient.saveItem(DANCEABILITY, "CancionesTest4", name,
node.get(DANCEABILITY).doubleValue());
  RecommenderClient.saveItem(ENERGY, "CancionesTest4", name,
node.get(ENERGY).doubleValue());
  RecommenderClient.saveItem(SPEECHINESS, "CancionesTest4", name,
node.get(SPEECHINESS).doubleValue());
  RecommenderClient.saveItem(LIVENESS, "CancionesTest4", name,
node.get(LIVENESS).doubleValue());

  Exchanges.ok(exchange);
}
```

Figura 31 : Metodo de la clase Songs, donde podemos observar el uso de Exchanges

9.6 Back-end: Base de datos

La conexión con la base de datos se realiza siempre en los objetos handler, utilizando el objeto JOOQ.dsl. Este objeto nos permite encadenar secuencias de métodos como insert, select, update, where, etc que se corresponden directamente con sentencias SQL, las cuales se utilizan para acceder a la base de datos. Estas secuencias de métodos terminan con instrucciones como “execute” para ejecutar el insert e introducir los datos en la base de datos o “fetch into”, que nos asocia el contenido que se ha recibido a una clase, por ejemplo, Artist o Album. Gracias a esto, es como escribir sentencias SQL para trabajar con una base de datos, pero directamente en Java.

JOOQ también es capaz de generar una salida JSON a partir del contenido de la tabla de base de datos que se ha seleccionado. Este contenido se ha utilizado a menudo en el final de algunos métodos pertenecientes al paquete handler para devolverlo directamente al usuario de la API.

```
public static void create(HttpServerExchange exchange) {
  JsonNode node = Exchanges.readBodyAsNode(exchange);

  CheckUser.forType(node.get("token").asText(), 2);

  JOOQ.dsl.insertInto(ARTISTA,
ARTISTA.NOMBRE).values(node.get("name").asText()).execute();

  Exchanges.ok(exchange);
}

public static void all(HttpServerExchange exchange) {
  String artists =
JOOQ.dsl.select().from(ARTISTA).fetch().formatJSON(Format.JSON);

  Exchanges.serve(exchange, artists);
}
```

Figura 32 : Método de la clase Rules, donde podemos observar el uso de JOOQ

El handler más complejo es el de Recommendations, y es con este con el que obtenemos las recomendaciones de canciones para un usuario basadas en un contexto que este especifique. Recommendations pide a la API una cantidad de recomendaciones de canciones, apropiadas para el contexto que especifica el cliente. A continuación, cada una de esas canciones (ítems) devueltas por la API se coteja con canciones de la base de datos y se devuelve un join con el contenido de las canciones, el artista, el álbum y la información que devuelve la API directamente al cliente.

```
public static void recommendations(HttpServerExchange exchange) {
    JsonNode node = Exchanges.readBodyAsNode(exchange);

    List<Recommendation> recommendations =
RecommenderClient.getRecommendations("CancionesTest4", "ReglasTest1",
node.get("contexto").asText());

    recommendations = recommendations.subList(0, 10);

    List<Song> songs = new ArrayList<>();

    for (Recommendation recommendation : recommendations) {
      try {
        Song song = JOOQ.dsl
            .select(CANCION.ID, CANCION.NOMBRE, ALBUM.NOMBRE.as("album"),
ARTISTA.NOMBRE.as("artista"), CANCION.GENERO)
            .from(CANCION)
            .join(ALBUM)
            .on(CANCION.ALBUM.eq(ALBUM.ID))
            .join(ARTISTA)
            .on(ALBUM.ARTISTA.eq(ARTISTA.ID))

.where(upper(CANCION.NOMBRE).eq(recommendation.getName().toUpperCase()))
            .fetchAny().into(Song.class);
        songs.add(song);
      } catch (Exception e) {
        // Ignore it, keep going
      }
    }

    Exchanges.serve(exchange, Serializer.toJson(songs));
  }
```

Figura 33 : Método recommendations, con el que obtenemos las recomendaciones

9.7 Back-end: Clase de utilidad CheckUser

La clase CheckUser se utiliza en la gran mayoría de las clases del paquete handler. Esta clase simplemente nos da una utilidad para consultar el objeto LoggedUsers. Se le daba pasar un token y un número, que representa el tipo de usuario, los cuales tienen diferentes permisos a la hora de ejecutar una acción. Para ello, recoge el usuario de la clase LoggedUsers y comprueba que el tipo del usuario sea el adecuado para poder realizar el método que se llama. Si no lo es, se lanza una excepción y se interrumpe la llamada al método.

Capítulo 10 - Implementación del sistema de recomendación: Front-end

10.1 Front-end: Introducción

El front-end es la parte de la aplicación con la que interactúa el usuario, y dicha parte de la aplicación ejecuta los diferentes métodos disponibles en el back-end según las acciones del usuario. Se ha utilizado esta aproximación por comodidad, reusabilidad y mantenibilidad.

10.2 Estructura del proyecto de Front-End

Se ha utilizado como IDE Eclipse Java Photon, con el correspondiente plugin de Angular para poder trabajar con fichero de tipo TypeScript, etc. La estructura del proyecto se puede observar en la figura 10.6.2. El Front-end se encarga de poner a disposición del usuario todas las funcionalidades y lógica desarrollada en back-end.

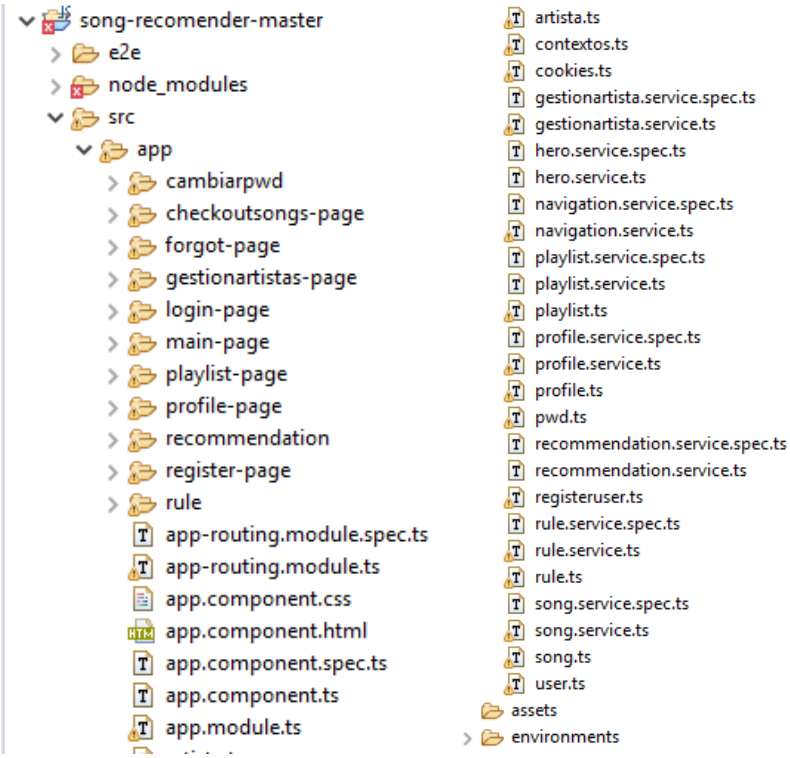

Figura 34 : Método recommendations, con el que obtenemos las recomendaciones

La estructura del proyecto se basa en las buenas prácticas recomendadas de la comunidad angular, separando las clases, los componentes y los servicios en diferentes ficheros para poder mantener una alta readibilidad, reusabilidad y mantenibilidad. Esto hace el trabajo mucho más metódico y, por tanto, menos propenso a errores.

10.3 Conexión del Front-End con el Back-End

Para ilustrar el proceso de cómo funcionan la conexión entre el Front-End y el Back-End, vamos a trabajar sobre un ejemplo concreto. En este caso, el de la página de login. El objetivo de este es muy sencillo; se le muestra al usuario una página web con dos campos a rellenar por él. Uno es su correo electrónico y otro es su contraseña, lo que le permitirá el acceso al sistema en caso de que ese usuario este registrado en el sistema.

A continuación, podemos observar el código de la clase componente login.

En primer lugar, cuando el usuario introduce los datos requeridos, y pulsa el botón iniciar sesión, se llama a la función onSubmit(), la cual se ejecuta cuando el usuario pulsa dicho botón. En la variable data se almacenan el email y contraseña introducidos en formato JSON, para comunicarnos con nuestro back-end.

```
  public onSubmit()_: void {


var obj = { email: this.model.email, password : this.model.password};

var data = JSON.stringify(obj);
console.log(data);
var xhr = new XMLHttpRequest();
xhr.withCredentials = true;
xhr.addEventListener("readystatechange", function () {
  if (this.readyState === 4) {
    console.log(this.responseText);
  }
});

xhr.open("POST", this.puerto1.url + "/api/v1/login",false);
xhr.setRequestHeader("Cache-Control", "no-cache");
xhr.setRequestHeader("Postman-Token", "92b37cde-0cba-4761-9ad7-
cd42c668c4ba");

xhr.send(data);

    console.log(xhr.response);
```

Figura 35 : Login desde el front end, primera mitad del método onSubmit()

Lo siguiente que hacemos es imprimirlo por consola. El motivo de esto es que, cuando abramos el modo desarrollador del navegador, podamos usar dichos mensajes en caso de que sea necesario debugear la aplicación por algún error.

Creamos una petición Http y la enviamos a la dirección de la API nuestra del back-end que contiene el método que buscamos. En este caso la segunda parte de la dirección es /api/v1/login. La primera parte es http://localhost:9001, pero dicha dirección la almacenamos en la clase url, ya que el puerto puede cambiar, y al escribirlo de esta manera, con solo cambiar la variable, cambiamos el puerto en todas las clases, en vez de tener que hacerlo de una en una.

```
console.log(xhr.response);

    document.cookie = xhr.response;
    var x = document.cookie;
    console.log("la cookie login -- es:  " + x);

    if (xhr.status == 200) {
    this.navigationService.navigateToPage('main');
     } else {
      console.log("Usuario y/o password incorrectos");
            }
}
```

Figura 36 : Login desde el front end, segunda mitad del método onSubmit()

En xhr.response obtenemos el token del back-end, el cual necesitaremos para la gran mayoría de operaciones que un usuario puede realizar en back-end, como se explicó en el capítulo anterior. Por tanto, es un atributo que debemos recordar cuando nos movamos de página, así que nos valemos de las características de javascript y guardamos nuestro token en document.cookie. Dicho valor persiste durante toda la sesión de un usuario, por lo que podremos acceder al token más adelante cuando sea necesario.

Siguiendo con nuestro ejemplo, si el email y la contraseña introducida son correctas, se redirige al usuario a la página principal (‘main’). En caso contrario, se le muestra un mensaje para que vuelva a introducir el email y la contraseña.

A continuación, se muestra el proceso en la aplicación front-end.

En la Figura 37, se ha introducido una contraseña incorrecta, y, por tanto, el usuario permanece en la página de login, y se registra una serie de mensajes en la consola. Por una parte, un mensaje de aviso indicando que el email y/o la contraseña son incorrectas, y que el token este vacio, lo cual es lógico, ya que al no haberse logeado correctamente nuestro back-end no devuelve nada.

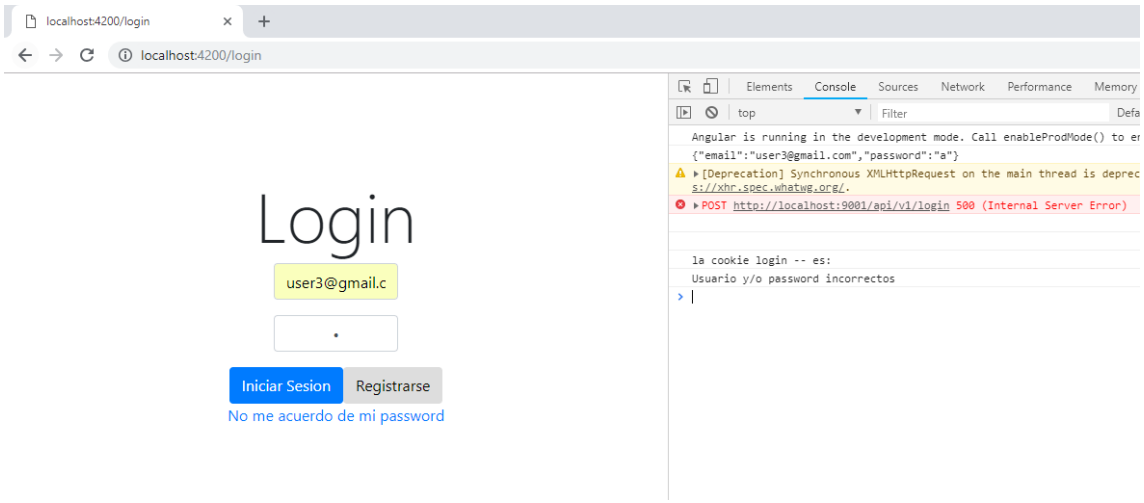

Figura 37 : Email y/o contraseña incorrecta

En la Figura 38, se ha introducido un email y contraseña valido, y por tanto la aplicación te redirige a la página principal. Además, podemos observar que se muestra el token tanto en la página de login como en la página principal, lo que indica que dicha variable persiste aun cambiado de página, y por tanto, puede ser utilizada para almacenar el token.

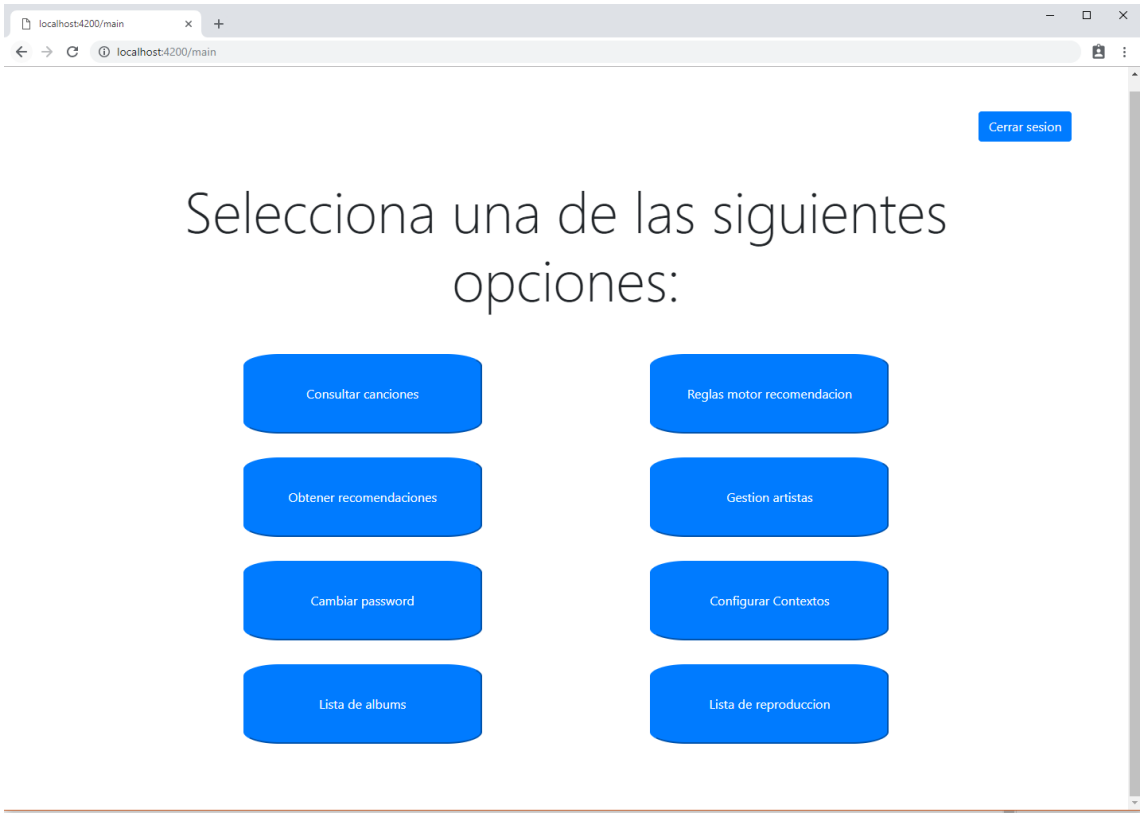

Figura 38 : Login con éxito

10.4 Servicios en el Front-End

Los componentes no deben buscar ni guardar datos directamente. Deben enfocarse en presentar datos de forma correcta y delegar el acceso a datos a un servicio. Los servicios son una excelente manera de compartir información entre clases que no se conocen entre sí. Por ello, hemos introducido diferentes servicios en nuestro Front-End.

De esta forma, los componentes quedan mucho más elegantes y sencillos, centrándose meramente en llamar a una función y en representar los resultados, mientras es la clase servicio de ese componente la que se encarga de llamar al back-end con los parámetros correspondientes.

```
export class GestionartistasPageComponent implements OnInit {

columnDefs = [
    {headerName: 'Artista Id', field: 'id' },
    {headerName: 'Nombre del artista', field: 'nombre' }
];

rowData = [];

  constructor(private navigationService: NavigationService, private
artistaService: GestionartistaService) { }

  ngOnInit() {
    this.getArtists();
  }

    getArtists(): void {
    this.rowData = this.artistaService.getArtists();
  }

}
```

Figura 39 : Componente gestión artistas

Capítulo 11 - Evaluación del sistema de recomendación

Para obtener las recomendaciones, utilizamos un motor basado en implicaciones, desarrollado por el grupo de investigación SICUMA. Los detalles respecto al mismo se pueden encontrar en el *Capítulo 3 – Descripción general del sistema* de esta memoria.

Antes de poder obtener recomendaciones, debemos preparar el sistema, introduciendo para ello una cantidad significativa de ítems, reglas y perfiles o contextos del sistema.

Debemos recordar que los ítems son los elementos o recursos a recomendar por el sistema, las reglas representan las implicaciones del sistema y los perfiles son los contextos.

Se ha elegido una muestra de cien ítems (en nuestro caso, canciones) de características variadas, asegurando así una recomendación de recursos variadas que nos permitan extraer diversas conclusiones respecto al funcionamiento del sistema.

A continuación, vamos a analizar los diferentes atributos que componen un ítem, el cual ha sido elegido al azar. Para la canción “Jesus of Suburbia”, del grupo americano Green Day tenemos los siguientes atributos:

Id	Nombre canción	Artista	Álbum	Año	Genero	Acústica	Bailable	Energía	Habla	Vivacidad
41	Jesus of Suburbia	Green Day	American Idiot	2004	Rock	0.6	0.7	0.3	0.8	0.5

Figura 40 : Atributos que componen el ítem “Jesus Of Suburbia”

Los atributos Id, Artista, Álbum, Año y Genero son descriptivos a nivel interno y no tienen ningún impacto en las recomendaciones. El resto de los atributos son los relevantes de cara al motor de recomendación, pues en función de esos valores una canción determinada será adecuada para un contexto u otro. Cada uno de estos atributos tiene un significado concreto, los cuales son los siguientes:

- Acústica: Un valor de 0.0 a 1.0 de si la canción es acústica. El valor 1.0 representa una alta confianza de que la pista es acústica.

- Bailable: Describe la idoneidad de una pista para bailar en función de una combinación de elementos musicales que incluyen el tempo, la estabilidad del ritmo, la fuerza del latido y la regularidad general. Un valor 0.0 equivale a que la canción es muy poco bailable y un valor de 1.0 representa que la canción es muy bailable.

- Energía: La energía se mide de 0.0 a 1.0 y representa una medida perceptual de intensidad y actividad. Normalmente, las pistas energéticas son rápidas y

ruidosas. Las características perceptuales que contribuyen a este atributo incluyen el rango dinámico, la sonoridad percibida, el timbre y la velocidad de inicio.

- Habla: Mide la presencia de palabras habladas en una canción determinada. Cuantas más palabras se cante en una canción, más cerca estará de 1.0 el valor del atributo. Al contrario, cuanto más se acerque este atributo al 0.0, significará que en la canción se canta poco.

- Vivacidad: Una medida de 0.0 a 1.0 que describe la positividad musical transmitida por una pista. Las pistas con una alta vivacidad suenan positivas, por ejemplo, alegres o eufóricas, mientras que las pistas con baja vivacidad suenan más negativas, como, por ejemplo, triste o enfadado.

Por otra parte, tenemos los contextos del sistema. Para el propósito de la evaluación, se ha elegido nueve contextos diferentes, que ofrecen claras distinciones entre unos y otros. Los mismos se pueden agrupar en tres grupos; actividad que se está realizando en un momento determinado, el momento del día que es y, por último, el estado de ánimo del usuario.

Al primer grupo pertenecen los contextos “estudiar”, “running” y “relajación”.

Al segundo pertenecen “mañana”, “tarde” y “noche”.

Respecto al estado de ánimo, se han seleccionado los contextos “alegre”, “triste” y “nostálgico”.

Cada contexto y parte de una regla (es decir tanto los antecedentes como los consecuentes) llevan asignados un valor que varia entre 0.0 y 1.0, siendo el valor 1.0 una alta representatividad de ese atributo o parte de regla, y siendo 0.0 una nula representatividad del mismo.

Para cada uno de los contextos se determinan unas reglas, las cuales indican al motor de recomendación que ítems son adecuados para un contexto determinado. Estas reglas se establecen tomando en cuenta diferentes factores, y serán los expertos musicales los encargados de retocarlas para que las recomendaciones sean lo más adecuadas posible a cada situación. Por ejemplo, si el contexto es relajación, no sería adecuado que la canción recomendada tuviera un alto valor en energía, ya que eso significa que la canción es muy ruidosa y rápida, justo lo contrario que se busca en un contexto de relajación. Por ello, debe primar el sentido común (y el conocimiento del experto musical) a la hora de elaborar estas reglas.

Hemos establecidos las siguientes reglas para las pruebas:

- Contexto: Estudiar / 0.7.

La regla que se ha elegido en este contexto está pensada para obtener canciones que faciliten la concentración.

Acústica / 0.3, Energía / 0.4 → Habla / 0.6

- Contexto: Running / 0.8

La regla para este contexto tiene como objetivo el obtener canciones con una gran energía, y que estas sean además mayormente instrumentales, con pocas letras.

Bailable / 0.6, energía / 0.8 → Habla / 0.4

- Contexto: Relajación / 0.8

Esta regla persigue obtener canciones tipo acústicas con un alto contenido de letras.

Acústica / 0.5 → Habla / 0.7

- Contexto → Mañana / 0.8

Esta regla hace que obtengamos canciones con una gran vivacidad, y que también tengan una energía moderada, como mínimo.

Vivacidad / 0.8 → Energía / 0.3

- Contexto →Tarde / 0.8

Esta regla hace que obtengamos canciones a partir de una vivacidad moderada, y con un contenido de letras alto.

Vivacidad / 0.3 → Habla / 0.7

- Contexto → Noche /0.8

Esta regla permite obtener canciones apropiadas para la noche, con diversos valores en el atributo bailable y energía., excluyendo las muy poco bailables.

Bailable / 0.3 → Energía / 0.2

- Contexto → Alegre / 0.8

Aplicando estas reglas obtendremos canciones moderadamente bailables, y con gran energía y vivacidad.

Bailable / 0.6 → Energía / 0.9

---------- → Vivacidad / 0.5

- Contexto → triste / 0.8

Aplicando esta regla obtendremos canciones que tengan un mínimo en energía de 0.3 y contenido lírico a partir de 0.5

Energía / 0.3 → Habla / 0.5

- Contexto → nostálgico / 0.8

Con esta regla obtendremos canciones moderadamente bailables y energía variada, adecuadas para este estado de ánimo.

Bailable / 0.4 → Energía / 0.2

Los resultados en las pruebas realizadas son reveladores y bastantes intuitivos, ya que las reglas tienen una clara relación con el contexto.

Para ilustrar esto podemos observar que, con la información anterior que hemos introducido en el sistema, al pedir una recomendación para un usuario con contexto alegre / 0.8 obtenemos los siguientes resultados:

[{

"Nombre canción": "let''s not shit ourselves (to love and be loved)",

"Valores de los atributos": {

"Acústica": 0.6,

" Bailable": 0.9,

" Energía": 0.9,

"Vivacidad": 0.9,

" Habla": 0.4

}, {

"Nombre canción": "helplessness blues",

"Valores de los atributos": {

"Acústica": 0.8,

" Bailable": 0.9,

" Energía": 0.9,

"Vivacidad": 0.8,

" Habla": 0.7

}, {

"Nombre canción": "tamago",

"Valores de los atributos": {
" Acústica": 0.7,
" Bailable": 0.9,
" Energía": 0.9,
"Vivacidad": 0.8,
" Habla": 0.1
}, {
"Nombre canción": "dog problems",
"Valores de los atributos": {
" Acústica": 0.1,
" Bailable": 0.6,
" Energía": 0.8,
"Vivacidad": 0.6,
" Habla": 0.9
}, {
"Nombre canción": "exit music (for a film)",
"Valores de los atributos": {
" Acústica": 0.7,
" Bailable": 0.7,
" Energía": 0.9,
"Vivacidad": 0.5,
" Habla": 0.2
}, {
"Nombre canción": "american girl",
"Valores de los atributos": {
" Acústica": 0.1,
" Bailable": 0.7,
" Energía": 0.9,
"Vivacidad": 0.9,
"Habla": 0.2

```
} ]
```

En total hemos obtenido seis canciones (ítems) de los cien que se encuentran en el sistema. Las canciones obtenidas son las que se ajustan a un contexto de estado de ánimo, de sentirse alegre, ya que, en general, cuando una persona se siente de esa manera, le apetece escuchar canciones que le permitan bailar, que le transmitan energía y que tengan una gran vivacidad.

Esto es lo que hemos conseguido con el ajuste de nuestras reglas. A la hora de ajustar los valores de las reglas, hay que tener en cuenta que cuantas más reglas se añadan y mayor sean los valores, más precisos serán nuestros resultados, pero, como consecuencia, también lo será el número de ítems que serán excluidos de la recomendación. Por el contrario, si somos muy laxos con nuestras reglas, corremos el riesgo de obtener demasiados recursos en nuestra recomendación que no se ajustan realmente al contexto del usuario. Por ello, esta tarea requiere de experiencia y conocimiento para ajustar las reglas de forma adecuada.

Volviendo a nuestro ejemplo, las reglas que corresponde al contexto elegido por el usuario son las siguientes:

- Bailable / 0.6 → Energía / 0.9

- ---------- → Vivacidad / 0.5

Teniendo una muestra de cien ítems, al aplicar las reglas anteriores obtenemos 6 ítems, es decir, solo un 6% del total de los recursos disponibles que pueden ser recomendados.

Si, por ejemplo, en vez de seleccionar el contexto “alegre”, seleccionamos el contexto noche, cuyas reglas son las siguientes:

- Bailable / 0.3 → Energía / 0.2

Obtenemos un total de 66 ítems de los 100 que hay en el sistema; es decir, dos tercias partes de los recursos que hay en el sistema están dentro del grupo de los ítems recomendables.

Esto es debido a que las reglas que se han definido en este contexto son mucho más laxas. Cuantas más reglas se añadan con diferentes atributos a un contexto determinado, o por ejemplo con valores muy altos para algunos de los atributos, más se restringirá los posibles resultados a obtener, por lo que es importante la labor de experto musical para ajustar adecuadamente dichas reglas y sus valores.

Capítulo 12 - Conclusiones

La primera parte de este trabajo consistió en una investigación del estado del arte en cuanto a sistemas de recomendación se refiere. Debido al auge de dichos sistemas, y a su actualidad en gran parte de las aplicaciones más populares, como por ejemplo Netflix [26] o Spotify [27], existen numerosos artículos científicos sobre la materia y diversas aproximaciones, todas interesantes y algunas con mejores resultados que otras.

En dicha primera parte, se trató de exponer, dando una visión general, cuales son algunos de los métodos más comunes y efectivos en dichos sistemas.

Después se llevó a cabo una fase de diseño y análisis, en la que se pormenorizo cada uno de los aspectos de esta y se llevaron a cabo diversos diagramas relevantes.

Acto seguido, se pasó a la parte de desarrollo, la cual ha sido la que más tiempo ha llevado, ya que lo que se proponía no era tarea baladí. Respecto al desarrollo, a nivel académico, ha sido estimulante trabajar con tecnologías web, ya que no es mi especialidad, por lo que ha sido todo un reto en algunos casos y una experiencia enriquecedora.

Por otra parte, me gustaría remarcar que una parte crítica de la utilización de nuestra aplicación es la introducción de las diferentes reglas con sumo cuidado, una tarea que solo debería estar al alcance de un profesional de la música en nuestro caso, pues de ello dependerá en buena parte lo adecuadas que sean las recomendaciones y el éxito de la aplicación en sí.

Con respecto a la metodología utilizada, debemos destacar que Taiga nos ha permitido llevar el desarrollo de forma intuitiva, marcando tareas claras por cada sprint, permitiéndonoslos saber en qué momento del proyecto nos encontrábamos y cuáles eran los siguientes pasos.

También cabe destacar el hecho de que ha sido muy interesante trabajar con un sistema de recomendación, ya que, como se mencionó anteriormente, estos están presentes en buena parte de las aplicaciones que se utilizan diariamente, aun cuando muchas veces no nos fijemos de que estemos obteniendo una experiencia personalizada gracias a un sistema de recomendación, lo que da buena cuenta de la efectividad de estos.

Capítulo 13 – Referencias

13.1 Referencias bibliográficas

[1] G. Adomavicius, A. Tuzhilin, Context-aware recommender systems, in: Recommender Systems Handbook, Springer, 2011, pp. 217–253.

[2]: Geoffray Bonnin and Dietmar Jannach, Automated generation of music playlists: Survey and Experiments, ACM Computing Surveys 47, 2, 26, pp. 1-35, 2014.

[3] J.-J. Aucouturier, F. Pachet, Music similarity measures: what's the use? in: ISMIR, 2002.

[4] J. Wang, A. Vries, M. Reinders, Unifying user-based and item-based collaborative filtering approaches by similarity fusion, in: Proc. SIGIR Conf., 2006, pp. 501–508.

[5] Ke Ji, Runyuan Sun, Wenhao Shu, Xiang Li, Next-song recommendation with temporal dynamics, Knowledge-Based Systems 88, pp. 134-143, 2015.

[6] Kuang Mao, Gang Chen, Yuxing Hu, Luming Zhang, Music recommendation using graph based quality model, Signal Processing 120, pp. 806-813, 2016.

[7] Spotify Technology S.A., Washington, D.C. 20549, Form F-1, Securities and Exchange Commission registration statement under the Securities Act of 1933, pp. 1-3 URL: https://www.sec.gov/Archives/edgar/data/1639920/000119312518063434/d494294df1.htm

[8] Spotify, How can we help you?, Using Spotify, pp. 1-2, 2018

13.2 Referencias Web

[9] Bootstrap. getbootstrap.com.

[10] Eclipse. https://www.eclipse.org/.

[11] JSON. http://www.json.org/.

[12] Maven. maven.apache.org/.

[13] Scrum. https://www.scrum.org/.

[14] Web del grupo SICUMA. http://www.sicuma.uma.es/es/.

[15] Draw. https://www.draw.io/

[16] Docker. https://www.docker.com/

[17] Angular. https://angular.io/

[18] PostgreSQL. https://www.postgresql.org/

[19] Java. https://www.java.com

[20] IntelliJ IDEA. https://www.jetbrains.com/idea/

[21] MagicDraw UML. https://www.nomagic.com/products/magicdraw

[22] UML. http://www.uml.org/

[23] HTML. https://www.w3.org/html/

[24] Balsamiq. https://balsamiq.com/

[25] JavaScript. https://www.javascript.com/

[26] Netflix. https://www.netflix.com/

[27] Spotify. https://www.spotify.com/

[28] Taiga. https://taiga.lcc.uma.es/