TRABALLO FIN DE GRAO
GRAO EN ENXEÑARÍA INFORMÁTICA
MENCIÓN EN ENXEÑARÍA DO SOFTWARE

Recomendación de canciones y listas de reproducción sobre Spotify

Estudante: David Touriño Calvo
Dirección: Javier Parapar López
Daniel Valcarce Silva

A Coruña, xuño de 2020.

A los amantes de las nuevas tecnologías y la evolución del software

Agradecimientos

A mis tutores, Javier Parapar y Daniel Valcarce, por guiarme para elegir un proyecto, y por la paciencia que han tenido conmigo.

A mis compañeros del trabajo, por enseñarme a entender un proyecto real, y darme la oportunidad de aprender y conocer mejor algunas de las herramientas que se utilizan en este proyecto.

A mis amigos, por animarme en mis peores momentos.

Y ante todo a mi familia, por cuidarme, educarme y guiarme hasta este momento.

Resumen

En este proyecto se ha elaborado una aplicación web de recomendación de música sobre *Spotify* que cualquier usuario con una cuenta en la plataforma podrá utilizar para obtener recomendaciones de canciones y listas que añadir a su cuenta.

La aplicación permite explorar las diversas listas de reproducción del usuario, así como sus reproducciones recientes. Cuando un usuario acceda a una página, podrá encontrar instantáneamente música recomendada dependiendo de lo que esté explorando, pudiendo ser en el perfil del usuario en su conjunto, en las canciones que ha escuchado recientemente, o recomendar una canción para una lista de reproducción en concreto, entre otras. El usuario podrá añadir cualquiera de las recomendaciones a su cuenta.

Las recomendaciones se basan en una búsqueda de vecinos, para ello, se utiliza a-NN, una variación de k-NN, un algoritmo de aprendizaje basado en instancias. En el núcleo del sistema se considerará como entrada una lista de reproducción, y su salida serán otras listas de reproducción vecinas. Para recomendar canciones, se hará una suma ponderada de las canciones que contienen las listas de reproducción vecinas.

La aplicación se ha desarrollado como dos artefactos. Uno utilizando el lenguaje de programación Python que contiene el motor del sistema de recomendaciones y expone una serie de servicios internos de recomendación a través de un frontal REST. El otro artefacto será una aplicación web desarrollada en JAVA utilizando el *framework* Spring, que contendrá la interfaz de usuario y estará encargada de orquestar las comunicaciones con el otro artefacto, el API de Spotify y la base de datos.

Los componentes de la aplicación se han identificado como contenedores independientes y se han desplegado en un clúster Kubernetes en la nube de Microsoft Azure.

Abstract

In this project, it has been made a web application of music recommendation over Spotify, which any user with a registered account on the referenced platform could use to obtain recommendations of songs and playlists to add them to the account.

The application allows exploring the different user's playlists, as well as their recently played songs. When a user access to a different view, he'll find instantly recommended music to fit what he is exploring, could be the user's profile, recently'songs, or finding a single song to match an existing playlist, among other. The users add any of these recommendations to their account.

Recommendations are based on a neighbor search, to achieve this, a-NN is used, a k-NN variation, an instance based machine learning algorithm. A playlist will be considered as an input on the system core, and the output will be another neighbour playlists. To recommend songs, a weighted sum of the neighbour playlists songs will be done.

The application has ben developed as two artifacts. One is using the Python programming language, which contains the recommendation system engine and exposes some recommendation services through a REST interface. The otherone artifact will be a Java web application, using the Spring framework, which will contain the user interface and will be the one who coordinates the comunications with the python artifact, the Spotify API and the database.

The application components have been identified as independent containers whose has been deployed in a Kubernetes cluster in the Microsoft Azure cloud.

Palabras clave:

- Recomendaciones
- Música
- Aplicación web
- Java
- Lista de reproducción
- Spring framework
- Scrum
- Python
- Vecinos
- Docker

Keywords:

- Recommendations
- Music
- Web application
- Java
- Playlist
- Spring framework
- Scrum
- Python
- Neighbors
- Docker

Índice general

1 Introdución **1**

1.1 Motivación . . . 2
1.2 Objetivos . . . 2
1.3 Estructura de la memoria . . . 3

2 Fundamentos **5**

2.1 RecSys Challenge 2018 . . . 5
2.1.1 Características del conjunto de datos . . . 5
2.2 Sistema de recomendaciones . . . 6
2.2.1 Filtrado basado en contenido . . . 6
2.2.2 Filtrado colaborativo . . . 6
2.3 Rapidez y eficiencia . . . 7
2.4 Sistema escogido . . . 8
2.4.1 Búsqueda de vecinos . . . 8
2.4.2 WSR . . . 10

3 Metodología y Gestión **13**

3.1 Metodologías ágiles . . . 13
3.1.1 Manifiesto Ágil . . . 13
3.2 Scrum . . . 14
3.3 Roles . . . 15
3.4 Artefactos . . . 16
3.4.1 Product Backlog . . . 16
3.4.2 Sprint Backlog . . . 17
3.4.3 Incremento . . . 17
3.5 Flujo . . . 18
3.5.1 Organización del Product Backlog . . . 18
3.5.2 Reunión de planificación del Sprint . . . 19

3.5.3 Daily Scrum 19
3.5.4 Sprint Review y Retrospectiva del Sprint 19
3.6 Gestión del proyecto 20
3.6.1 Estimación 20
3.6.2 Planificación 20
3.6.3 Recursos 20
3.6.4 Costes 21

4 Tecnologías **23**
4.1 Lenguajes 23
4.1.1 Java 23
4.1.2 Python 24
4.1.3 HTML 24
4.1.4 CSS 24
4.1.5 JavaScript 24
4.1.6 SQL 25
4.2 Librerías 25
4.2.1 Spotify Web API Java 25
4.2.2 Thymeleaf 25
4.2.3 PySparNN 25
4.2.4 Pandas 26
4.2.5 Otras librerías 26
4.3 Frameworks 27
4.3.1 Swagger 27
4.3.2 Flask 27
4.3.3 Spring boot 28
4.3.4 Bootstrap 28
4.3.5 JUnit 28
4.4 Herramientas 29
4.4.1 Taiga 29
4.4.2 Intellij Idea 29
4.4.3 Pycharm 29
4.4.4 Git y GitLab 30
4.4.5 Maven 31
4.4.6 SonarQube 31
4.4.7 Docker 32
4.4.8 Kubernetes 33
4.4.9 Microsoft Azure 33

4.4.10 Redis . . . 33
4.4.11 MySQL . . . 33

5 Desarrollo 35
5.1 Requisitos . . . 35
5.1.1 Requisitos funcionales . . . 35
5.1.2 Requisitos no funcionales . . . 35
5.2 Arquitectura de la aplicación . . . 36
5.2.1 Servidor Web . . . 36
5.2.2 Servicio REST de recomendaciones . . . 36
5.3 Modelo de datos . . . 38
5.3.1 MySQL . . . 38
5.3.2 Redis . . . 39
5.4 Obtención de recomendaciones . . . 40
5.4.1 Transformación de una lista . . . 40
5.4.2 Lista de reproducción ”fusión” . . . 41
5.4.3 Búsqueda de vecinos . . . 41
5.4.4 Aplicación de WSR . . . 41
5.4.5 Incrementos . . . 42
5.5 Desarrollo . . . 42
5.5.1 Sprint 0: Preparación de datos . . . 42
5.5.2 Sprint 1: Primeras recomendaciones . . . 45
5.5.3 Sprint 2: Integración con Spotify . . . 46
5.5.4 Sprint 3 : Visualización de recomendaciones avanzada . . . 50
5.5.5 Sprint 4: Diversidad de recomendaciones . . . 53
5.5.6 Sprint 5: Más tipos de recomendaciones y usabilidad . . . 57
5.5.7 Sprint 6: Mejoras de usabilidad final . . . 59
5.5.8 Acceso a datos . . . 61
5.5.9 Mapeo de entidades de dominio . . . 61

6 Análisis de las recomendaciones 63
6.1 Datos de prueba . . . 63
6.2 Implementación de k-NN . . . 64
6.3 Comparación a-NN frente a k-NN . . . 64
6.4 Tiempos . . . 65
6.4.1 Tiempos de a-NN . . . 65
6.4.2 Tiempos de carga . . . 66

7 Configuración y Despliegue **67**
7.1 Contenedores 67
7.2 Docker y Kubernetes 69
7.2.1 Dockerfile 69
7.2.2 Docker Compose 70
7.2.3 Kubernetes 72
7.3 Despliegue en la nube: Azure 73
7.3.1 Azure Container Registry 74
7.3.2 Azure Files 74
7.3.3 Azure Public IP 75
7.3.4 Azure Kubernetes Service 75

8 Conclusiones **77**
8.1 Valoración final 77
8.2 Lecciones aprendidas 78
8.3 Líneas futuras 78

A Material adicional **83**

Lista de acrónimos **87**

Glosario **89**

Bibliografía **91**

Índice de figuras

2.1 Árbol de búsqueda para a-NN 10

3.1 Esqueleto de Scrum 15
3.2 Burndown chart del proyecto 17
3.3 Sprint Backlog del Sprint 1 18
3.4 Sprint Burndown del Sprint 1 18

4.1 Resultado del análisis de SonarQube 32

5.1 Diagrama de arquitectura de la aplicación 37
5.2 Modelo de datos en MySQL 38
5.3 Distribución de los datos en Redis 39
5.4 Ejemplo de una lista de reproducción del conjunto de datos inicial 44
5.5 Interfaz generada para uno de los servicios REST 47
5.6 Al autenticarse en la aplicación, redirige a la pasarela de Spotify y se pide acceso al usuario a determinadas partes de su cuenta 49
5.7 Listas de reproducción del usuario 50
5.8 Pantalla de detalle de una lista de reproducción, donde se mostrarán las recomendaciones sobre una lista 51
5.9 Listas de reproducción sugeridas 52
5.10 Listas de reproducción sugeridas con las imágenes cargadas 53
5.11 Recomendaciones a listas de reproducción del usuario, añadidas en la pantalla de visualización de sus listas 54
5.12 Pantalla de detalle de una lista de reproducción, donde se mostrarán las recomendaciones sobre una lista 55
5.13 Canciones recomendadas a un usuario 56
5.14 Recomendaciones según las reproducciones recientes 56
5.15 Pantalla de búsqueda de canciones 58

5.16 Pantalla de búsqueda de listas de reproducción 59
5.17 Al añadir una canción, se muestran las playlists del usuario en un modal. . . . 60
5.18 Botón de borrado de canciones al final de la lista. 60
5.19 Diagrama de secuencia para mostrar una lista de reproducción (simplificado) . 62

7.1 Diagrama de infraestructura de los sistemas en Azure 68
7.2 Diferencia entre el uso de contenedores o de máquinas virtuales 69
7.3 Recursos creados por Azure para AKS 76

Índice de cuadros

2.1 Estadísticas del conjunto de datos 6

3.1 Backlog del proyecto finalizado 16
3.2 Estimación de costes para los recursos humanos del proyecto 21
3.3 Desglose del coste de los recursos humanos 21
3.4 Desglose de los costes materiales 21
3.5 Coste mensual Azure 22
3.6 Comparativa entre dos máquinas de Azure 22

6.1 Comparación a-NN - Resumen 64
6.2 Tiempos a-NN y k-NN 65
6.3 Tiempos de carga de la web en `/playlist` para los diferentes valores de *k-clústeres* 66

A.1 Calidad a-NN - Desglose 85

Capítulo 1

Introdución

Las plataformas digitales se extienden cada vez con más facilidad a todos los usuarios del mundo, plataformas como *YouTube*, *Amazon* o *Netflix* empiezan a formar parte del día a día de muchas personas. Según la *Statista Global Consumer Survey*, un 70% de los encuestados en España admitían haber realizado compras en Amazon en el último año [1]. En otras plataformas como YouTube, su uso se extiende a prácticamente la totalidad de la población con acceso a Internet. Una de las claves del éxito de estas plataformas han podido ser sus **recomendaciones**. Todos estos servicios ofrecen a los usuarios una experiencia personalizada para motivarles a seguir consumiendo.

Durante mucho tiempo las recomendaciones han existido en manos de amigos y familiares, que ofrecían alternativas de compra, libros similares, películas u otros medios de entretenimiento y servicios, todo esto es posible gracias al conocimiento que se tiene sobre el individuo y sobre el mundo que lo rodea. Al trasladar el conocimiento a las máquinas pueden nacer los sistemas de recomendación. Basándose en sus experiencias anteriores, usuarios similares, o cualidades de los productos que consumen, las distintas plataformas son capaces de ofrecer más contenido al usuario, que repercute en un beneficio del servicio al autopromocionarse y enriquecer la participación.

Según el centro de investigaciones Pew, hasta un 81% de los usuarios de YouTube visualizan alguna vez los vídeos que le recomienda la plataforma [2]. Estas cifras se corresponden también a los datos que nos ofrece YouTube, que en el CES 2018 comunicó que un 70% de las visualizaciones provenían de su sistema de recomendaciones [3]. Por otro lado, la revista *Fortune* nos dice que hasta un 60% [4] de las recomendaciones en *Amazon* pueden acabar convertidas en compras. En otras plataformas como *Netflix*, las recomendaciones son el 75% del contenido que visualizan los usuarios [5].

Por estas razones, al ser una funcionalidad clave y computacionalmente compleja, analizaremos las posibilidades de mejora algorítmica del propio proceso de generación de las recomendaciones y crearemos un servicio de recomendaciones en el campo de la música, en

concreto sobre la plataforma de Spotify.

1.1 Motivación

Descubrir nueva música siempre ha sido una motivación para todo tipo de usuarios. Cuando hablamos de recomendaciones de música, ya no es el servicio intentando venderle algo al usuario, como es en el caso de *Amazon*, sino que se identifica más con los servicios como *Netflix*, donde el usuario toma más protagonismo y quiere descubrir nuevos títulos que complementen sus gustos.

Estos sistemas se vuelven importantes tanto para el usuario como la compañía, que observa que las recomendaciones influyen directamente en el uso de la plataforma. En plena era de transformación digital, las empresas que adapten estos servicios tendrán más ventajas frente al resto, y el usuario estará más contento.

Pero implementar estos sistemas no es trivial, tenemos diversas estrategias que pueden traer problemas de escalado, o cómputos que se pueden demorar demasiado al tener una gran complejidad o por causa de ser un problema con un tamaño considerablemente elevado.

Debido a esto queremos analizar las posibilidades de mejora algorítmica del propio proceso de generación de las recomendaciones. Para ello, adaptaremos una implementación estado del arte de filtrado colaborativo [6] a la tarea de recomendación de canciones. Además, abordaremos también una posible mejora en el proceso de construcción de vecindarios. La elección de vecinos en los algoritmos de filtrado colaborativo se ha demostrado clave en la calidad de los mismos [7]. Pretendemos con este proyecto reexaminar un algoritmo clásico de construcción de vecinos k-Neareast Neighbor (k-NN) [8] por una variante que mejora considerablemente la eficiencia del proceso: Approximate Neareast Neighbor (a-NN) [9].

En este proyecto queremos abordar también la mejora en la eficiencia de construcción de vecindarios en los métodos de filtrado colaborativo. Para ello proponemos el uso de métodos aproximados de construcción de vecindarios. Estos métodos suponen una mejora importante en los tiempos de cómputo de usuarios similares a costa de proporcionar una solución aproximada.

1.2 Objetivos

Se desarrollará un motor de recomendaciones a partir del conjunto de datos de Spotify de un millón de listas de reproducción que liberó para el *RecSys Challenge 2018* [10]. Este conjunto de datos se transformará y explotará de diversas formas dentro de la aplicación, sirviendo tanto para obtener recomendaciones, como para obtener información de las mismas.

El principal objetivo es probar métodos de generación de vecindarios más eficientes [9] y

apoyarnos en estrategias de filtrado colaborativo [6] para reducir tiempos de recomendación y poder calcularlas en tiempo real sobre grandes cantidades de datos.

Como objetivo final se ofrecerá una plataforma web donde cualquier usuario de Spotify pueda obtener recomendaciones de música, además de permitirle administrar su cuenta añadiendo o quitando canciones de sus listas de reproducción. Deberán poder obtenerse recomendaciones de canciones o listas de reproducción, en base a:

1. Listas de reproducción o canciones.

2. Música escuchada recientemente.

3. Composición de las listas de un usuario.

Las recomendaciones deben poder obtenerse rápidamente y tener calidad, estando fuertemente relacionadas con la referencia de la recomendación.

Esta plataforma se desplegará en la nube pública de Azure junto a toda la infraestructura de la aplicación.

1.3 Estructura de la memoria

Se seguirá el desarrollo de este proyecto a través de los 8 capítulos detallados a continuación.

1. Introducción: Se plantea el proyecto, explicando los objetivos y la motivación del mismo de forma global. Muestra la estructura de la memoria.

2. Fundamentos: Se explican los fundamentos técnicos del núcleo de la aplicación, el sistema de recomendaciones. Introduce diferentes técnicas de filtrado y búsqueda de vecinos.

3. Metodología y Gestión: Se habla de la metodología y su aplicación y gestión en el proyecto.

4. Tecnologías: Se muestran las principales herramientas, librerías, lenguajes y frameworks empleados en el desarrollo del proyecto.

5. Desarrollo: Se explica como se ha diseñado la arquitectura del sistema, el diseño del *software* y cómo se ha implementado y probado a lo largo del proyecto en Scrum, con cada historia de usuario por *Sprint*.

6. Análisis de las recomendaciones: Se estudia la diferencia de a-NN frente a k-NN y cómo afecta a los tiempos y a la calidad de las recomendaciones.

7. Configuración y despliegue: Se explican los fundamentos de los contenedores y Docker y cómo se ha aplicado en el proyecto. Se explica la configuración del despliegue en producción en la nube de Azure.

8. Conclusiones: Se exponen las conclusiones, lo que se ha aprendido, y cómo podría seguir evolucionando el proyecto.

Capítulo 2

Fundamentos

La base del proyecto será su sistema de recomendaciones. Para este proyecto se ha escogido un sistema de recomendaciones de filtrado colaborativo basado en vecindario basado en usuarios, que se explicará a continuación.

2.1 RecSys Challenge 2018

En 2018, Spotify, junto a varias universidades, organiza el RecSys Challenge 2018. El RecSys Challenge [11] es una competición anual que se centra en encontrar la mejor aproximación de recomendaciones para un escenario en concreto. Durante los años, ha ido ganando participantes tanto del entorno empresarial como del académico.

Como parte de este desafío, Spotify libera un gran conjunto de datos de listas de reproducción asociadas a canciones. El objetivo del desafío consiste en añadir una o varias canciones a una lista de reproducción, de forma que encajasen en las características de la lista original. Para ello, los participantes tienen que predecir, para una lista de reproducción, una lista ordenada de 500 canciones recomendadas [10].

En este proyecto hemos utilizado este conjunto de datos liberado por Spotify.

2.1.1 Características del conjunto de datos

El conjunto de datos liberado, nombrado The Million Playlist Dataset (MPD), consiste en un millón de listas de reproducción creadas por los usuarios de la plataforma. El MPD incluye, para cada lista de reproducción, su nombre así como una lista de canciones incluyendo álbumes y nombres de artistas. También contiene metadatos como las URIs de Spotify y el número de seguidores de cada lista. Este conjunto de datos ocupaba un total de 30 GB, y estaba compuesto por mil ficheros en formato JavaScript Object Notation (JSON), cada uno con mil listas de reproducción. En el cuadro 2.1 se pueden ver las estadísticas del conjunto de datos.

Característica	**Valor**
Número de listas de reproducción	1,000,000
Número de canciones	66,346,428
Número de canciones únicas	2,262,292
Número de álbumes únicos	734,684
Número de artistas únicos	295,860
Número de nombres de listas de reproducción únicos	92,944
Número de nombres de listas de reproducción únicos normalizados	17,381
Duración media de una lista de reproducción (canciones)	66.35

Cuadro 2.1: Estadísticas del conjunto de datos

2.2 Sistema de recomendaciones

Un sistema de recomendaciones [12] es un conjunto de herramientas software que provee sugerencias de *objetos* que pueden aportar valor a un usuario. Esos objetos pueden ser películas, objetos en venta, noticias, usuarios o incluso listas de reproducción y canciones, los objetos que se recomendarán en este proyecto.

Para ofrecer estas recomendaciones, tenemos diferentes técnicas que seguir, a continuación se explican las dos principales y en concreto el filtrado colaborativo, la utilizada en este proyecto.

2.2.1 Filtrado basado en contenido

El sistema recomienda los objetos en base a otros objetos parecidos que al usuario le hayan gustado en el pasado. La similitud de los objetos se comparan por sus características; por ejemplo, si a un usuario le han gustado las canciones de Jazz, el sistema le recomendará otras canciones de Jazz.

2.2.2 Filtrado colaborativo

El sistema más básico de Collaborative Filtering (CF) recomendará objetos que hayan gustado a otros usuarios con gustos parecidos. La similitud de gustos entre los dos usuarios se basará en la similitud del historial de los usuarios. Esta técnica se considera la más popular [12] y mayormente implementada. Los sistemas de recomendaciones de filtrado colaborativo pueden ser basados en modelos, en vecindario o híbridos.

Sistemas basados en modelo

Los sistemas basados en modelo intentan crear un modelo que se explota mediante diversos algoritmos probabilísticos. Se computa el valor esperado de una predicción del usuario basándose en las puntuaciones que ha dado para otros objetos.

Sistemas basados en vecindario

Los sistemas basados en vecindario, también llamados basados en memoria usan las puntuaciones de los usuarios para calcular la similitud entre usuarios u objetos. Se suelen utilizar para encontrar las mejores N recomendaciones. Existen 2 aproximaciones para estos sistemas.

- Basado en Objetos: Construye una matriz de objeto x objeto que contrasta contra el usuario.

- Basado en Usuarios: Su objetivo es encontrar usuarios similares a partir de los cuales extraer recomendaciones. Es el sistema elegido en este proyecto. Para implementarlo se ha utilizado k-NN, explicado en la sección 2.4.1 y Weighted Sum Recommender (WSR), explicado en la sección 2.4.2.

2.3 Rapidez y eficiencia

El usuario de la aplicación espera tanto rapidez como calidad a la hora de obtener las recomendaciones, es importante encontrar una solución rápida y preferiblemente en tiempo real. Además, es importante ofrecer una solución escalable, el mundo continúa creciendo, los datos cambian, y podrían llegar a ralentizar el cálculo de las recomendaciones. Un método para ofrecer una solución escalable podría ser plantear el sistema para que las operaciones más computacionalmente costosas se ejecutasen de manera distribuida. Valcarce et al. [13] proponen una solución para la escalabilidad de un sistema de recomendación existente, donde traslada el mayor coste computacional a dentro del framework MapReduce, lo que permite crear una solución paralelizable y escalable acorde al crecimiento del modelo de recomendaciones. De manera alternativa, también se puede optar por mejorar los algoritmos para reducir el tiempo de cómputo. Valcarce et al. [14] estudian y aplican diferentes algoritmos de filtrado colaborativo que acaban mejorando considerablemente el tiempo de computación e incluso ofrecen recomendaciones más variadas. Valcarce et al. [6] plantean como instrumentalizar los métodos clásicos de *ranking* sobre índices invertidos bajo Modelos de Lenguaje Estadísticos para computar de manera eficiente los vecindarios como si de una búsqueda se tratase.

2.4 Sistema escogido

Se ha escogido un sistema de recomendaciones de Collaborative Filtering basado en vecindario basado en usuarios. Se escoge un sistema basado en vecindario por la naturaleza del conjunto de datos. Además de que este método tiene bastante fiabilidad, el conjunto de datos no nos permite obtener características de una canción más allá de su género. Por otro lado el hacer la búsqueda de vecinos en memoria no supone un gran problema ya que el conjunto de datos no es excesivamente grande.

Se aplicaron varias técnicas a la hora de obtener recomendaciones, que son generalizables para distintos casos de uso de la aplicación.

Por un lado, tenemos la búsqueda de vecinos, que se explica en 2.4.1, donde se encontrarán listas de reproducción similares. A estas listas, aplicaremos Weighted Sum Recommender (2.4.2) para encontrar canciones similares.

El usuario podrá entonces obtener recomendaciones de listas, para lo que sólo se hará la búsqueda de vecinos, o de canciones, para lo que adicionalmente se aplica WSR.

2.4.1 Búsqueda de vecinos

La búsqueda de vecinos cercanos es un problema para el que dado un conjunto de puntos en un espacio, se puede obtener de forma rápida el subconjunto de puntos más cercanos a un punto de entrada que pertenezca al mismo espacio. Este problema se puede aplicar a una gran variedad de aplicaciones, entre ellas, la búsqueda de recomendaciones.

En el caso de este proyecto, los objetos serán listas de reproducción que serán tratadas como vectores de N dimensiones, tantas como canciones presenta el conjunto de datos.

Se enumeran las canciones del conjunto de datos a través de un índice de 0 a 2262291, para cada lista de reproducción se crea un vector de tamaño 2262291 compuesto por ceros. En las posiciones cuyo índice corresponda a una canción presente en esa lista, se marca con un uno.

De esta forma, tenemos cada uno de los puntos del conjunto de datos representados como un vector, que uniéndolos en una matriz, comprenderá nuestro espacio de búsqueda. La función de evaluación de similitud entre dos listas de reproducción será la distancia coseno entre los dos vectores, que se ha demostrado que destaca frente a otras métricas ante el problema de las mejores N recomendaciones [6].

Este espacio de búsqueda se creará y operará como una matriz dispersa, debido a que una lista de reproducción por lo general tendrá un porcentaje muy pequeño de las canciones totales del conjunto de datos, por lo que la mayor parte de los valores de los vectores serán ceros.

La explotación de estos datos a nivel técnico se explica en el capítulo 5, Desarrollo , apartado 5.5.1.

k-Neareast Neighbor (k-NN)

k-Neareast Neighbor o en español, *k-vecinos más próximos* es un tipo de aprendizaje basado en instancia [8]. En k-NN intentamos encontrar los usuarios más parecidos a otro. En nuestro caso las listas de reproducción tomarán el papel de usuarios, siendo compuestas por varias canciones.

Su fase de entrenamiento consiste en almacenar vectores en un espacio multidimensional. Para recuperar los vecinos, ante un nuevo punto o vector que no esté en el espacio inicial, se buscan los puntos más cercanos. Para medir la distancia de un punto a otro, se suele tomar la distancia euclídea.

Como el tamaño del vector no es un punto relevante en este proyecto, utilizaremos la distancia coseno para calcular la distancia.

k-NN asume que puntos cercanos los unos de los otros serán parecidos. Podemos aplicarlo a nuestro proyecto, ya que las distintas listas de reproducción serán más cercanas entre sí cuantas más canciones en común tengan.

Sin embargo, como estamos ante un espacio dimensional grande, las búsquedas utilizando k-NN se hacen pesadas [15], por lo que se plantea utilizar Approximate Neareast Neighbor.

Approximate Neareast Neighbor (a-NN)

a-NN es una variante de k-Neareast Neighbor que se basa en buscar los k-vecinos más cercanos de manera aproximada, reduciendo el tiempo de computación a cambio de una menor calidad de búsqueda [9].

Originalmente, tratar de encontrar los vecinos más cercanos en nuestro conjunto de datos implicaría buscar en todo el conjunto, calculando la distancia coseno de una lista entrada con todas las listas del conjunto de datos, lo que implicaría una operación con una complejidad de O(n). Como es un espacio de búsqueda grande (1 millón de listas de reproducción), el rendimiento se vería afectado [15].

Utilizar una implementación de a-NN no garantiza que vaya a encontrar los vecinos más cercanos, pero en este proyecto, se puede permitir una buena estimación en lugar de los mejores para poder mejorar significativamente el rendimiento.

Entonces, para realizar búsquedas aproximadas, podemos crear un árbol de búsqueda con $\sqrt{n}$ nodos líderes seleccionados aleatoriamente para estar en el nivel más alto del árbol. Después, para el resto del conjunto de datos, podemos asignarlos a uno de los nodos líderes, pasando a ser un nodo seguidor.

Con este árbol de búsqueda, en lugar de buscar en todo el conjunto, podemos buscar entre los primeros $\sqrt{n}$ nodos líderes, y los nodos seguidores de éste. Esto permite que las búsquedas se realicen en O($2 \cdot \sqrt{n}$)en lugar de O(n), a costa de que se puedan perder vecinos más cercanos. La figura 2.1 muestra esta distribución gráficamente

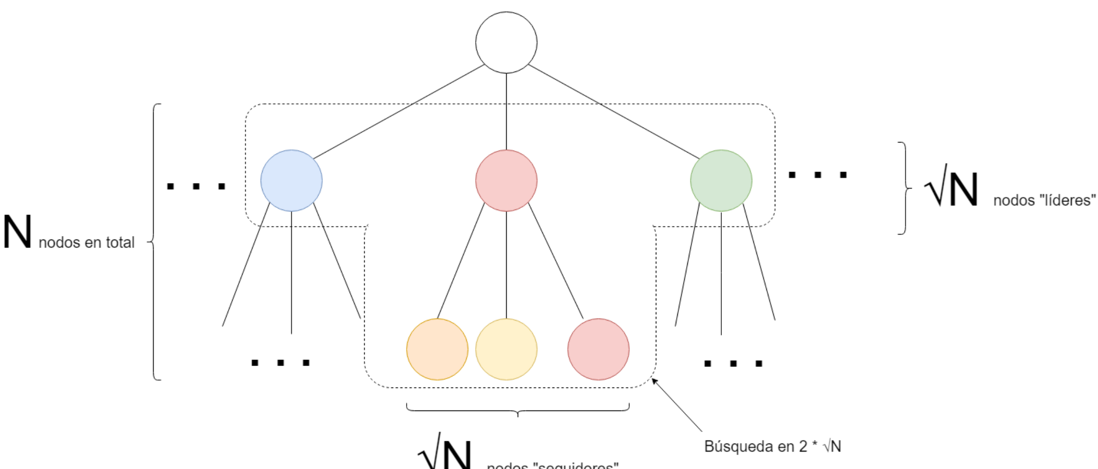


Figura 2.1: Árbol de búsqueda para a-NN

Este tipo de búsquedas es generalizable para hacer k niveles, convirtiéndose la complejidad en O($k \cdot \sqrt[k]{n}$). A esta técnica se le llama *Cluster Pruning* [8].

Chierichetti et al. [16] estudia la aplicación de *cluster pruning* para la búsqueda de vecinos cercanos. Se utilizan tanto esquemas teóricos como experimentos donde se muestra que el uso de esta técnica ahorra considerablemente el costo computacional a la vez que compromete levemente la calidad de los objetos recuperados.

2.4.2 WSR

Si bien para recomendar canciones de listas de reproducción nos bastaba con encontrar los vecinos, para poder recomendar canciones hará falta analizar esos vecinos y recomendar las mejores canciones que contengan.

Para ello se emplea WSR [6], una suma ponderada para determinar las canciones más aptas para recomendar.

$$s(p, s_i) = \sum_{q \epsilon \gamma} sim(p, q) \cdot score(q, s_i) \tag{2.1}$$

p : Playlist original para la cual se quieren hallar recomendaciones
γ : Conjunto de playlists vecinas de la playlist p
S: Conjunto de canciones que forman los vecinos de forma: $S = \bigcup_{i=1}^{\infty} \gamma_i$
$s(p, s_i)$: Similitud de una canción s_i con una playlist p
$sim(p, q)$: Similitud de una playlist p con una playlist q
$score(q, s_i)$: Puntuación de la canción s_i dentro de la playlist q

El objetivo es obtener un vector $\vec{s}(p, S)$ con las mejores canciones para la lista de reproducción original según la cual se están hallando vecinos. Para ello, podemos construir una matriz $M = |\gamma| \times |S|$ que representará las diferentes puntuaciones ($score(\gamma_k, s_i)$) de una canción sobre una lista de reproducción. Estas puntuaciones serán binarias, 1 si la canción está presente en la lista de reproducción, y 0 si no está presente.

$$M = \begin{array}{c} \\ \gamma_1 \\ \gamma_2 \\ ... \\ \gamma_{n-1} \\ \gamma_n \end{array} \begin{array}{c} \begin{array}{ccc} s_1 & & s_m \end{array} \\ \begin{pmatrix} score(1,1) & ... & score(1,m) \\ score(2,1) & ... & score(2,m) \\ ... & ... & ... \\ score(n-1,1) & ... & score(n-1,m) \\ score(n,1) & ... & score(n,m) \end{pmatrix} \end{array} \tag{2.2}$$

Y a partir del vector de similitud entre γ_k y p, obtenido a partir de la distancia coseno entre las listas de reproducción vecinas y la lista de reproducción origen.

$$\vec{sim}(p, \gamma) = \begin{array}{c} \begin{array}{ccccc} \gamma_1 & \gamma_2 & & \gamma_{n-1} & \gamma_n \end{array} \\ \begin{pmatrix} sim(p, \gamma_1) & sim(p, \gamma_2) & ... & sim(p, \gamma_{n-1}) & sim(p, \gamma_n) \end{pmatrix} \end{array} \tag{2.3}$$

Finalmente, podemos aplicar la suma ponderada descrita en 2.1 al hacer el producto escalar entre 2.3 y 2.2.

$$\vec{s}(p, s) = \vec{sim}(p, \gamma) \cdot M \tag{2.4}$$

Con lo que tenemos en $\vec{s}(p, s)$ las puntuaciones de cada canción para la lista de reproducción objetivo.

Capítulo 3

Metodología y Gestión

Para la elaboración de este proyecto software se utilizará una metodología de desarrollo. Se ha empleado una adaptación de Scrum, ya que por cómo está estructurado Scrum no es posible aplicar la metodología de forma fiel en este proyecto.

3.1 Metodologías ágiles

En los últimos años las metodologías ágiles se está extendiendo. Utilizan ciclos iterativos para entregar software funcional en cada ciclo. Con esto formamos productos en continua evolución y mejora. Gracias a estas metodologías, se forma un producto utilizable en las primeras semanas, con las necesidades fundamentales del cliente.

3.1.1 Manifiesto Ágil

En marzo de 2001, 17 profesionales se reunieron convocados por Kent Beck, que había publicado unos años antes un libro donde explicaba una nueva metodología, Extreme Programming.

Los participantes resumieron en 4 postulados los valores donde se asientan los métodos ágiles, que nacían como alternativa a las metodologías formales.

Los postulados, eran:

1. Valorar a los individuos y su interacción por encima de los procesos y herramientas.
2. El software que funciona, por encima de la documentación exhaustiva.
3. La colaboración con el cliente, por encima de la negociación contractual.
4. La respuesta al cambio, por encima del seguimiento de un plan.

El manifiesto ágil, tras estos 4 postulados, establece 12 principios:

1. Nuestra principal prioridad es satisfacer al cliente a través de la entrega temprana y continua de software de valor.

2. Son bienvenidos los requisitos cambiantes, incluso si llegan tarde al desarrollo. Los procesos ágiles se doblegan al cambio como ventaja competitiva para el cliente.

3. Entregar con frecuencia software que funcione, en periodos de un par de semanas hasta un par de meses, con preferencia en los periodos breves.

4. Las personas del negocio y los desarrolladores deben trabajar juntos de forma cotidiana a través del proyecto.

5. Construcción de proyectos en torno a individuos motivados, dándoles la oportunidad y el respaldo que necesitan y procurándoles confianza para que realicen la tarea.

6. La forma más eficiente y efectiva de comunicar información de ida y vuelta dentro de un equipo de desarrollo es mediante la conversación cara a cara.

7. El software que funciona es la principal medida del progreso.

8. Los procesos ágiles promueven el desarrollo sostenido. Los patrocinadores, desarrolladores y usuarios deben mantener un ritmo constante de forma indefinida.

9. La atención continua a la excelencia técnica enaltece la agilidad.

10. La simplicidad como arte de maximizar la cantidad de trabajo que no se hace, es esencial.

11. Las mejores arquitecturas, requisitos y diseños emergen de equipos que se autoorganizan.

12. En intervalos regulares, el equipo reflexiona sobre la forma de ser más efectivo y ajusta su conducta en consecuencia

A pesar de no poder seguir estos principios debido a las condiciones del proyecto, siendo desarrollado por una única persona y en periodos de tiempo no constantes, se han intentado seguir en lo posible estos principios, aplicando para ello la metodología Scrum.

3.2 Scrum

Scrum [17] es un método de desarrollo ágil de software nacido en la década de 1990 por Jeff Sutherland y su equipo de desarrollo. En los últimos años Ken Schwaber y Jeff Sutherland han continuado desarrollando la metodología. [18].

Sigue los principios del manifiesto ágil y se basa en un proceso iterativo e incremental.

Representamos el esqueleto de Scrum en la figura 3.1. La entrada es una lista de requisitos. El círculo inferior representa una iteración del desarrollo de actividades que ocurren una tras otra. El círculo superior representa las reuniones diarias que realiza el equipo para comprobar el estado del desarrollo. La salida de cada iteración es un incremento del producto. Este ciclo continúa hasta que el proyecto termina.

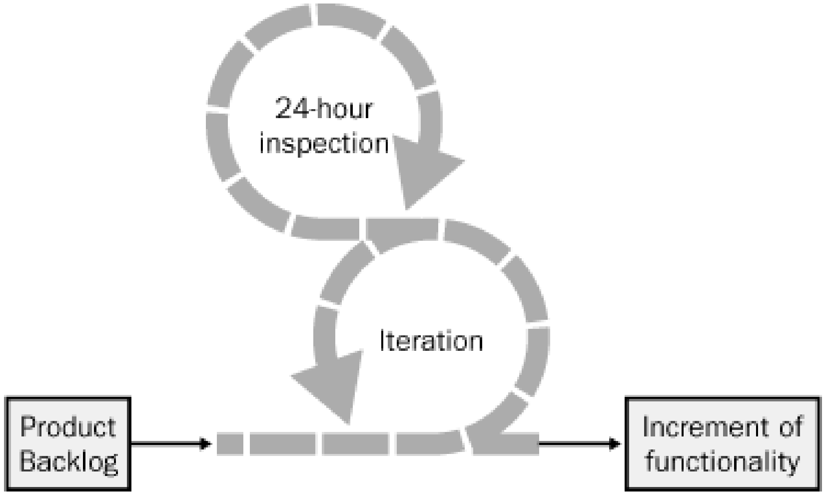


Figura 3.1: Esqueleto de Scrum

3.3 Roles

Existen 3 roles a los que se delega toda la responsabilidad de la administración del proyecto: El **Product Owner**, el **Equipo de desarrollo** y el **Scrum Master**.

- El **Product Owner** es el responsable de maximizar el valor del producto que resulte del trabajo del equipo de desarrollo. Es responsable de administrar la lista de requisitos, el *Product Backlog*, explicado en la sección 3.4, para asegurarse que las funcionalidades más importantes se crean primero. Ésto lo consigue priorizando la cola en cada iteración.
- El **Equipo de desarrollo** es el responsable de desarrollar las funcionalidades. Los equipos son autoorganizables, y son responsables convertir el *backlog* en un incremento de la funcionalidad. El equipo es el responsable del éxito de una iteración y del proyecto en su conjunto.
- El **Scrum Master** es el responsable del proceso de Scrum. Debe enseñar Scrum a todo el mundo involucrado en el proyecto y asegurarse de que todos siguen sus reglas y prácticas.

En este proyecto los roles se han representado de la siguiente forma:

- Product Owner: Javier Parapar y Daniel Valcarce
- Scrum Master: David Touriño
- Equipo de desarrollo: David Touriño

3.4 Artefactos

Scrum introduce una serie de nuevos artefactos que son utilizados en su proceso.

3.4.1 Product Backlog

Es un listado de requisitos del proyecto. El Producto Owner es el responsable de su contenido, priorización y disponibilidad del mismo. Nunca está completo y es simplemente una estimación inicial de los requisitos. El Product Backlog es dinámico y se adapta a las necesidades del usuario. En la figura 3.1 se muestra el backlog de este proyecto.

ref	Nombre	Sprint	Puntos
4	Registrarse con Spotify	2	8
5	Buscar canciones	5	20
7	Reproducir una canción	4	5
8	Administrar canciones de las listas de reproducción	6	20
9	Preparar datos del servicio de recomendaciones	0	40
12	Obtener Recomendaciones de canciones para una playlist	1	20
13	Obtener Recomendaciones de playlists según canciones escuchadas recientemente	4	13
14	Añadir listas de reproducción	6	20
24	Obtener Recomendaciones de playlists según otra playlist	1	8
25	Recomendar canciones dadas reproducciones recientes	5	8
26	Recomendar canciones a partir de todas las playlists de un usuario	4	8
27	Recomendar playlists a partir de todas las playlists de un usuario	1	13
38	Mostrar recomendaciones de playlists según otra playlist	3	20
39	Mostrar recomendaciones de canciones para una playlist	4	8
40	Mostrar recomendaciones de playlists a partir de todas las playlists de un usuario	4	8
65	Visualizar recomendaciones sin estar registrado	5	10
66	Buscar listas de reproducción	5	8

Cuadro 3.1: Backlog del proyecto finalizado

Un *Burndown chart* muestra la cantidad de trabajo pendiente en el tiempo. Es útil para ajustar el trabajo incluyendo o retirando tareas.

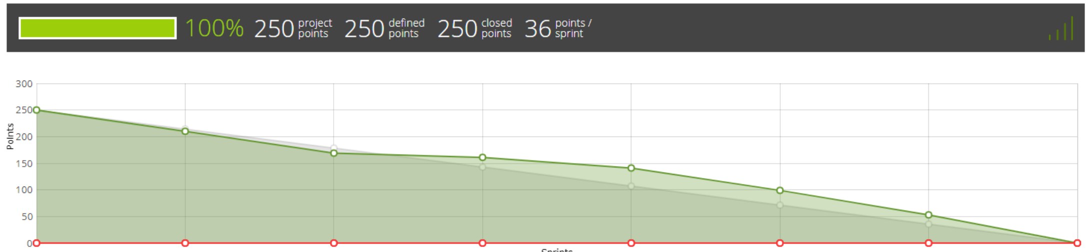


Figura 3.2: Burndown chart del proyecto

En la figura 3.2 se puede ver el Burndown chart de este proyecto. Se puede apreciar una subida de la deuda de puntos de historia en el tercer y cuarto Sprint debido a asuntos personales del alumno que impidió entregar todo lo planeado en el Sprint, por lo que se forzó a realizar algunos puntos historia más en los siguientes Sprints.

3.4.2 Sprint Backlog

El Sprint Backlog es el conjunto de tareas seleccionadas a partir del Product Backlog que se puede convertir en un incremento. En la reunión de planificación del Sprint, el equipo recopila la lista inicial de estas tareas. Estas tareas deben ser divididas de forma que empleen entre 4 y 16 horas. Si se declaran de más duración, serán simplemente tareas temporales mientras no se hayan definido correctamente.

Solo el equipo puede cambiar el Sprint Backlog. Debe ser visible facilmente y mostrar en tiempo real lo que el equipo planea alcanzar durante el Sprint.

Durante el desarrollo del proyecto, para organizar el Sprint Backlog se ha usado Taiga, explicado en el capítulo 4. En la figura 3.3 se puede ver la pantalla del Sprint Backlog para el Sprint 1, con todas las tareas divididas y completadas.

3.4.3 Incremento

Scrum requiere que se construya un incremento de la funcionalidad del producto cada Sprint. Todas las tareas que se han terminado en el Sprint forman el incremento. Podríamos referirnos a un incremento como una versión, aunque no sea entregada.

En la figura 3.4 se puede ver el desarrollo del incremento del Sprint 1.

SPOTIFYRECSYS SPRINT 1 07 APR 2018-21 APR 2018

100% 41 total points 41 completed points 0 open tasks 10 closed tasks 0 iocaine doses

USER STORY | NEW | IN PROGRESS | DONE

#24 Obtener recomendaciones de playlists según otra playlist
Done
8 points

#28 Instalar PySparNN
#37 Reemplazar vecinos subset del original
#29 Ajustar k_clusters
#30 Refactorizar y ordenar codigo
#31 Probar la búsqueda de vecinos

#12 Obtener recomendaciones de canciones para una playlist
Done
20 points

#35 Obtener las canciones con prod. escalar más alto
#33 Obtener vector de similaridades para cada playlist
#34 Construir matriz de playlists sugeridas x canciones sugeridas
#36 Refactorizar y organizar código

#27 Recomendar playlists dadas todas las playlists del usuario
Done
13 points

#32 Crear playlist fusión

Figura 3.3: Sprint Backlog del Sprint 1

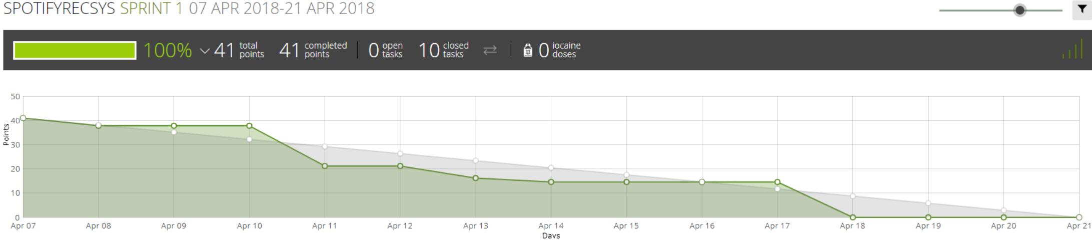


Figura 3.4: Sprint Burndown del Sprint 1

3.5 Flujo

3.5.1 Organización del Product Backlog

Un proyecto de Scrum comienza con una visión de un sistema a desarrollar, puede ser una vision vaga al principio, pero se aclarará según avance el proyecto. El Product Owner se encargará de convertir esta visión de forma que maximice la rentabilidad del proyecto. Para ello recurrirá al Product Backlog, la lista de requisitos funcionales y no funcionales que darán forma a esa visión vaga.

El Product Backlog se priorizará de forma que la mayoría de elementos más importantes estarán en la cima, y se repartirán entre las siguientes versiones. El Product Backlog y sus contenidos suelen cambiar en el momento en que el proyecto empieza.

3.5.2 Reunión de planificación del Sprint

El trabajo se hace en cada Sprint, una iteración de X días consecutivos. Cada Sprint se inicia con una reunión de planificación del Sprint donde colaboran el Product Owner y el equipo para decidir qué se hará durante el próximo Sprint. Empezando con la tarea de máxima prioridad del backlog, el Product Owner comunica al equipo lo que se necesita, y el equipo comunica cuánto de lo que se necesita se cree que se puede convertir en una funcionalidad para el Sprint. Estas reuniones no pueden durar más de 8 horas.

La reunión de planificación del Sprint tiene 2 partes.

En las primera parte el Product Owner presenta las tareas de más alta prioridad al equipo. El equipo discute sobre el contenido, significado e intenciones del Product Backlog. El equipo selecciona todas las tareas del Product Backlog que creen que se pueden convertir en un incremento de la funcionalidad potencialmente entregable al acabar el Spring.

Durante la segunda parte, el equipo planea el Sprint. Las tareas que se llevarán a cabo son colocadas en el Sprint Backlog.

3.5.3 Daily Scrum

Cada día el equipo se reúne en el *Daily Scrum*, una reunión de 15 minutos donde cada miembro del proyecto donde cada miembro responde a las tres preguntas:

- ¿Qué has hecho desde el último *Daily Scrum*?
- ¿Qué planeas hacer desde ahora hasta el próximo *Daily Scrum*?
- ¿Qué impedimentos tienes o vas a tener que impida cumplir el objetivo del Sprint?

Su propósito es sincronizar el trabajo de todos los miembros del equipo diariamente y concertar cualquier reunión que se necesite para avanzar.

Debido a que no es un proyecto dentro de un equipo real, no se han llevado a cabo las reuniones diarias dentro de este proyecto.

3.5.4 Sprint Review y Retrospectiva del Sprint

Al acabar el Sprint, el equipo se reúne con el Product Owner y cualquier otro usuario que quiera asistir en el *Sprint review*, dónde presentan lo que se ha desarrollado. Esta reunión se presenta para determinar qué debería hacer a continuación el equipo. Entre la Sprint review

y el Sprint planning, el Scrum Master tiene una reunión de retrospectiva del Sprint con el equipo, Donde se revisa el proceso y las prácticas para hacer más efectivo el siguiente Sprint.

3.6 Gestión del proyecto

La gestión del proyecto tiene como objetivo cumplir con la planificación realizada y los costes, optimizar el uso de recursos, y hacer seguimiento del estado del proyecto.

3.6.1 Estimación

Para estimar las historias de usuario, se ha seguido de manera orientativa la equivalencia:

$$\textit{1 punto historia} \simeq \textit{1.5 horas de esfuerzo}$$

Como se ha determinado que los Sprints de este proyecto abarcan 40 puntos historia, y tienen una duración de 2 semanas, el tiempo ideal de trabajo sería de aproximadamente unas 4 horas al día. Debido a circunstancias personales no se han reflejado fielmente estas estimaciones, pero sí el número de Sprints y su duración.

3.6.2 Planificación

Se describen el número de Sprints:

1. Sprint 0: Preparación de datos
2. Sprint 1: Primeras recomendaciones
3. Sprint 2: Integración con Spotify
4. Sprint 3 : Visualización de recomendaciones avanzada
5. Sprint 4: Diversidad de recomendaciones
6. Sprint 5: Más tipos de recomendaciones y usabilidad
7. Sprint 6: Mejoras de usabilidad final

3.6.3 Recursos

Existen 3 recursos humanos, según explicado en la sección de roles 3.3 de la página 15:

- El equipo: David Touriño Calvo.
- Los directores: Javier Parapar López y Daniel Valcarce Silva.

Los recursos materiales son propiedad del alumno, las herramientas de soporte al desarrollo como GitLab o Taiga fueron proporcionadas por los directores.

3.6.4 Costes

Para los recursos humanos se contemplan los costes siguientes:

Recurso	Coste €/hora
Equipo de desarrollo	18,70
Directores	41,66

Cuadro 3.2: Estimación de costes para los recursos humanos del proyecto

Por lo tanto, teniendo en cuenta que:

- El proyecto tiene 7 sprints
- Los directores dedican 3 horas por sprint
- El equipo dedica 60 horas / sprint según lo comentado en 3.6.1

Recurso	Tiempo (h/sprint)	Número de sprints	Coste (€/h)	Total(€)
Equipo de desarrollo	60	7	18.70	7854,00
Directores	6	7	41.66	1749,72
			Total	9603.72

Cuadro 3.3: Desglose del coste de los recursos humanos

Se puede ver en el cuadro 3.3 que el coste total de los recursos humanos del proyecto serían 9603,72

Sobre los costes materiales, se pueden ver en el cuadro 3.4. No se han necesitado licencias ni equipos especiales. No se tiene en cuenta la electricidad gastada.

Recurso	Unidades	Coste	Vida útil	Tiempo de uso	Total(€)
Licencias	-	-	-	-	0
Estación de trabajo	1	1800	48	6	225
				Total	225

Cuadro 3.4: Desglose de los costes materiales

Además, como hemos desplegado en el cloud, habrá un gasto fijo mensual por la máquina que se está alquilando, que podemos calcular en la página de Azure [19].

Servicio	Unidades	Coste €/h	Total mensual(€)
Azure Kubernetes Service	2 VMs	0,117	170,82
Dirección IP	1	0	0
Almacenamiento	1	-	0,63
Container Registry	1	-	5
		Total	176,45

Cuadro 3.5: Coste mensual Azure

Por lo tanto, el coste total del proyecto es de 9828,72€ + 176,45 €/mes mientras estuviese la aplicación en funcionamiento. Este es el precio de coste, sin contemplar ningún margen de beneficio. Si el coste mensual fuese demasiado elevado, podríamos utilizar una máquina más pequeña. Aquí tenemos una comparativa entre la máquina elegida y una más económica:

Cantidad	Modelo	vCPUs	RAM	Costo €/hora	Total mensual(€)
2	D2v3	2	8	0,117	170,82
2	B2s	2	4	0,050	72,42

Cuadro 3.6: Comparativa entre dos máquinas de Azure

Sin embargo en este proyecto la RAM es muy necesaria, y puede que la B2s no sea suficiente para arrancar la aplicación y atender las peticiones.

Capítulo 4

Tecnologías

Es importante escoger un buen conjunto de herramientas para completar con éxito el desarrollo del proyecto. En este capítulo se hablarán de las tecnologías más importantes utilizadas durante el desarrollo de este proyecto. Se han agrupado en los apartados lenguajes, librerías, frameworks, y herramientas.

4.1 Lenguajes

Los diferentes lenguajes nos permitirán definir instrucciones, formatos, o consultas que son la base del proyecto.

4.1.1 Java

Java es un lenguaje de programación orientado a objetos. Para su uso requiere una máquina virtual de Java (JVM). El hecho de que funcione a través de la Java Virtual Machine permite que un programa escrito en JAVA pueda ser trasladado a diversos entornos, añadiendo un nivel de abstracción entre el sistema operativo y el programa.

Java es un lenguaje fuertemente tipado y es tanto interpretado como compilado, ya que el compilador de java genera bytecode que es interpretado por la máquina virtual. La JVM posee un recolector de basura o *Garbage Collector* que se encarga de liberar la memoria de los objetos creados cuando éstos ya no son referenciados.

Su tipado estricto y la administración automática de la memoria a través del recolector de basura hace que sea un lenguaje robusto, que anticipa errores en tiempo de compilación y elimina la responsabilidad al programador de liberar la memoria.

Su uso en el proyecto fue servir de base para el backend de la aplicación web, permitiendo que sea fácilmente escalable e instalada en cualquier entorno que soporte Java 8 Para este proyecto se ha empleado JAVA 11 Long Time Support (LTS) que gracias al uso de *lambdas* y

el *api de streams* ha facilitado el desarrollo a la hora de crear hilos y convertir datos desde el api de Spotify a la pantalla de la aplicación web.

4.1.2 Python

Python [20] es un lenguaje de programación multiparadigma. Es un lenguaje de propósito general. Soporta tanto orientación a objetos, como procedural, como funcional. Python es interpretado y utiliza tipado dinámico, lo que otorga al desarrollador más libertad a la hora de operar y reutilizar referencias. Posee también un recolector de basura que liberará la memoria de los objetos que ya no son referenciados.

El lenguaje está diseñado para ser legible y reutilizable, debido a su claridad en la sintaxis el código suele ser más sencillo y corto que los programas en otros lenguajes como JAVA o C.

El intérprete de Python permite la portabilidad de los programas, pudiendo ejecutar el mismo código en otro sistema que pueda ejecutar el intérprete.

Su uso en el proyecto fue ser el núcleo del sistema de recomendaciones, debido a su facilidad de uso para el tratamiento masivo de datos al utilizar librerías como Pandas o Numpy.

4.1.3 HTML

HyperText Markup Language (HTML) [21] es un lenguaje de marcas para la elaboración de páginas web. El lenguaje permite definir el contenido de una página web utilizando una serie de etiquetas que diferencian los elementos como: cabeceras, pie de página, títulos, artículos... Su uso en el proyecto fue servir como base a la hora de construir las plantillas de la aplicación web para elaborar el frontend.

4.1.4 CSS

Cascading Style Sheets (CSS) es un lenguaje de estilos que permite describir la presentación de un documento en un lenguaje de marcas como HTML. CSS es un estándar del World Wide Web Consortium (W3C) (W3C).

Se ha utilizado junto con *Bootstrap* sobre las plantillas HTML para definir el estilo de la aplicación web y personalizar la interfaz de usuario. En concreto se ha utilizado CSS3 [22].

4.1.5 JavaScript

JavaScript [23] es un lenguaje interpretado, no tipado, que permite programar las interacciones entre los elementos de la vista del usuario.

En este proyecto se ha utilizado para hacer peticiones Asynchronous JavaScript And XML (Ajax), crear enlaces dinámicamente, y reordenar elementos según las acciones del usuario.

4.1.6 SQL

Structured Query Language (SQL) es un lenguaje de consultas que permite obtener datos de una base de datos relacional.

En este proyecto se ha utilizado para manejar la información de los usuarios, y obtener datos de las listas de reproducción y canciones del propio motor de recomendaciones.

4.2 Librerías

Se han utilizado librerías de JAVA, Python y JavaScript. Llamamos librerías a las herramientas que, dentro de una tecnología concreta, contienen utilidades reutilizables, pero no llegan a definir un estilo de desarrollo.

4.2.1 Spotify Web API Java

Librería de código abierto en JAVA no soportada oficialmente por Spotify. Es un conector entre el API REST de Spotify en código JAVA. Se decidió utilizarla para simplificar el desarrollo de un cliente ya que Spotify no ofrece un documento de definición del API a partir del cual generar un cliente. Utiliza el patrón Builder de Effective Java [24] Para construir las llamadas al API.

A pesar de que no está soportada oficialmente, Spotify recomienda su uso y el repositorio está actualmente activo.

4.2.2 Thymeleaf

Thymeleaf es una librería y motor de plantillas para JAVA. Permite definir una serie de plantillas en HTML, y, utilizando un namespace propio definir una serie de operaciones con las que formar un código HTML en caliente, que será el que se envíe al cliente.

Se ha utilizado sobre todo a la hora de mostrar la información en distintos elementos HTML, iterando desde una lista en JAVA para formar una tabla automáticamente, o personalizar ciertos *scripts* que dependían del resultado del backend.

El principal motivo para la elección de este motor por encima de otros es la facilidad de uso y su integración con Spring.

4.2.3 PySparNN

Desarrollada por Facebook Research [25], esta librería en Python es el núcleo del sistema de recomendaciones. Permite buscar vecinos para un vector utilizando Approximate Neareast Neighbor (a-NN), tras definir una matriz dispersa.

Se utiliza para obtener vecinos de una lista de reproducción, utilizando una matriz dispersa de N (listas de reproducción) X M (canciones).

4.2.4 Pandas

Este conjunto de librerías para Python permiten manejar grandes volúmenes de datos fácilmente. Originalmente nació como extensión de numpy. Es software libre, desarrollado por la comunidad.

Utilizando *Dataframes* se ha exportado la información del conjunto de datos inicialmente en 1000 archivos en formato JSON a la base de datos no relacional Redis y a la relacional MySQL. Para ello se han realizado una serie de *scripts* utilizando *jupyter-notebook* en los que se procesa el dato por fases para no saturar la memoria.

4.2.5 Otras librerías

Numpy

Numpy [26] es una librería para Python desarrollada principalmente para ofrecer computación científica. Provee la capacidad de manejar vectores de N dimensiones y operar con ellos. En este proyecto se ha utilizado para operar con los vectores que representan listas de reproducción, para cualquier tipo de cálculo. Así como para crear la matriz dispersa que se utiliza como base para la construcción del árbol de búsqueda con *Pysparnn*.

Mapstruct

Mapstruct es una librería en JAVA utilizada para el mapeo de entidades de diferentes dominios. A través de una abstracción definida por el desarrollador, genera el código con la implementación.

Se ha utilizado para hacer las transformaciones entre las entidades de Spotify y las entidades propias que se mostraban por pantalla.

También para hacer las transformaciones entre el modelo de base de datos y la vista.

JQuery

JQuery es una librería de JavaScript que nos permite de manera sencilla interactuar con los objetos existentes en la página, definir manejadores para determinados eventos y modificar el aspecto de la página. Se ha utilizado para identificar las interacciones del usuario con la página y realizar las llamadas al backend necesarias.

redis-py

Es un conector de la base de datos no relacional Redis con Python.

Se utiliza constantemente en el motor de recomendaciones para hacer las conversiones de tracks de Spotify a los números que entiende nuestro modelo.

4.3 Frameworks

Un framework es una herramienta o conjunto de herramientas que nos permite establecer una metodología de desarrollo y simplificar tareas comunes tediosas y repetitivas. Se han utilizado diferentes frameworks para los diferentes lenguajes y módulos del proyecto.

4.3.1 Swagger

Swagger es un conjunto de herramientas que permiten definir un API de servicios Representational state transfer (REST) que se puede explotar, utilizando sus propias herramientas, de diversas formas y en diferentes entornos.

Se basa en crear una especificación a partir de la cual podemos:

- Crear una plantilla para el servidor en diferentes lenguajes y tecnologías.
- Generar un cliente en diferentes lenguajes y tecnologías
- Crear una documentación (Swagger-UI) con la definición de todos los servicios y una forma de invocarlos.

En este proyecto se ha utilizado para realizar la comunicación entre el motor de recomendaciones en Python y el backend de la aplicación web.

Para ello se ha definido un API utilizando Swagger 2.0, un formato de descripción de APIs REST, y con ello se ha generado una plantilla de servidor en Python utilizando Flask. En el lado del cliente se ha configurado un *plugin* de *maven* para generar el cliente en compilación, y que añada directamente el componente en tiempo de ejecución.

4.3.2 Flask

Flask es un framework de desarrollo que permite definir servicios REST en Python de manera sencilla. Es un framework ligero con pocas características, pero admite extensiones que pueden añadirlas.

Se ha utilizado para exponer los servicios REST del motor de recomendaciones. Para ello se ha generado un stub a partir de la definición de API creada en Swagger 2.0

4.3.3 Spring boot

Spring boot [27] es un framework de desarrollo en JAVA que permite crear una aplicación web que se puede arrancar por sí sola. Proporciona las herramientas y librerías necesarias a la hora de crear una aplicación web. Es autoconfigurable, permitiendo cambiar la mayor parte de la configuración mediante archivos de propiedades (en formato YAML o PROPERTIES) o en anotaciones.

El objetivo de Spring es facilitar el desarrollo de las aplicaciones Java EE, promoviendo buenas practicas de diseño y programación.

Es un proyecto modular. Spring boot proporciona varios *starters* que automáticamente configuran la aplicación para utilizar las diferentes tecnologías que incorpora.

Dentro del proyecto, se ha utilizado Spring boot para:

- Administrar la inyección de dependencias: Es el núcleo de Spring, permite construir los objetos de manera automática.
- Configurar la programación orientada a aspectos: Utilizando Spring-AOP
- Gestionar la transaccionalidad.
- Anotar los endpoints y exponer los servicios.
- Acceso a la base de datos mediante Java Database Connectivity (JDBC).

4.3.4 Bootstrap

Boostrap es un framework de estilos, que permite crear y configurar una interfaz responsive de manera sencilla, utilizando clases definidas en las hojas de estilos del framework.

La aplicación está hecha utilizando Boostrap 4.1, la mayor parte del estilo son componentes de bootstrap.

4.3.5 JUnit

JUnit es un framework de desarrollo de pruebas unitarias para aplicaciones JAVA. Permite definir una serie de pruebas que se integran con *Maven* y *Spring* para configurar y ejecutar las pruebas de la aplicación. Junto con JUnit se ha utilizado Mockito para la elaboración de los pruebas unitarias.

En el proyecto se han creado varias pruebas unitarias con JUnit para cada componente del código, alcanzando una cobertura del 90%, referenciado en la figura 4.1 de la página 32

4.4 Herramientas

En esta sección se explicarán las herramientas que se han utilizado como soporte al desarrollo del proyecto.

4.4.1 Taiga

Taiga [28] es una plataforma de desarrollo de proyectos ágiles de código abierto. Soporta tanto Scrum como Kanban, permitiendo crear un backlog de historias de usuario que se pueden mover después a distintos Sprints.

4.4.2 Intellij Idea

Intellij Idea es un Integrated Development Environment (IDE) desarrollado por *Jetbrains* [29] para desarrollar código en JAVA. Como IDE, ofrece, entre otras, las siguientes características:

- Editor de código adaptado para JAVA con compilador integrado en tiempo real.
- *Debugging*: Proporciona la capacidad de depurar la aplicación ejecutando paso a paso las instrucciones.
- Refactorización: Detecta malos hábitos de programación y sugiere el cambio. Pudiendo detectar errores antes de ejecutarlo.
- Generación de código: El IDE provee atajos para generar código repetitivo sencillo (toString, hashCode, equals...)

El uso de un IDE facilita el trabajo y la eficiencia del mismo por éstas y otras características.

Se ha utilizado para desarrollar la aplicación web que se expone al usuario.

4.4.3 Pycharm

Pycharm es un IDE desarrollado por *Jetbrains* [29] para desarrollar código en Python. Como IDE, ofrece características similares al Intellij IDEA, entre ellas:

- Editor de código adaptado a Python con intérprete integrado en tiempo real.
- *Debugging*: Capacidad de depurar la aplicación en Python ejecutando las instrucciones paso a paso.
- Generación de código trivial.
- Refactorización de código.

Se ha utilizado para desarrollar el motor de recomendaciones.

4.4.4 Git y GitLab

El control de versiones es un requisito fundamental en cualquier proyecto de desarrollo software, utilizar un Version Control System (VCS) nos proporciona entre otras las siguientes ventajas:

- Controlar los cambios
- Todos las versiones están etiquetadas
- Deshacer errores con facilidad
- Se puede integrar con otras herramientas de Continuous Integration (CI)
- Promueve trabajar en equipo sobre un mismo proyecto sin interferir los desarrolladores entre sí.

Git es un software de control de versiones de código abierto diseñado para manejar proyectos tanto pequeños como grandes con eficiencia. Es el gestor de versiones más empleado actualmente. Su gran comunidad hace que haya varios proyectos que permiten la integración de Git con otras herramientas. Permite trabajar sin conexión a Internet, ya que se basa en la creación de un repositorio local el cual opcionalmente se sincroniza con un repositorio remoto. También permite la creación de ramas (*branches*) y moverse entre ellas.

En este proyecto se ha utilizado git para el control de versiones y GitLab como plataforma remota.

GitFlow

En un proyecto de desarrollo con varios participantes el repositorio git puede ser un caos. Para evitarlo, nace GitFlow.

GitFlow es un flujo de trabajo diseñado por Vincent Driessen en 2010. Su objetivo es organizar el trabajo de los contribuyentes al repositorio mediante el uso de *branches*. Existen 5 tipos de *branches*:

- *develop*: Rama principal de desarrollo. A partir de la cual nacen las *feature branch*
- feature branch: Se crea una rama por cada característica o funcionalidad que se vaya a incluir en la aplicación a partir de la *develop*.
- *release branch*: Una vez que están todas las características implementadas se crea una rama *release* a partir de la *develop* y se le hacen pruebas en un entorno de apropiado.
- *master*: Una vez que la aplicación ya está probada y corregida se puede integrar con la rama *master* para llevar a producción.

- *hotfix branch* Si se detecta algún fallo en producción, se crea una rama *hotfix* a partir de la *master* y luego se integra tanto en *master* como en *develop*

4.4.5 Maven

Maven es una herramienta desarrollada por Apache Software Fundation [30] para la construcción y gestión de proyectos JAVA.

Con maven, definiendo un Project Object Model (POM) en formato XML podemos:

- Automatizar la descarga e instalación de las dependencias, que se encuentran en un repositorio central (Artifactory) configurable también para el usuario.
- Manejar el uso de distintos perfiles/configuraciones.
- Declarar los módulos del proyecto y las dependencias entre los mismos, así como organizar el orden de construcción de los mismos.
- Administrar las fases de construcción de un proyecto.

La aplicación web JAVA utiliza maven para construir el proyecto y administrar las dependencias.

4.4.6 SonarQube

SonarQube es una plataforma de software libre de evaluación de código fuente automatizada. SonarQube es capaz de analizar:

- Cobertura de código, apoyándose en otros plugins como Jacoco para maven
- Código duplicado
- Vulnerabilidades
- Errores
- *Code Smell*

Se ha utilizado SonarQube para el análisis de código de la aplicación. Se han pasado los resultados y se ha alcanzado un 90% de cobertura:

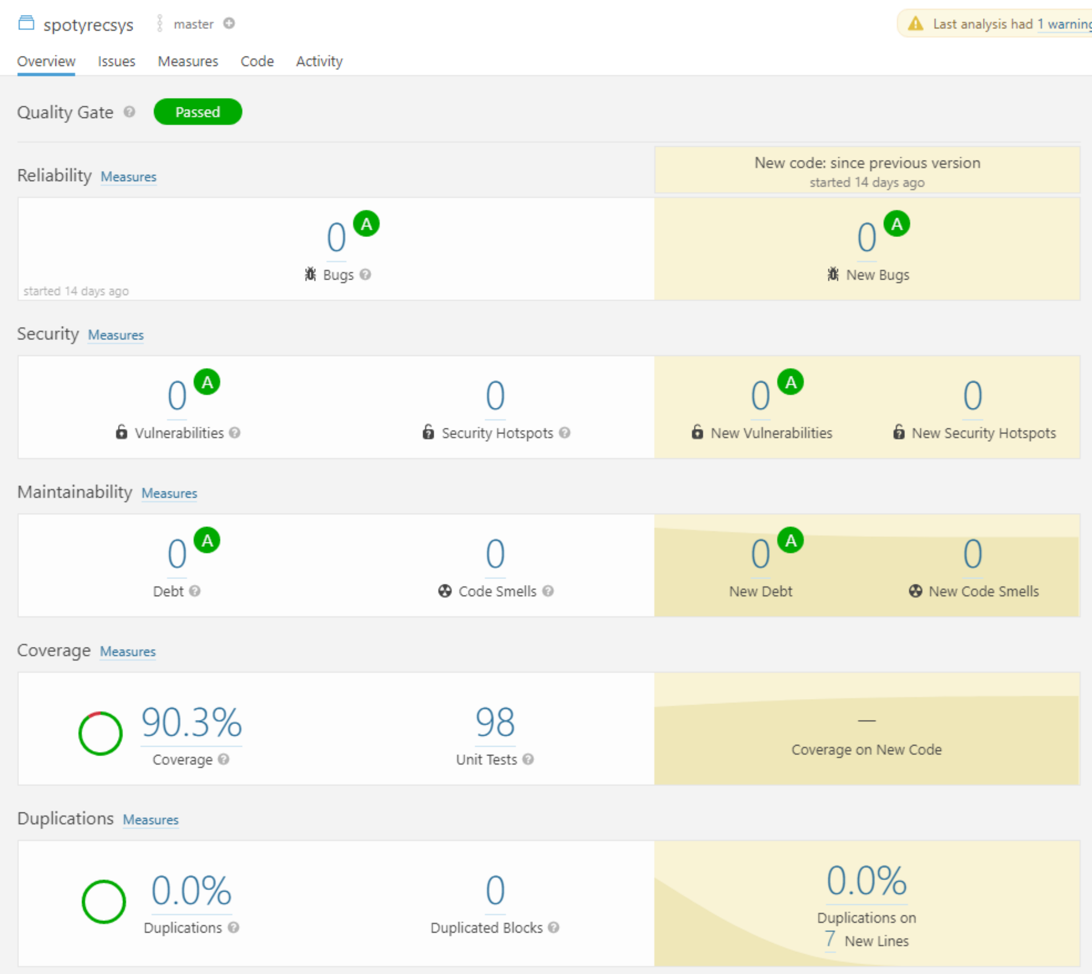


Figura 4.1: Resultado del análisis de SonarQube

4.4.7 Docker

Docker es una plataforma de desarrollo basada en contenedores.[31] Permite configurar y desplegar las aplicaciones en contenedores individuales, con todas las dependencias necesarias ya configuradas.

Se ha utilizado *Compose* [32], una herramienta que complementa a Docker para organizar todos los contenedores que requiere la solución del proyecto y poder instalar el proyecto entero de forma sencilla en cualquier máquina con Docker instalado.

En concreto, se han utilizado como base las imágenes por defecto oficiales de *Docker Hub* [33], Openjdk, Redis y MySql. Para la imagen de Python, se ha utilizado una imagen especializada en servidores UWSGI y FLask.

4.4.8 Kubernetes

Kubernetes es un orquestador de contenedores de aplicaciones. Permite desplegar varios contenedores en un clúster y maneja la infraestructura. La herramienta *kubectl* permite administrar un clúster remoto.

Se ha utilizado Kubernetes para el despliegue en producción, se han definido unas plantillas para el despliegue basándose en las imágenes de Docker del desarrollo. En concreto, se ha utilizado Microsoft Azure Kubernetes Service (AKS) para la creación y gestión de Kubernetes.

4.4.9 Microsoft Azure

Microsoft Azure es un servicio que ofrece infraestructura y plataforma en la nube para las aplicaciones. Nos permite alquilar su infraestructura. Se han utilizado varios servicios de Azure para desplegar en producción, entre ellos Azure Files, Azure Kubernetes Service, Azure Container Registry y Azure Public IP.

4.4.10 Redis

Redis es un motor de base de datos NoSQL en memoria. Sirve principalmente para almacenar pares clave valor. En la implementación de sistemas de recomendación eficientes, se ha demostrado [34] que el uso de almacenamiento caché de este tipo es fundamental en la eficiencia del funcionamiento. En este proyecto se ha utilizado para almacenar la relación entre:

- Las listas de reproducción y las canciones.
- Identificador de Spotify de canciones e identificador interno de canciones.

4.4.11 MySQL

MySQL es un sistema gestor de base de datos relacional de código abierto. Se ha utilizado para persistir la información de las listas de reproducción y canciones del conjunto inicial de datos de Spotify y para poder hacer búsquedas por nombre.

Capítulo 5
Desarrollo

En este capítulo de la memoria se procede a hablar del análisis de requisitos, diseño, implementación y pruebas a lo largo del proyecto con Scrum.

5.1 Requisitos

El análisis de requisitos es una tarea fundamental en el desarrollo de un proyecto software. En esta sección se mostrarán los requisitos funcionales y no funcionales elaborados por el alumno y los tutores durante el desarrollo del proyecto.

5.1.1 Requisitos funcionales

Entendemos requisito funcional como la descripción del comportamiento deseado entre entradas y salidas de un sistema o un componente. En este proyecto los requisitos funcionales se expondrán como historias de usuario, definidas en la figura 3.1 de la página 16.

5.1.2 Requisitos no funcionales

Llamamos requisitos no funcionales a los requisitos que especifican criterios para medir la operativa de un sistema, en lugar de especificar su comportamiento.

Se especifican los siguientes requisitos no funcionales dentro de este proyecto:

- **Rendimiento** La navegación a través de la página tiene que ser fluida. Por lo tanto la carga de una página no debe tardar más de 5 segundos en momentos de carga y normalmente entre 1 y 2 segundos.

- **Usabilidad** Los usuarios que vayan a utilizar la aplicación pueden tener todo tipo de experiencia informática por lo que la interfaz debe ser sencilla y amigable. Utilizando a poder ser elementos que recuerden a cómo se administran listas de reproducción en la propia aplicación de Spotify, con la que los usuarios sí estarán más familiarizados.

- **Seguridad** Es importante cifrar las conexiones para mantener la privacidad de los usuarios. Así mismo la información de las cuentas y acceso debe almacenarse de forma segura.
- **Robustez** La aplicación deberá tener en cuenta los posibles errores tanto humanos como del propio sistema. Es importante mantener la integridad de la cuenta del usuario y evitar que se borre por accidente cualquier tipo de información del usuario, ya sean listas de reproducción o canciones.

5.2 Arquitectura de la aplicación

En la figura 5.1 de la página 37 se puede ver el diagrama de arquitectura de la aplicación. Para diseñar esta arquitectura, hemos separado los componentes más distinguibles según la carga de la funcionalidad y el dominio de los datos de salida. Así, distinguimos las partes que se detallan a continuación.

5.2.1 Servidor Web

Es la aplicación web que se expone al cliente a través de las interfaces de usuario en HTML5, CSS3 y JavaScript, explicados en los apartados 4.1.3, 4.1.4 y 4.1.5 de la página 24. Este módulo implementa la lógica de los servicios que no son recomendaciones y sirve como conector entre el servicio de recomendaciones y el usuario. Así mismo, también se realizan las peticiones a la API de Spotify desde este módulo para recuperar la información que se muestra al usuario sobre su cuenta y sus listas de reproducción. Se conecta a la base de datos MySQL mediante Spring JDBC.

5.2.2 Servicio REST de recomendaciones

Es un servicio interno que se expone al servicio web a través de una Interfaz REST. Su interfaz se define utilizando un contrato de Swagger 2.0, formato descrito en el apartado 4.3.1 de la página 27.

Su objetivo es realizar la lógica de obtención de las recomendaciones sin implicarse en la conectividad con Spotify, simplificando el desarrollo. Está escrito en Python y las entradas serán los identificadores de canciones de Spotify, mientras que las salidas serán los identificadores internos de listas de reproducción o canciones.

Implementación de la arquitectura

Para llevar a cabo la implementación de esta arquitectura se han creado dos artefactos. Un artefacto JAVA, basado en Spring boot, y un artefacto Python, basado en Flask.

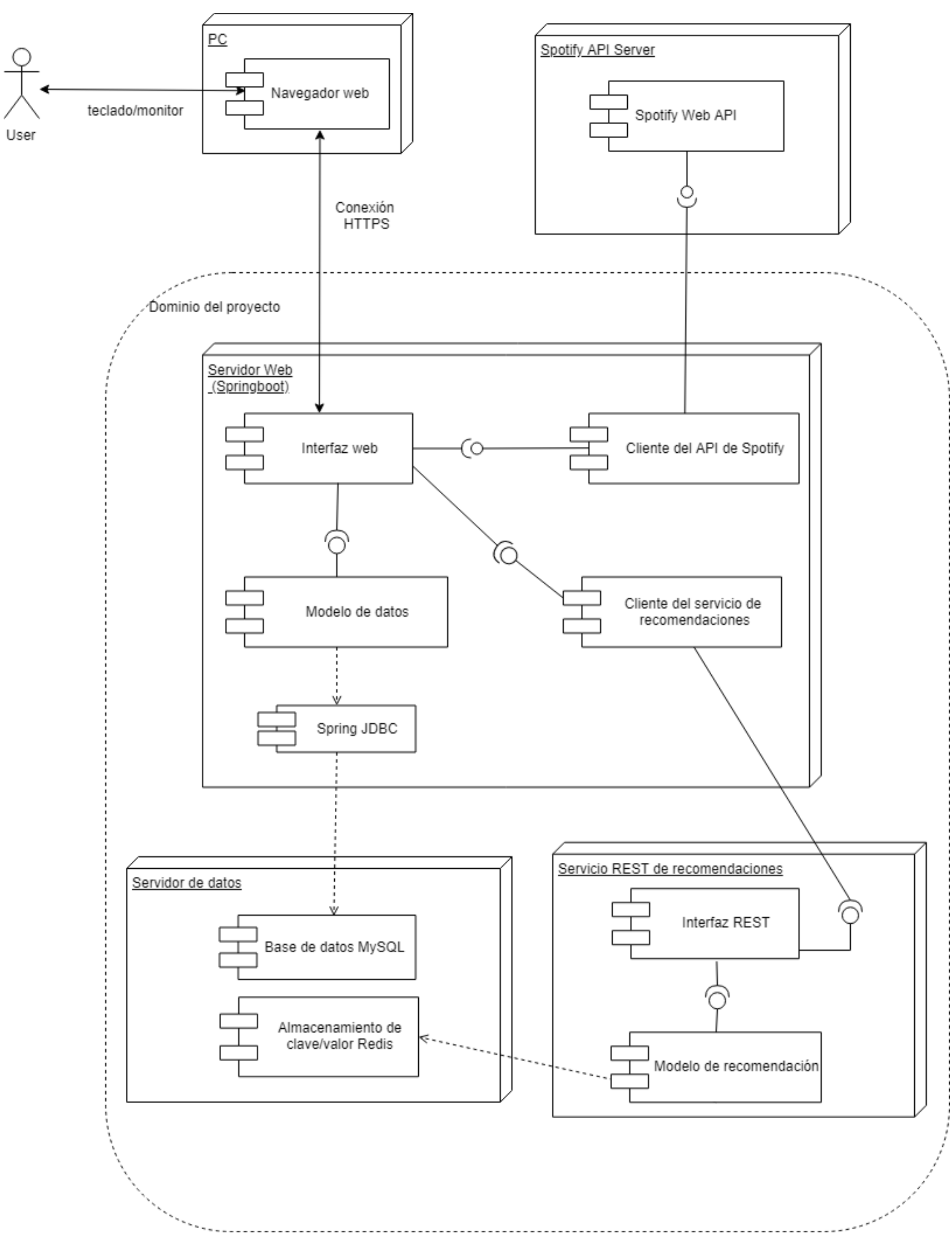


Figura 5.1: Diagrama de arquitectura de la aplicación

El artefacto JAVA se configura con Thymeleaf, para ofrecer la interfaz web, y Spring JDBC para acceder a los datos. Para la comunicación con el servicio de recomendaciones se crea un cliente REST dinámicamente y una fachada expone los servicios sin detallar la implementación. Para la comunicación con el API de Spotify, se delega en la librería detallada en el apartado 4.2.1.

El artefacto Python se configura con Flask. Se crea el índice de búsqueda detallado como se detalla en 2.4.1 en la página 9 y se vuelca a un fichero para su posterior carga rápida. Para crear el índice de búsqueda se ha utilizado la librería de PySparNN. Para construir la matriz dispersa a partir de la cual crear el índice se han elaborado una serie de *scripts* utilizando *Pandas* con los que se explota el conjunto de datos de *Spotify*

Se instalan los gestores de base de datos y se configuran los conectores en los dos artefactos.

5.3 Modelo de datos

Se proponen 2 motores de almacenamiento de datos que se explican a continuación. La razón de elegir 2 motores es motivada por temas de rendimiento y memoria. Mientras que Redis es muy rápido, tratar todo el dato en memoria dispara el consumo de RAM, por lo que las operaciones del motor de recomendaciones utilizarán Redis, mientras que para mostrar la información de la lista de reproducción o canción se usará MySQL.

5.3.1 MySQL

MySQL es un gestor de base de datos relacional explicado en el apartado 4.4.11 de la pagina 33. Se crea el siguiente modelo para el gestor:

account	
PK	id
	created
U1	email
	access_token
	refresh_token
	role

playlists	
PK	pid
I1	name
	description
	num_tracks
	num_albums
	num_followers
	num_edits
	num_artists
	duration_ms
	collaborative
	image1
	image2
	image3
	image4

songs	
PK	sid
U1	track_uri
I1	track_name
	album_name
	album_uri
	duration_ms

Figura 5.2: Modelo de datos en MySQL

Para optimizar el rendimiento del servicio, no se presenta de forma relacional. Ya que no se realizarán escrituras. El servicio de recomendaciones devolverá el id de una lista de reproducción o de una canción, de forma que las búsquedas serán a través de clave primaria.

5.3.2 Redis

Redis es un gestor de base de datos relacional explicado en el apartado 4.4.10 de la página 33. En él, definimos unos conjuntos de pares clave/valor que se almacenarán en memoria y se consultarán en tiempo de ejecución. Definimos 2 instancias de Redis, representadas gráficamente en la figura 5.3:

- La primera instancia tiene la relación entre los identificadores internos de las canciones y el identificador que otorga Spotify. Para ello se utiliza el hash de Redis *track_to_id*. Esta relación al principio se redundó de forma que se pueda consultar bidireccionalmente, utilizando el hash de Redis *id_to_track*. Sin embargo, por cómo se organizó el modelo en MySQL se comprobó que no era necesario.
- La segunda instancia es la relación entre una lista de reproducción y un conjunto de los diferentes ids de canciones.

Estas operaciones se ejecutan con complejidad O(1).

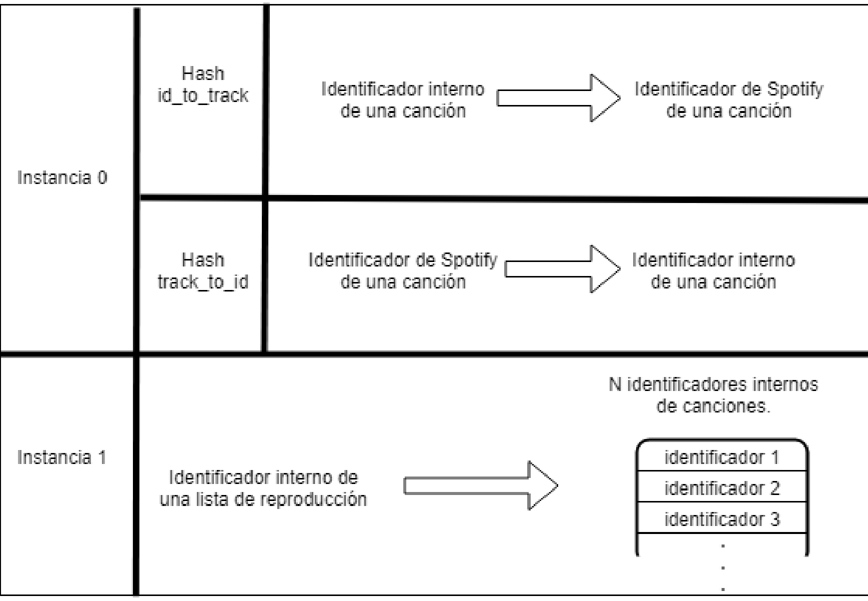


Figura 5.3: Distribución de los datos en Redis

Estos datos se explotarán solamente desde el motor de recomendaciones escrito en Python, de la siguiente forma:

1. Al comenzar un servicio se traducen los identificadores de Spotify a los identificadores internos utilizando el hash de Redis *track_to_id*.

2. Para las recomendaciones de canciones al aplicar Weighted Sum Recommender (WSR), definido en el apartado 2.4.2 de la página 10, se obtienen las canciones de los vecinos obtenidos, utilizando para ello el *hash set* de Redis de la instancia 1.

3. Los identificadores de canciones o listas de reproducción se devuelven en crudo para que el servicio web acceda a los datos en MySQL a través de la clave primaria. En un principio la intención del servicio es que las entradas y salidas fuesen los identificadores externos, para mayor usabilidad ante futuras evoluciones, pero se comprobó que añadía pasos innecesarios que penalizaban el rendimiento.

5.4 Obtención de recomendaciones

A continuación se explicará el proceso general y más completo para obtener recomendaciones en esta aplicación. Se han definido varias casuísticas de recomendaciones, pero todas están relacionadas y contienen código reutilizable constantemente.

A la hora de obtener recomendaciones, podemos clasificarlo como un proceso por fases. A continuación se describen las fases para " (26) Obtener recomendaciones de canciones dadas las listas de reproducción del usuario", ya que es el caso más completo que utiliza todas las fases. Otro tipo de casuísticas se saltarán algunas fases.

5.4.1 Transformación de una lista

La entrada inicial del servicio es una lista de identificadores de canciones de Spotify. Para poder explotarlos se convierten primero a la lista de identificadores de canciones utilizando Redis. Esta conversión es posible gracias al modelo de datos explicado en la sección 5.3.2 de la página 39. Este proceso al comienzo del desarrollo tuvo su inverso para obtener la información de una canción, pero evolucionó para utilizar el modelo en MySQL.

Después de esto, para que la entrada sea reconocida por la función de búsqueda, se convierte esta lista de identificadores en un vector de ceros disperso de un tamaño de hasta el número de canciones del conjunto de datos. En los índices correspondientes a los identificadores obtenidos, se escribe un uno. Y pasa a ser válido para la búsqueda de vecinos. Los identificadores internos que se utilizan para almacenar las canciones y listas representan los índices que utilizamos para escribir ceros o unos.

5.4.2 Lista de reproducción ”fusión”

Una de las funcionalidades de obtención de recomendaciones solicitada era a partir de la cuenta del usuario. Esto plantea un problema ya que aunque se trabaja con un modelo de búsqueda de vecinos, los puntos que se suelen representar como usuarios con gustos similares, en este caso son listas de reproducción, por lo que el concepto de usuario no está en nuestro modelo.

Para poder hacer esta funcionalidad, se crea una nueva lista de reproducción llamada ”lista fusión” que comprime todas las listas del usuario en una sola lista.

Para ello creamos entonces una fila de la Matriz, un vector disperso utilizando todas las listas del usuario y asignando valores más altos a aquellas canciones que se presenten varias veces entre las distintas listas de reproducción.

5.4.3 Búsqueda de vecinos

Para la búsqueda de vecinos se está utilizando la estructura de búsqueda de a-NN incorporada en la librería de PySparNN. El resultado de la salida de dicha estructura es una lista de tuplas, vecino/similitud.

Estrictamente hablando, la similitud sólo sería necesaria para los casos en los que hiciera falta aplicar WSR (sección 2.4.2, página 10). Sin embargo, la obtenemos en todos los casos y la estudiamos para hacer ajustes a los vecinos que encontramos.

Los ajustes que hacemos corresponden a:

- Si detectamos una nota muy negativa sobre la similitud, esto es, las listas de reproducción son muy diferentes entre sí, no devolvemos recomendaciones. Esto puede pasar si en la entrada vienen solamente canciones que no están dentro del conjunto de de datos inicial.

- Si detectamos una distancia coseno 0, ya que corresponde a la misma lista de reproducción que la que lista de entrada y se descarta, haciendo otra petición pidiendo más listas.

5.4.4 Aplicación de WSR

Creamos una función genérica para aplicar WSR en cualquiera de los casos de uso descritos. Aplicamos WSR cuando tenemos una lista de vecinos, sus similitudes, y una lista de reproducción original.

Lo primero que se hace es obtener las canciones que correspondan a las listas de reproducción y recopilarlas en un *DataFrame* que elimina las canciones que tuviera la lista de reproducción original.

Para cada columna, aplicamos el escalar como se señala en 2.3 en la página 11.

Recuperamos un número fijo de identificadores de canciones del resultado y finaliza este proceso.

5.4.5 Incrementos

Uno de los problemas al recomendar canciones es que no tendría sentido ante una lista de reproducción recomendar canciones que ya estuvieran dentro de la misma. Para evitarlo eliminamos las canciones duplicadas que salen de la búsqueda de vecinos.

El problema es que si una lista entrada es demasiado grande, o coincide que todos los vecinos que encuentra son subconjuntos de la primera lista, no obtenemos recomendaciones ya que todos los posibles candidatos se eliminan por estar repetidos.

Para solucionar este problema, planteamos una serie de incrementos. Al llamar a un servicio de recomendaciones de canciones, se llama hasta un máximo de N veces a la búsqueda de vecinos, las llamadas continuadas se detienen cuando se consiguen recomendaciones adecuadas, pero continúan mientras no haya encontrado canciones o se acabe el número máximo de incrementos.

Si tras el número máximo de incrementos aún no se encuentran canciones, se devuelve un mensaje informativo conforme no se ha podido encontrar ninguna recomendación.

5.5 Desarrollo

Se han elaborado un total de 7 Sprints con una duración de 2 semanas cada uno. Cada sprint incrementaba de forma evolutiva la funcionalidad y usabilidad de la aplicación. En esta sección se detallará cada sprint del desarrollo.

5.5.1 Sprint 0: Preparación de datos

Este Sprint inicial consistió en preparar el conjunto de datos inicial de Spotify a partir del cual se obtendrían las recomendaciones. Este Sprint no contenía ninguna funcionalidad para el usuario, y solo consistía en una preparación inicial para poder abordar las tareas de los Sprints futuros.

Para la realización de este proyecto partimos de un conjunto de datos o *dataset* liberado por Sporify para el RecSys Challenge 2018, el The Million Playlist Dataset (MPD). Este conjunto de datos consistía en un fichero zip con mil archivos en formato JSON, con el formato que se muestra en la figura 5.4. Cada uno de los archivos contenía mil listas de reproducción con información sobre las listas y las propias canciones. Para poder explotar estos datos, se elaboraron una serie de *scripts* utilizando Jupyter Notebook y un conjunto de librerías de *pandas*.

Para poder obtener todos los datos preparados de la forma adecuada, se han hecho distintas fases para ir almacenando de forma persistente los datos, así como generar un identificador numérico para las canciones.

Las listas de reproducción ya incorporaban un identificador numérico y como además cada una es única, no hizo falta hacer ninguna gestión especial. Se almacenaron recorriendo los ficheros y añadiéndolos a base de datos fila a fila en un *script*.

Almacenamiento de canciones y generación de identificador numérico.

Primero se han recorrido todos los ficheros uno a uno, extrayendo sólo el dato del identificador de la canción de Spotify, y se han ido añadiendo estos identificadores a un *Dataframe* en memoria, eliminando en cada iteración de cada fichero los identificadores repetidos.

Una vez que tenemos el *Dataframe* con los identificadores de Spotify, generamos un índice numérico, que se corresponderá al identificador único de cada canción.

Finalmente, recorremos el *Dataframe*, insertando cada fila en el almacén de datos clave/-valor de Redis, en el *hash track_to_id* e *id_to_track*

Almacenamiento en MySQL

Para almacenar los datos referentes a las canciones y las listas como descripciones, álbumes y demás, se ha seguido la siguiente aproximación:

Se recorren de nuevo todos los ficheros. Es necesario hacerlo dos veces porque sin una referencia de almacenamiento como la creada en el punto anterior habría que mantener todo el dato de un millón de listas de reproducción en memoria, y se llenaría la memoria.

Para cada fichero, para cada canción se va insertando en una tabla de canciones utilizando el nuevo id, obtenido de nuevo de Redis. Cuando el id ya existe, este se ignora. Esto lo conseguimos utilizando el modificador del *statement* de SQL ”IGNORE”.

Como el único propósito de las tablas *songs* y *playlists* son hacer lecturas de los datos, se ha escogido MyISAM como motor interno de la base de datos, que además permite añadir índices *FULL_TEXT*.

Para los casos de uso de buscar canciones y listas de reproducción, se han añadido dos índices *FULL_TEXT* sobre el campo *track_name* de la tabla *songs* y sobre el campo *name* de la tabla *playlists*.

Construcción de la instancia

Debido a la naturaleza del conjunto de datos, de M canciones para N listas de reproducción, si queremos construir una matriz de $N \times M$ que relacione las canciones con las listas de reproducción, veremos que la mayoría de datos de la matriz estará compuesta por 0s, ya que

```json
{
    "name": "musical",
    "collaborative": "false",
    "pid": 5,
    "modified_at": 1493424000,
    "num_albums": 7,
    "num_tracks": 12,
    "num_followers": 1,
    "num_edits": 2,
    "duration_ms": 2657366,
    "num_artists": 6,
    "tracks": [
        {
            "pos": 0,
            "artist_name": "Degiheugi",
            "track_uri": "spotify:track:7vqa3sDmtEaVJ2gcvxtRID",
            "artist_uri": "spotify:artist:3V2paBXEoZIAhfZRJmo2jL",
            "track_name": "Finalement",
            "album_uri": "spotify:album:2KrRMJ9z7Xjoz1Az4O6UML",
            "duration_ms": 166264,
            "album_name": "Dancing Chords and Fireflies"
        },
        {
            "pos": 1,
            "artist_name": "Degiheugi",
            "track_uri": "spotify:track:23EOmJivOZ88WJPUbIPjh6",
            "artist_uri": "spotify:artist:3V2paBXEoZIAhfZRJmo2jL",
            "track_name": "Betty",
            "album_uri": "spotify:album:3lUSlvjUoHNA8IkNTqURqd",
            "duration_ms": 235534,
            "album_name": "Endless Smile"
        },
        {
            "pos": 2,
            "artist_name": "Degiheugi",
            "track_uri": "spotify:track:1vaffTCJxkyqeJY7zF9a55",
            "artist_uri": "spotify:artist:3V2paBXEoZIAhfZRJmo2jL",
            "track_name": "Some Beat in My Head",
            "album_uri": "spotify:album:2KrRMJ9z7Xjoz1Az4O6UML",
            "duration_ms": 268050,
            "album_name": "Dancing Chords and Fireflies"
        },
        {
            "pos": 11,
            "artist_name": "Mo' Horizons",
            "track_uri": "spotify:track:7iwx00eBzeSSSy6xfESyWN",
            "artist_uri": "spotify:artist:3tuX54dqgS8LsGUvNzgrpP",
            "track_name": "Fever 99°",
            "album_uri": "spotify:album:2Fg1t2tyOSGWkVYHlFfXVf",
            "duration_ms": 364320,
            "album_name": "Come Touch The Sun"
        }
    ]
}
```


Figura 5.4: Ejemplo de una lista de reproducción del conjunto de datos inicial

en el conjunto de datos hay más de 2 millones de canciones, y en cada lista de reproducción no suele haber más de 50 canciones.

Para esto, utilizamos matrices dispersas. Esta matriz nos permite ahorrar rendimiento y memoria.

Para construir el árbol de búsqueda sobre donde buscar los ”aproximadamente vecinos más cercanos” utilizaremos la librería de PySparNN [25]. Esta librería sirve para encontrar vecinos construyendo el árbol de búsqueda descrito en el apartado 2.4.1 a partir de una matriz dispersa.

Para construir la matriz creamos un dataframe con los índices de las listas de reproducción para las filas y los índices de las canciones para las columnas, para cada fichero para cada lista de reproducción, se examinan las canciones y se añade la fila a la Matriz. Al acabar la matriz, se forma el árbol de búsqueda.

5.5.2 Sprint 1: Primeras recomendaciones

En este Sprint se comienza a definir la arquitectura del servicio, creando uno de los artefactos. Se crea un servicio REST que devuelve diversos tipos de recomendaciones.

(24) Obtener recomendaciones de listas de reproducción según otra lista de reproducción.

Como: Usuario registrado.

Quiero: Poder obtener recomendaciones de listas de reproducción según otra lista de reproducción de mi cuenta.

Para: Poder añadirlas a mi cuenta.

Para esta historia de usuario se ha realizado una primera aproximación creando un servicio REST a través del cual se pueden obtener las recomendaciones enviando los identificadores de Spotify que componen una lista de reproducción. Para calcular las recomendaciones, se siguen algunas de las fases explicadas en la sección 5.4 de la página 40. En concreto, las fases que se siguen para este caso sería la transformación de identificadores inicial (5.4.1, página 40) y la búsqueda de vecinos (2.4.1, página 8).

Tras obtener las recomendaciones, se transforman de nuevo los identificadores utilizando almacenes del modelo de datos de Redis. Esta transformación ha quedado en desuso por nuevas aproximaciones para aumentar la eficiencia en la aplicación web, con el fin de evitar redundancia.

En la figura 5.5 se puede ver la interfaz autogenerada a partir de la especificación del API Swagger de los servicios que se hicieron. A pesar de no ser una interfaz adecuada, permite a cualquier usuario obtener recomendaciones válidas. Se hizo de esta manera para poder

entregar un mínimo producto entregable en este sprint. Se ha repetido la estrategia, aunque con diferentes formatos, en las siguientes recomendaciones de este sprint. Este API, una vez integrada en la web, se ha modificado para ganar eficiencia y en lugar de devolver listas de listas de identificadores de Spotify, devuelve una serie de identificadores internos.

(12) Obtener recomendaciones de canciones para una listas de reproducción.

Como: Usuario registrado.

Quiero: Poder obtener recomendaciones de canciones para una listas de reproducción.

Para: Poder añadirlas a una lista de reproducción de mi cuenta.

Del mismo modo que en la historia anterior, se ha realizado una primera aproximación creando un servicio REST a través del cual se pueden obtener las recomendaciones enviando los identificadores de Spotify que componen una lista de reproducción. Para calcular las recomendaciones, en contraste con la anterior historia, se añade una de las fases explicadas en la sección 5.4 de la página 40. La fase nueva que se añade para este caso es la aplicación de WSR (5.4.4, página 41.).

(27) Obtener recomendaciones de listas de reproducción dadas las listas de reproducción del usuario.

Como: Usuario registrado.

Quiero: Poder obtener recomendaciones de listas de reproducción dadas todas mis listas.

Para: Poder añadirlas a mi cuenta.

Del mismo modo que en la historia anterior, se ha realizado una primera aproximación creando un servicio REST a través del cual se pueden obtener las recomendaciones enviando los identificadores de Spotify que componen una lista de reproducción. Para calcular las recomendaciones, se añade una de las fases explicadas en la sección 5.4 de la página 40. La fase nueva que se añade para este caso es el cálculo de una *lista de reproducción fusión*(5.4.2, página 41).

5.5.3 Sprint 2: Integración con Spotify

En este Sprint la arquitectura del sistema acaba de tomar forma. Se crea el artefacto en JAVA para poder consultar los servicios creados en el Sprint 1 de forma sencilla y amigable.

Spotify Web API

Para la comunicación con el servidor de Spotify para obtener datos de los usuarios, administrar su cuenta, y poder obtener datos de las canciones se ha utilizado el *wrapper* Spotify

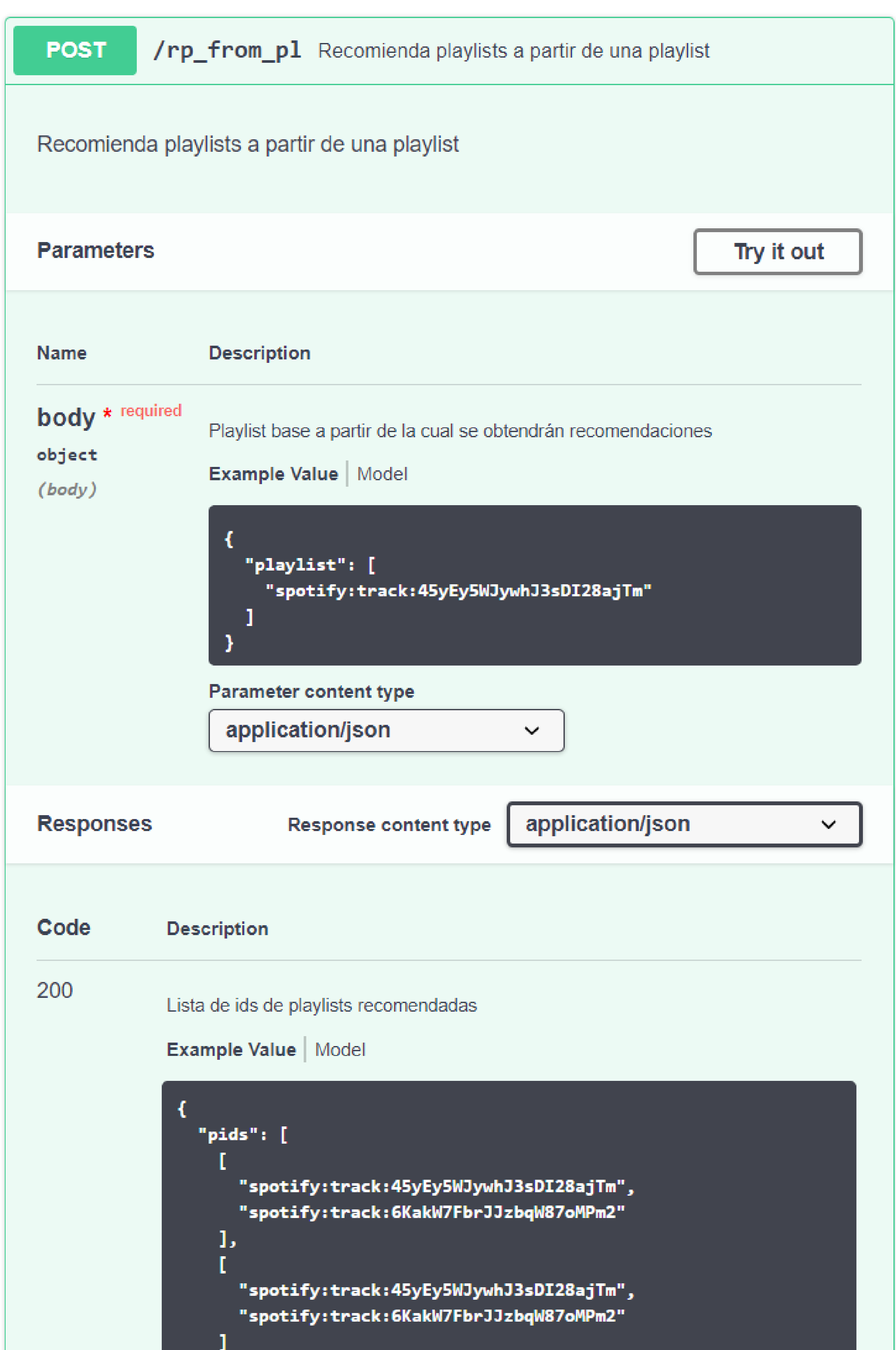


Figura 5.5: Interfaz generada para uno de los servicios REST

Web Api Java (4.2.1, página 25). Anunciado desde la guía de referencia de la página principal de Spotify, aunque la librería no está soportada de forma oficial.

Se ha implementado una fachada que expone los métodos específicos que se usan del API de Spotify, traduciendo las entidades de dominio de Spotify a las que se emplean dentro de la aplicación.

(4) Inicio de sesión con Spotify.

Como: Usuario.
Quiero: Poder conectarme a la aplicación con mi cuenta de Spotify.
Para: Poder utilizar la información de mi cuenta para calcular las recomendaciones.

Para realizar esta historia se ha tenido que hacer una integración con el servicio de Spotify explicado en la sección anterior. A mayores se ha tenido que implementar el protocolo OAuth como se detalla a continuación.

OAuth OAuth es un estándar para organizar flujos de autorización entre aplicaciones [35]. Permite que creemos una pseudo-autenticación a través de otra aplicación. En este proyecto se ha implementado el cliente de OAuth con Spotify. Para ello al autenticarse, nos llevará a la página de Spotify.

- En la dirección se le pasará un endpoint de *callback* que deberá estar dado de alta dentro de una aplicación de Spotify.
- Al introducir los credenciales, el servicio de OAuth de Spotify redirige al *callback* especificado pasándole un código de acceso.
- Con ese código de acceso, podemos solicitar a un servicio 2 tokens, el token de acceso y el de refresco.
- Cuando el token de acceso caduca, utilizamos el de refresco para pedir otro token.
- Persistimos los tokens en base de datos, para futuras referencias.

En la figura 5.6 se muestra la pasarela que vería el usuario al autenticarse.

Como en esta aplicación se realizan llamadas constantemente al Web Api de Spotify, el token de acceso se almacena en la sesión de usuario, para tenerlo disponible en cada llamada sin tener que penalizar la operación con acceso a disco. Para simplificar el desarrollo esta parte se ha desarrollado programando un *aspecto*.

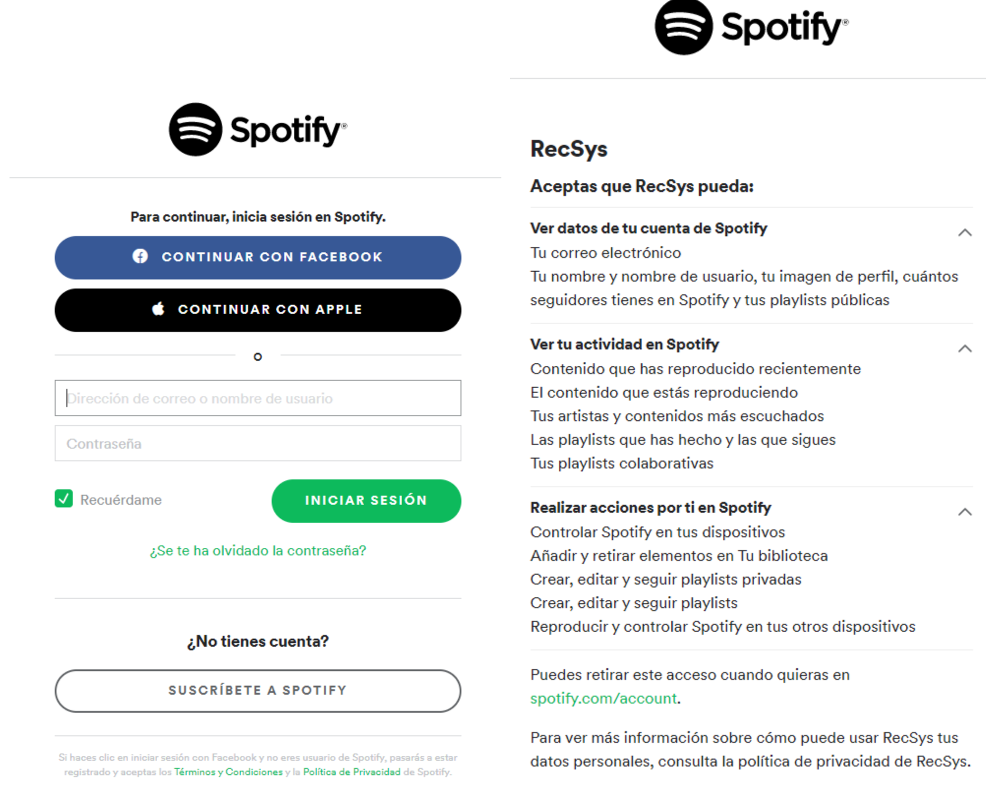


Figura 5.6: Al autenticarse en la aplicación, redirige a la pasarela de Spotify y se pide acceso al usuario a determinadas partes de su cuenta

Programación Orientada a Aspectos Se ha utilizado Aspect Oriented Programming (AOP) para implementar el refresco del token de acceso de OAuth.

La implementación se lleva a cabo mediante Spring AOP, incorporado en Spring Framework. El aspecto se ha definido mediante una serie de anotaciones en Java. Creamos una anotación específica para nuestro aspecto, que invocará a cualquier método que tenga esa anotación.

El Aspecto se ha definido para que actuase en cada operación de acceso al API de Spotify que usase un token de acceso. El aspecto llama al servicio del API que se haya solicitado. Si el token ha caducado, captura la excepción, pide otro token de acceso mediante el token de refresco, y vuelve a llamar al servicio una vez más.

Se ha decidido implementar un aspecto ya que era una tarea tediosa y repetitiva que ocurría siempre alrededor de una llamada. Para poder utilizar este aspecto es necesario que el primer parámetro del método donde se aplique sea un token de acceso.

Mostrar las listas de reproducción del usuario.

Afecta: Usuario autenticado.

Objetivo: Ver las listas de reproducción de mi cuenta de Spotify.

Razón: Acceder a cada lista y consultar las recomendaciones de cada una

Para poder utilizar las funcionalidades se necesita un interfaz con el cual acceder a las recomendaciones. Se crea una nueva pantalla donde el usuario puede visualizar sus listas tras haberse autenticado. Se realiza una comunicación con el API de Spotify a través de una fachada y se muestra la información de las listas en la pantalla. En la pantalla, como se puede ver en la figura 5.7, se muestran las listas de reproducción acompañadas de una imagen que puede ser propia de la lista o de las canciones que componen. Esta imagen la provee el servicio de Spotify. Para las carátulas de las listas recomendadas, hubo que hacer varios ajustes que se explicarán más adelante.

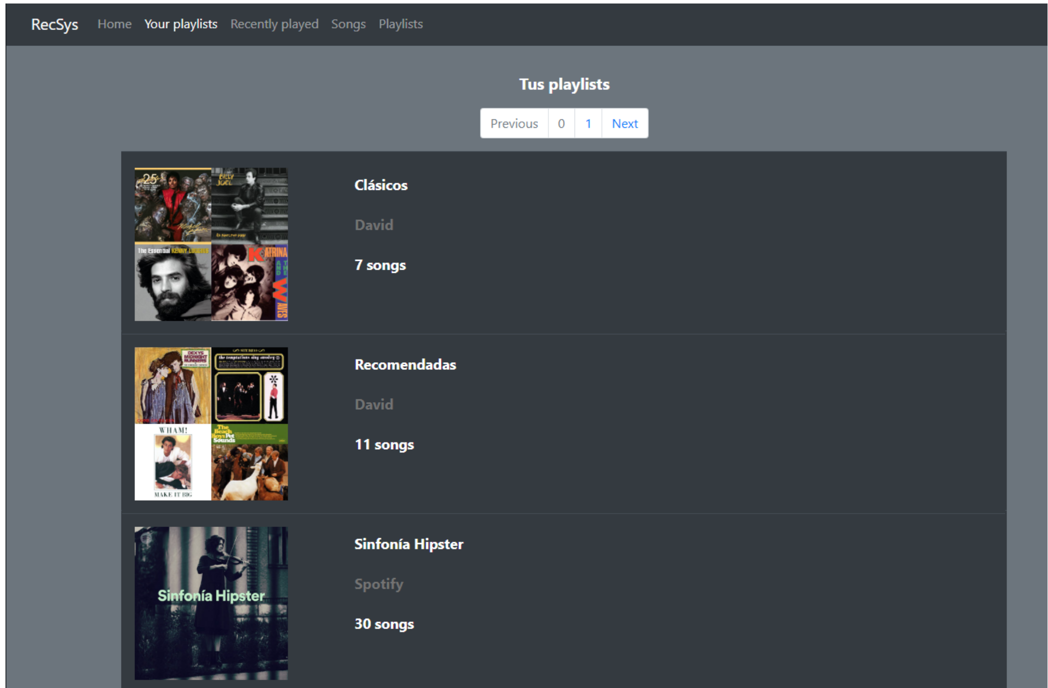


Figura 5.7: Listas de reproducción del usuario

5.5.4 Sprint 3 : Visualización de recomendaciones avanzada

En este Sprint se realiza la integración entre la aplicación web JAVA creada en el Sprint 2 y el servicio de recomendaciones REST creado en el Sprint 1.

(38) Mostrar recomendaciones de listas de reproducción según otra lista de reproducción.

Como: Usuario autenticado.

Quiero: Poder ver las recomendaciones de listas de reproducción al visualizar otra lista de reproducción.

Para: Descubrir música de manera amigable.

Se añade una pantalla para poder ver el detalle de una lista de reproducción y visualizar las recomendaciones, pudiendo acceder a ellas de forma amigable, se puede ver en la figura 5.8. Dentro de la pantalla tenemos una sección para la información general de la lista de reproducción, con su nombre, imagen, propietario y número de canciones. Por otro lado una tabla con la información de las canciones que componen la lista de reproducción desde donde podremos acceder al detalle de la canción y nos ofrecerá recomendaciones de canciones basándonos en esa canción.

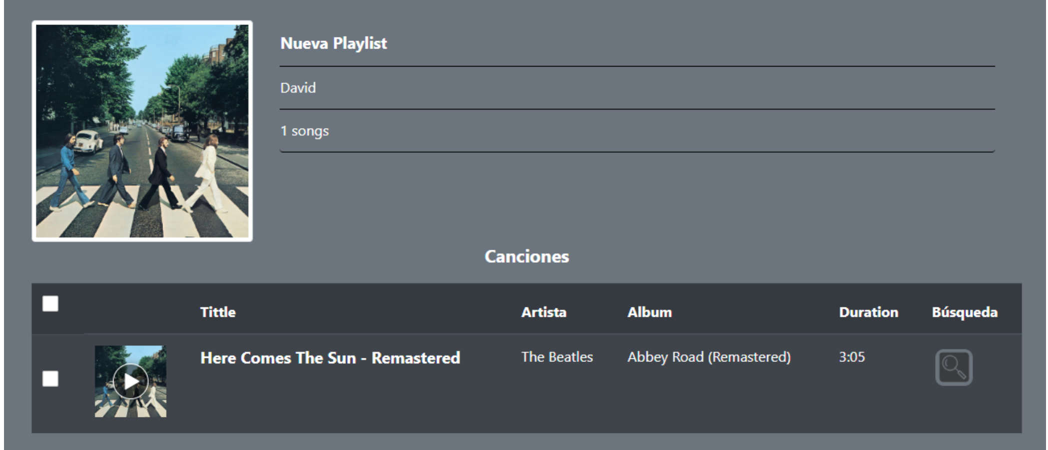


Figura 5.8: Pantalla de detalle de una lista de reproducción, donde se mostrarán las recomendaciones sobre una lista

En el conjunto de datos inicial, las listas de reproducción no tenían ningún tipo de imagen asociada, con motivo de mejorar la estética de la pantalla, se propone que al mostrar una lista de reproducción se muestren las carátulas de las primeras canciones.

Para poder hacer eso, se implementa un sistema de *lazy loading* o carga diferida que se detalla a continuación.

Carga de imágenes Una de las problemáticas de cara a la usabilidad y atractivo de la aplicación web es que las listas de reproducción del conjunto de datos inicial venían sin ningún tipo de imagen.

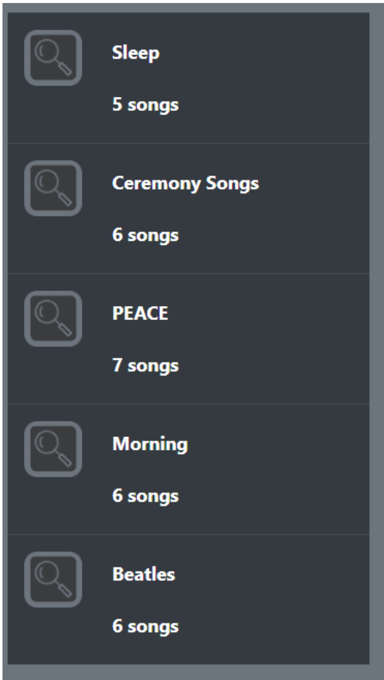


Figura 5.9: Listas de reproducción sugeridas

Para solucionarlo, para cada lista de reproducción se mostrará una imagen combinada de las carátulas de los 4 primeros álbumes de las 4 primeras canciones de la lista, que se obtienen al hacer una consulta al API de Spotify.

Esto trae otro problema, al obtener recomendaciones de N listas de reproducción se realizan 4N llamadas al API de Spotify en cada invocación de la página, lo que hace que sobrecarguemos las llamadas al API de Spotify y nos impida el acceso durante un período.

Para solucionarlo, decidimos almacenar las rutas de las imágenes en base de datos, también de forma desnormalizada, almacenando para una lista de reproducción en 4 columnas diferentes cada una de las rutas de las imágenes de los álbumes. Para llenar los datos de esas imágenes en base de datos, se diseña un mecanismo que carga las imágenes en base de datos si no están presentes cada vez que se carga una única lista de reproducción, al consultarse el detalle.

De este modo, a medida que se acceda a las distintas listas de reproducción, se irá llenando la base de datos y se harán cada vez menos consultas al API de Spotify. Para las listas de recomendaciones, se mostrará una imagen temporal mientras no acceda nadie a la lista en cuestión.

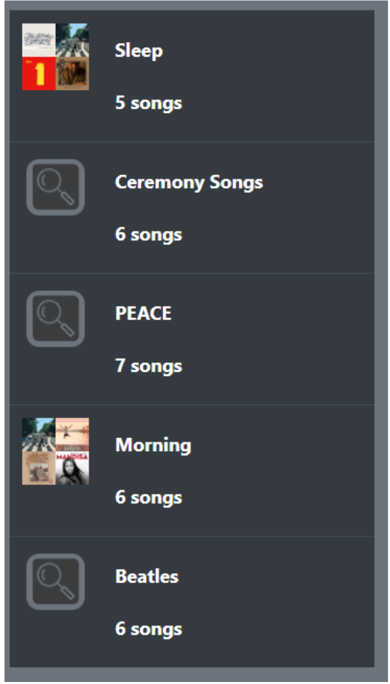


Figura 5.10: Listas de reproducción sugeridas con las imágenes cargadas

5.5.5 Sprint 4: Diversidad de recomendaciones

Una gran parte de funcionalidades empiezan a aparecer aquí, con la arquitectura y el requisito más claro, se desarrollan las siguientes funcionalidades.

(40) Mostrar recomendaciones de listas de reproducción a partir de un usuario.

Como: Usuario autenticado.

Quiero: Poder ver las recomendaciones de listas de reproducción en función de mis gustos, mostrándomelas junto a mis listas de reproducción.

Para: Descubrir música de manera general.

Se rehace ofreciendo una interfaz más amigable el servicio definido en el Sprint 1 de la historia 27. La interfaz de cara al usuario es la misma que la definida en 5.9, que se añade a la pantalla de la figura 5.7, resultando la pantalla que se muestra en 5.11.

(39) Mostrar recomendaciones de canciones para una lista de reproducción.

Como: Usuario autenticado.

Quiero: Poder ver las recomendaciones de canciones que concuerdan con una lista de reproducción

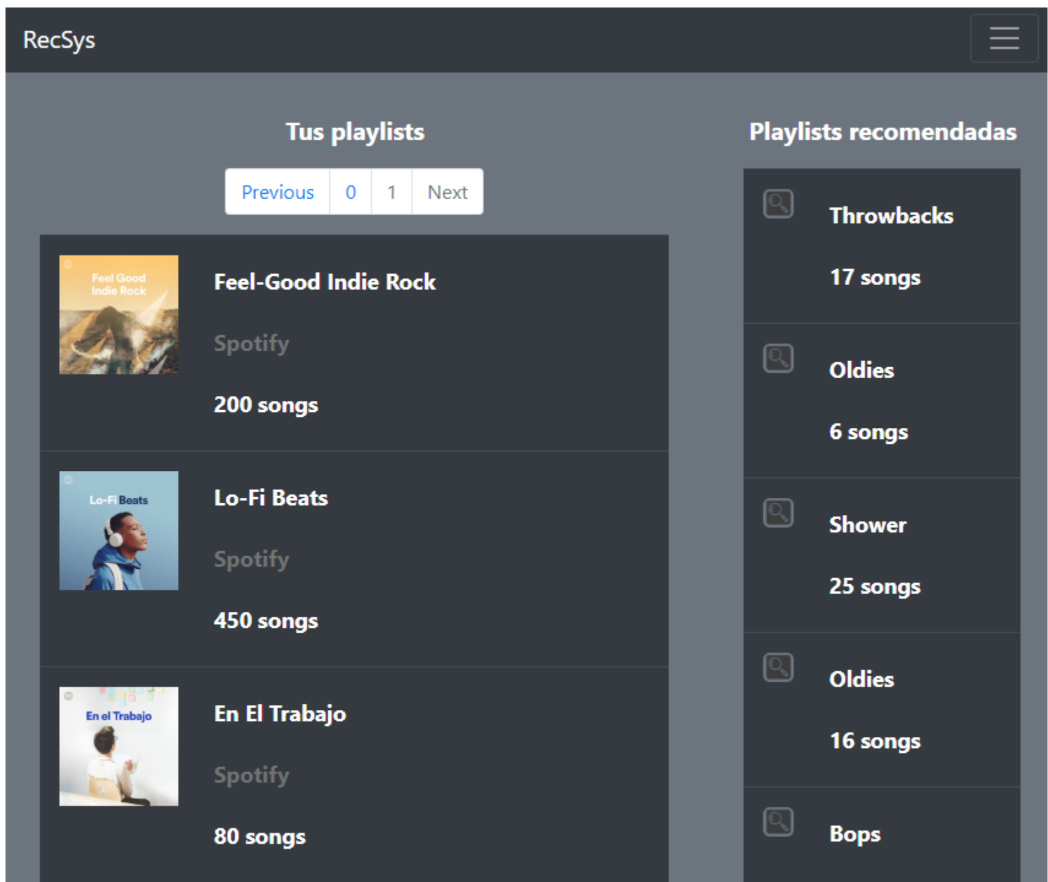


Figura 5.11: Recomendaciones a listas de reproducción del usuario, añadidas en la pantalla de visualización de sus listas

Para: Descubrir música nueva que encaje con listas de reproducción existentes.

Se rehace ofreciendo una interfaz más amigable el servicio definido en el Sprint 1 de la historia 12. Presentando las canciones en la parte inferior de la página de una lista de reproducción. En la figura 5.12 se pueden ver las nuevas recomendaciones ofrecidas añadidas en la página de visualización de detalles de una lista de reproducción.

(26) Obtener recomendaciones de canciones para un usuario.

Como: Usuario autenticado.
Quiero: Poder obtener recomendaciones de canciones según todas mis listas de reproducción.
Para: Descubrir música nueva afín a mis gustos.

Se crea el servicio REST y la interfaz en la aplicación web, se reutilizan los componentes

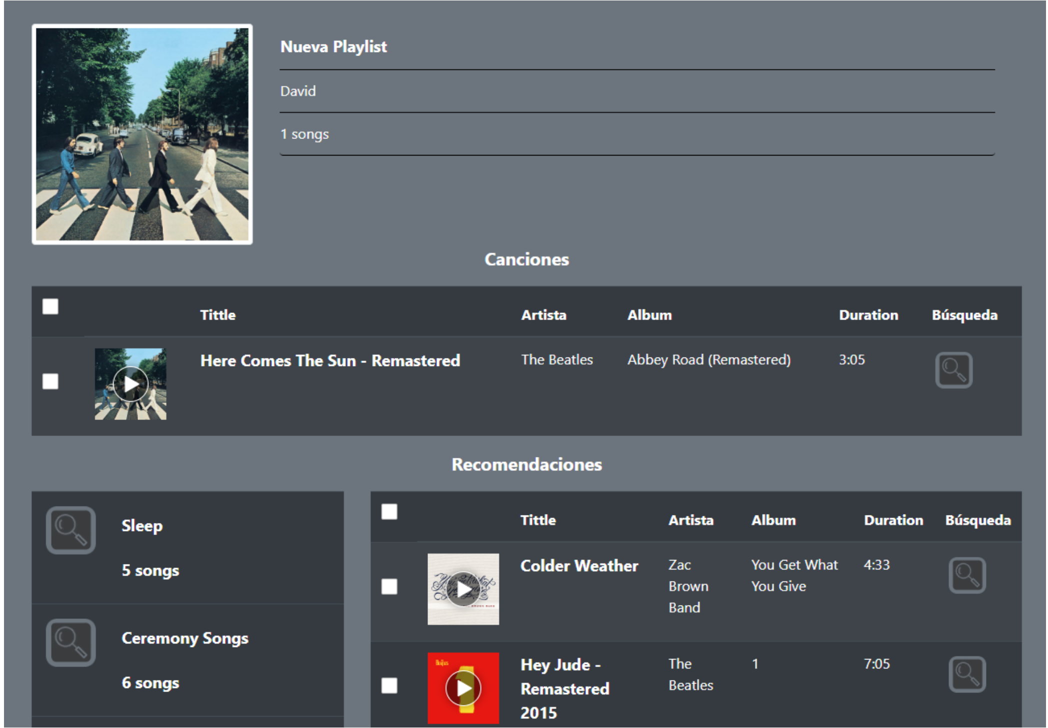


Figura 5.12: Pantalla de detalle de una lista de reproducción, donde se mostrarán las recomendaciones sobre una lista

de recomendaciones de canciones utilizados en otras historias de usuario.

Se añade la visualización de las recomendaciones de canciones en la parte inferior de la página, como se muestra en la figura 5.13

(7) Reproducción de una canción.

Como: Usuario autenticado.
Quiero: Poder reproducir una canción recomendada
Para: Saber si me va a gustar.

Spotify provee un API para poder introducir el reproductor en la propia página, y a través de un marco que se reproduzca en el propio sitio web. Se ha añadido a cada canción este marco con un botón de reproducción.

(13) Obtener recomendaciones de listas de reproducción según reproducciones recientes.

Como: Usuario autenticado.
Quiero: Poder obtener recomendaciones para las últimas canciones que haya escuchado.

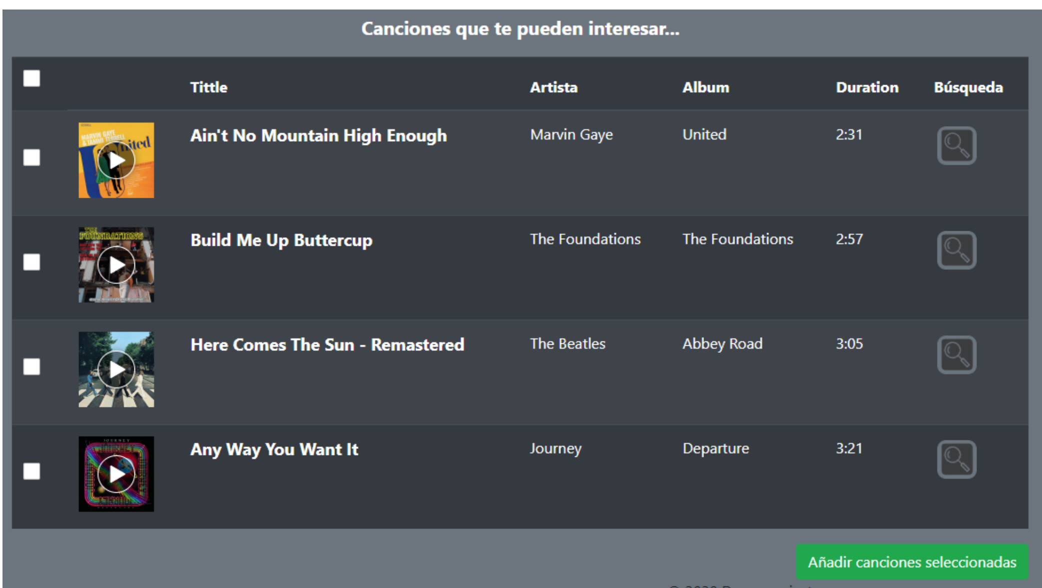


Figura 5.13: Canciones recomendadas a un usuario

Para: Descubrir nueva música en base a mis gustos recientes.

Se crea una pantalla de reproducciones recientes donde además de visualizar las últimas canciones reproducidas, se añaden las recomendaciones de listas de reproducción. Para ello se forma una lista con las últimas canciones que ha escuchado el usuario que se envía al motor de recomendaciones. Un ejemplo de esta vista se encuentra en la figura 5.14

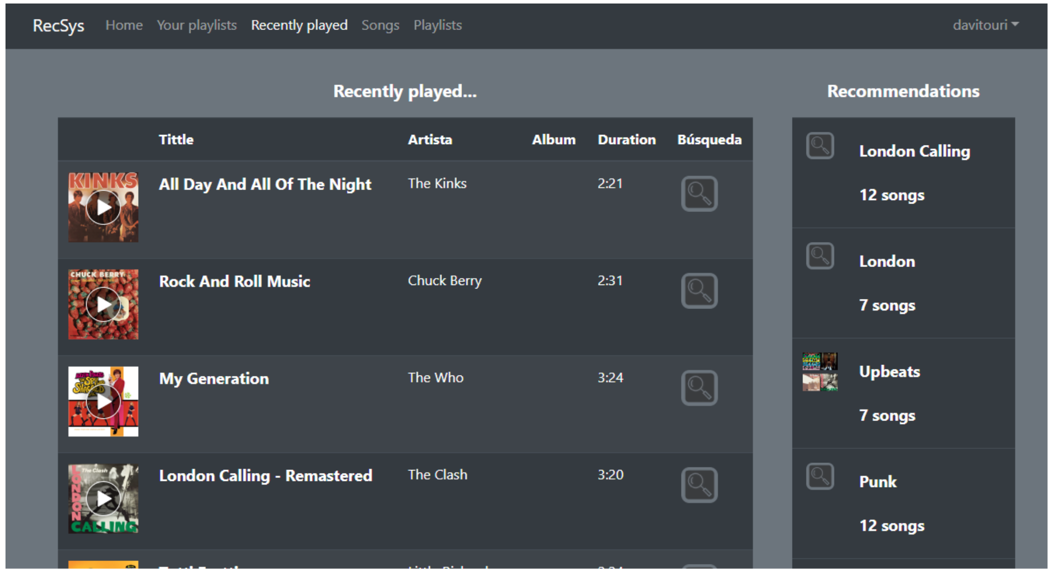


Figura 5.14: Recomendaciones según las reproducciones recientes

5.5.6 Sprint 5: Más tipos de recomendaciones y usabilidad

Se comienzan a implementar casos de uso no tan prioritarios y se finalizan los casos de uso destinados a obtener recomendaciones.

(25) Obtener recomendaciones de canciones según reproducciones recientes.

Como: Usuario autenticado.
Quiero: Obtener recomendaciones de canciones según lo último que he escuchado.
Para: Descubrir nuevas canciones que me puedan interesar recientemente.

Se aplica WSR 5.4.4 a las recomendaciones de listas de reproducción obtenidas en (13) 5.5.5. Se añade la pantalla de canciones recomendadas como en el resto de historias de usuario de recomendación de canciones.

Obtener recomendaciones de canciones según otra canción.

Como: Usuario autenticado.
Quiero: Obtener recomendaciones de canciones similares a otra canción.
Para: Descubrir canciones parecidas.

Se crea una pantalla con los detalles de una canción donde se añaden las recomendaciones, que se obtienen como las recomendaciones de canciones a una lista de reproducción compuesta por una única canción.

Obtener recomendaciones de listas de reproducción según una canción.

Como: Usuario autenticado.
Quiero: Obtener recomendaciones de listas de reproducción para una canción.
Para: Descubrir listas que encajen con una canción.

Se añade a la pantalla con los detalles de una canción las recomendaciones de listas, que se obtienen como las recomendaciones de listas a una lista de reproducción compuesta por una única canción.

(5) Buscar canciones.

Como: Usuario.
Quiero: Buscar canciones dentro de la plataforma
Para: Ver las recomendaciones para esas canciones

Se añade una pantalla de búsqueda. Se modifica el modelo de MySQL para añadir un índice FULL_TEXT sobre el nombre de la canción. Se implementa con Ajax la búsqueda de canciones a partir de un cuadro de texto. Se muestra en la figura 5.15.

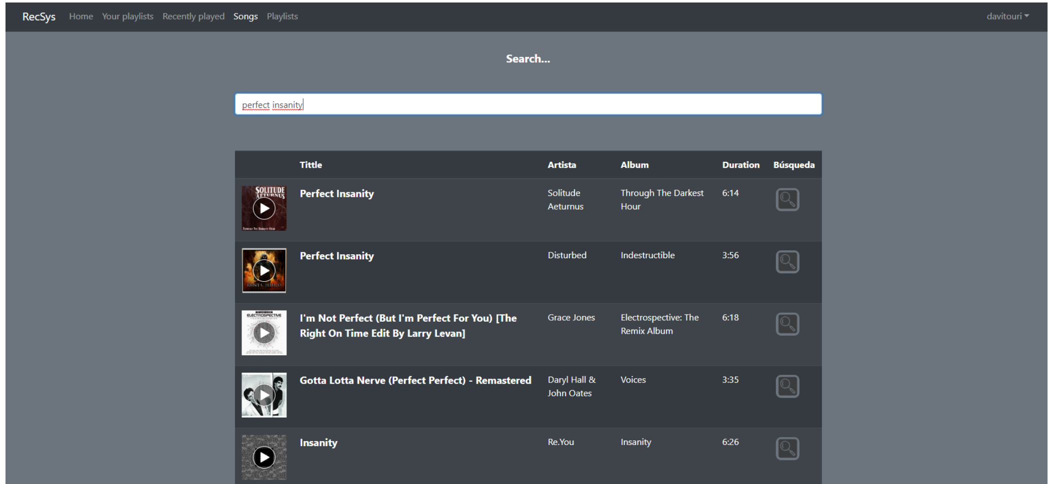


Figura 5.15: Pantalla de búsqueda de canciones

(66) Buscar listas de reproducción.

Como: Usuario.

Quiero: Buscar listas de reproducción dentro de la plataforma

Para: Ver las recomendaciones para esas listas

Se añade una pantalla de búsqueda. Se modifica el modelo de MySQL para añadir un índice FULL_TEXT sobre el nombre de la lista de reproducción. Se implementa con Ajax la búsqueda de listas a partir de un cuadro de texto. Se muestra en la figura 5.16.

(65) Visualizar recomendaciones sin estar registrado.

Como: Usuario.

Quiero: Navegar por la plataforma sin estar autenticado.

Para: Ver las recomendaciones y probar la aplicación antes de registrarme

Se modifican los permisos de algunas páginas, se permite buscar listas y canciones y visualizar sus recomendaciones sin estar registrado, solamente de las listas y canciones dentro de la plataforma, a través de las pantallas de las figuras 5.15 y 5.16.

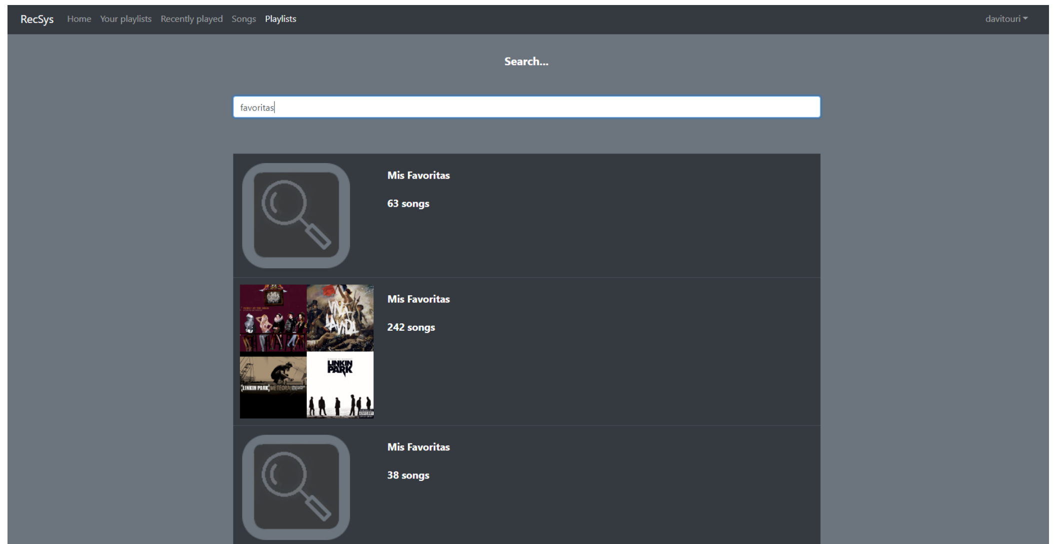


Figura 5.16: Pantalla de búsqueda de listas de reproducción

5.5.7 Sprint 6: Mejoras de usabilidad final

Sprint final, se realizan las últimas funcionalidades del Product Backlog.

Añadir Canciones.

Como: Usuario autenticado.
Quiero: Poder añadir las canciones recomendadas a mis listas de reproducción.
Para: Que suenen cuando reproduzco alguna de mis listas.

Se añade en la visualización de las recomendaciones unas casillas de verificación, con las que podemos añadir las canciones a la cuenta. Al añadir las canciones se actualiza la lista de canciones de la lista de reproducción objetivo y se recargan las recomendaciones mediante Ajax para ofrecer nuevas. Para las pantallas en las que no se recomiendan canciones para una lista de usuario se crea un modal donde se puede escoger entre las diferentes listas del usuario o crear una nueva, como se muestra en las figuras 5.13 y 5.17.

Borrar Canciones.

Como: Usuario autenticado.
Quiero: Poder borrar canciones de mis listas de reproducción.
Para: Eliminar las canciones que ya no encajen de una lista.

Se añaden checkbox a la lista de canciones de una lista de reproducción para eliminarlas.

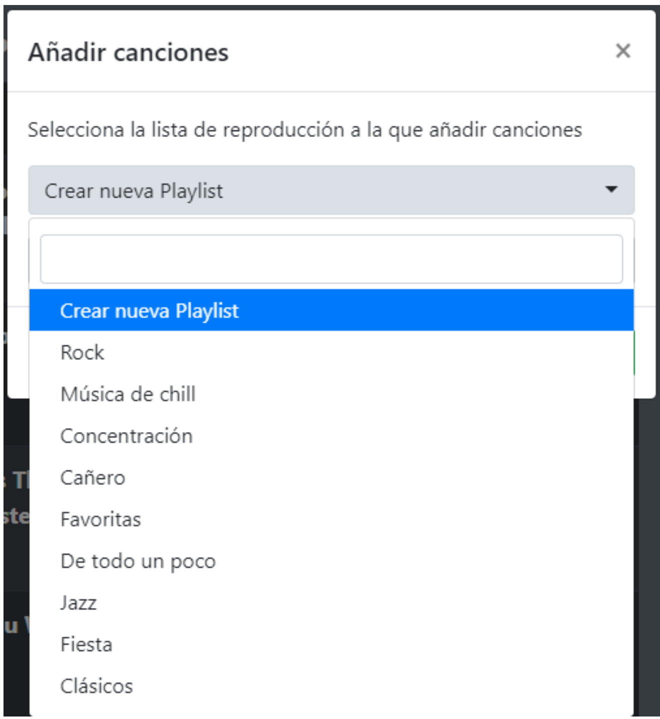


Figura 5.17: Al añadir una canción, se muestran las playlists del usuario en un modal.

Se confirma con un modal y se borran de la lista. El botón se coloca en la parte final del listado de canciones como se muestra en 5.18

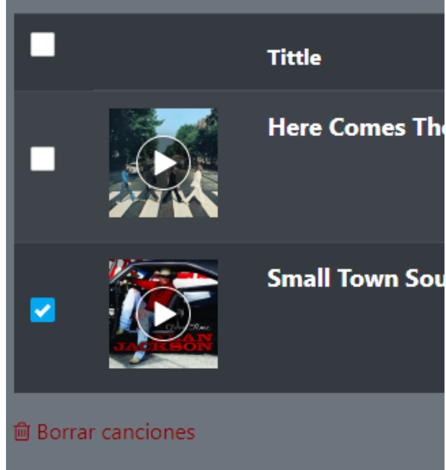


Figura 5.18: Botón de borrado de canciones al final de la lista.

Añadir listas de reproducción.

Como: Usuario autenticado.

Quiero: Añadir las listas de reproducción recomendadas a mi cuenta
Para: Poder reproducirlas y modificarlas desde mi cuenta de Spotify

Se añade un botón que crea en el usuario una lista de reproducción con todas las canciones que contiene.

5.5.8 Acceso a datos

Para el acceso a los datos se ha utilizado el conector de redis-py en el artefacto Python y JDBC abstraído mediante Spring en la interfaz NamedParameterJdbcTemplate de Spring 5 para el artefacto Java.

Para las listas de reproducción del usuario, el dato se encuentra en Spotify, y se accede a él mediante a la librería que se explica en el siguiente apartado.

Aparece un problema entonces, ante la visualización de una lista de reproducción, el sistema tienen que tratar la información que se le muestra de distintos datos. Para ello, definimos dos endpoints, uno para tratar las listas internas, y otro para las de spotify. Se representa en el diagrama 5.19 de la página 62. Se ha simplificado el diagrama, las llamadas al servicio de recomendaciones son 2, que se ejecutan en paralelo, junto a la obtención de los identificadores de canción en el caso de que sea una lista de reproducción interna.

5.5.9 Mapeo de entidades de dominio

Al estar formado de tantas capas el proyecto, distinguimos diversos tipos de entidades de dominio en los distintos puntos de la aplicación.

- Por un lado tenemos las entidades de listas y canciones de la vista, que solo muestra la información necesaria para que el usuario pueda ver la aplicación.
- También están las entidades de Acceso a datos que representan el objeto tal cual está en base de datos
- Por otro lado tenemos las entidades de Spotify, que tienen una gran parte de información que no se utiliza dentro de la aplicación.
- Por último tenemos las entidades del proceso de recomendaciones, que está marcado por un contrato de servicios en Swagger 2.0.

Para realizar las distintas transformaciones de unas entidades a otras, se ha utilizado en mayor medida Mapstruct, una librería Java que genera el código sobre los mapeadores que se definan en Interfaces Java. Los mapeadores funcionan automáticamente mediante los nombres de los campos, pero también se puede personalizar y desarrollar a medida, simplificando igualmente algunos puntos.

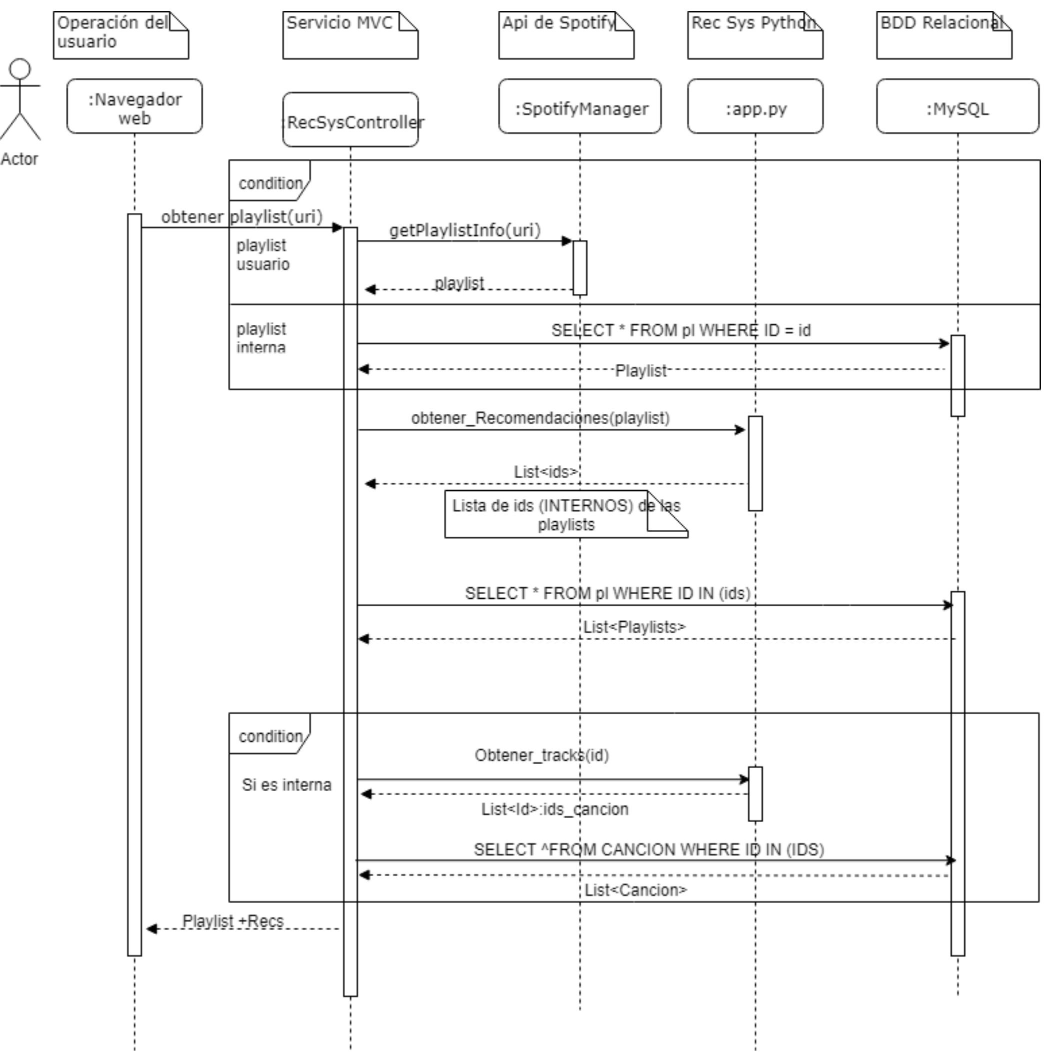


Figura 5.19: Diagrama de secuencia para mostrar una lista de reproducción (simplificado)

Capítulo 6

Análisis de las recomendaciones

En este capítulo se analizarán los beneficios de la implementación de la búsqueda de vecindarios desde un punto de vista práctico. Los fundamentos de esta búsqueda se explican en la sección 2.4.1 de la página 8, y su implementación en concreto en la sección 5.4 de la página 40.

Recordando lo comentado en la sección 2.4.1 de la página 9, la búsqueda se organizaba en un árbol. Esta búsqueda se puede realizar sobre una rama o varias, para lo que variamos el parámetro *k-clúster*. Si este valor se acerca a 1000 ($\sqrt{1000000}$) recorreríamos el árbol entero. Recorrer más puntos del árbol aumentará la calidad de los resultados a costa de perder eficiencia, durante las pruebas hemos probado diferentes valores hasta encontrar uno suficientemente elevado, pero que no afectase en exceso al tiempo.

6.1 Datos de prueba

Para los datos de prueba se ha seleccionado un pequeño conjunto de listas de reproducción públicas de Spotify. Se han incorporado a la cuenta personal y se han sacado los identificadores de las canciones de esas listas a partir de los registros de la aplicación. Este conjunto se ha utilizado para medir los tiempos de carga de las páginas, así como para medir la calidad y tiempos de a-NN de manera tanto automatizada como personal.

A mayores, se ha creado un *script* de Python para generar listas de reproducción aleatoria y comprobar que las pruebas eran suficientemente representativas. Se han visto correspondencias en los tiempos pero no en la calidad de las recomendaciones, debido a que al ser aleatorias las distancia entre el vecino más lejano y el vecino más corto tiende a acercarse.

6.2 Implementación de k-NN

Con objetivo de realizar la comparativa, se ha implementado una solución simple de k-NN para encontrar los k-vecinos más próximos. Para ello se ha utilizado la matriz dispersa definida en 5.5.1 de la página 43. Se ha hecho otro servicio reutilizando parte del código del servicio que utiliza a-NN, este servicio utiliza la función *cosine_similarity* del módulo *sklearn.metrics.pairwise* de scikit-learn [36].

6.3 Comparación a-NN frente a k-NN

Para medir la validez de a-NN se han hecho mediciones de diferentes listas de reproducción. En la prueba, se compara el top vecinos aproximados medidos con a-NN con el top vecinos medido con k-NN.

Se ha medido el porcentaje de éxito total, que representa el porcentaje de listas de reproducción recuperadas en a-NN que se encuentran en el top de k-NN (Los mejores vecinos). Hemos hecho 2 mediciones:

- **Top-k-success**: Es el porcentaje de los k-vecinos de a-NN que se encuentran en los mismos k-vecinos de k-NN
- **Top-2k-success**: Es el porcentaje de los k-vecinos de a-NN que se encuentran dentro del doble de vecinos más cercanos de k-NN. Este valor nos sirve para medir que las recomendaciones pese a no ser las mejores, son bastante buenas, ya que se encuentran dentro del top de k-NN.

Los resultados individuales se pueden ver en el cuadro A.1. Los resultados resumidos en el cuadro 6.1.

k-clúster	Top-k success	Top-2k success
5	34,55%	61,82%
10	41,82%	63,64%
20	44,00%	64,00%
40	58,18%	72,73%

Cuadro 6.1: Comparación a-NN - Resumen

A pesar de que pueda parecer que buscando entre 5 y 20 clústeres sólo hay entre un 34%-44% de éxito, es un buen número, en el desglose del cuadro A.1 se puede ver que en muchos casos ha acertado con el 100% de las recomendaciones. En el resto de casos las recomendaciones que proporciona a-NN siguen estando en la cima de las recomendaciones que proporciona k-NN salvo en algunas ocasiones.

Observando la aplicación como un usuario, a partir de 3 clústeres de búsqueda se empiezan a obtener buenas recomendaciones, quitando algunos casos localizados donde no se encuentran recomendaciones. Con 10 clústeres estos problemas desaparecen.

Con este experimento medimos únicamente como de bueno es a-NN a la hora de encontrar un clúster de vecino lo más parecido al que hubiera sido proporcionando por k-NN. Como trabajo futuro queremos evaluar como afectaría el trabajar con los vecinos aproximados en cuánto a la calidad de las recomendaciones de cara al usuario [37, 38]. Es de esperar que la pérdida de concordancia entre k-NN y a-NN no degrade significativamente la calidad del la recomendación.

6.4 Tiempos

6.4.1 Tiempos de a-NN

Se ha comparado el tiempo de a-NN contra el de k-NN. Se han probado en diferentes casos el número de clústeres visitados para encontrar el punto donde ofrezca mejor calidad en un tiempo razonable. Los resultados se indican en el cuadro 6.2. Los errores representan el porcentaje de vecinos que no se han podido encontrar (distancia coseno = 1).

k-clúster	**Tiempo/petición**	**Errores**	**Mejora k-NN**
1	0,0363	27,27%	99,24%
2	0,0480	27,27%	99,00%
3	0,0612	18,18%	98,73%
4	0,0766	18,18%	98,41%
10	0,1665	9,09%	96,54%
15	0,2385	9,09%	95,04%
20	0,3030	0,00%	93,70%
25	0,3747	0,00%	92,21%
30	0,4562	0,00%	90,51%
35	0,5267	0,00%	89,04%
55	0,8171	0,00%	83,01%
75	1,1014	0,00%	77,09%
95	1,3746	0,00%	71,41%
115	1,6700	0,00%	65,27%
135	1,9605	0,00%	59,22%
155	2,2574	0,00%	53,05%
175	2,5754	0,00%	46,44%
195	2,8300	0,00%	41,14%
1000	14,6518	0,00%	-204,74%
k-NN	4,8080	0,00%	0,00%

Cuadro 6.2: Tiempos a-NN y k-NN

En 6.2 se puede observar una mejora de más de un 90% del tiempo para la búsqueda en 30 *k-clústeres* o menos. Interesa obtener la mejor calidad posible en el menor tiempo. Como se comentaba en 6.3, a partir de 3 clústeres ya se notaba una buena calidad de la recomendaciones, siendo así un 98,73% de mejora respecto a k-NN.

También se puede observar que para recorrer todos los clústeres (k-clústeres = 1000), el tiempo es 3 veces superiores. Esta diferencia puede deberse a que el programa tiene que recorrer el árbol de búsqueda entero en lugar de hacerlo en la matriz directamente.

6.4.2 Tiempos de carga

Para valorar un buen valor para ofrecer recomendaciones de calidad sin afectar demasiado a los tiempos, se han medido los tiempos de carga de la web para los diferentes valores de los *k-clústeres* más considerados. Con esta medida tenemos una referencia con el cómputo global de obtener las diferentes recomendaciones en una pantalla, traer los datos del servidor de Spotify, leer los datos de la base de datos, y renderizar la pantalla.

Los datos se pueden ver en el cuadro 6.3. Si nos marcamos como objetivo poder cargar la página en alrededor de 1 segundo, con *k-clústeres=20* nos aseguramos de una buena calidad (64% de recomendaciones dentro del top $2k$ además de unos bajos tiempos.

Playlist	**t K = 5 (s)**	**t K = 10 (s)**	**t K = 20 (s)**	**t K = 40**
Classical Romance	0,853	0,926	1,190	1,360
Temazos Chill	1,010	1,040	1,070	1,810
Hits alegres	0,700	0,766	0,903	1,280
Chillin' On A Dirt Road	0,894	0,822	0,994	1,710
Chopin Nocturnes	0,462	0,486	0,738	1,100
Dinner with Friends	0,847	0,894	1,050	1,480
En El Trabajo	0,814	0,889	1,140	1,680
Canciones recientes	0,489	0,406	0,663	1,040
User playlists	2,030	2,210	2,280	2,780
Infancia	0,572	0,552	0,627	0,997
Clásica	0,388	0,278	0,439	0,822
Media	**0,824**	**0,843**	**1,009**	**1,460**

Cuadro 6.3: Tiempos de carga de la web en `/playlist` para los diferentes valores de *k-clústeres*

Se puede ver que de realizar la búsqueda en *k-clústeres = 5* a realizarla en 10 no hay aparente diferencia. Entre *k-clústeres = 10* y *k-clústeres = 20* sí se produce una diferencia, pero como el objetivo era acercarse a 1 segundo, y en *k-clústeres = 20* la calidad es buena, nos quedamos con este valor. Si utilizáramos k-NN se tardaría más de 5 segundos y la página web sería inutilizable.

Capítulo 7

Configuración y Despliegue

Con la puesta en producción de la aplicación necesitamos ofrecer una solución escalable y eficiente para que se pueda beneficiar de la aplicación el mayor número de usuarios. En este capítulo se explicará en qué consiste la tecnología de contenedores, cómo lo podemos aplicar en este proyecto y las herramientas necesarias para hacerlo. Se explicará toda la configuración de los componentes de la aplicación para llevarla a un entorno productivo, en nuestro caso, un clúster de Kubernetes en la nube de Microsoft Azure.

Durante todo el capítulo se hará hincapié en los cuatro artefactos que componen el proyecto, indicados en el diagrama 5.1 de la página 37. Tenemos cuatro artefactos que los convertiremos en cuatro contenedores independientes. Estos contenedores se orquestarán en el clúster de kubernetes y tendrán apoyo de otros componentes de Azure.

Para los dos artefactos principales, el servicio de recomendaciones y la aplicación web, se construirá una imagen utilizando Docker. Estas imágenes se almacenarán en Azure Container Registry, subidas mediante *docker*. Junto a otras dos de redis y MySQL se añadirán a un clúster de Kubernetes, que se podrá administrar mediante la herramienta *kubectl*. Para algunos contenedores se montará un volumen a una serie de archivos en Azure Files. Un balanceador redirigirá el tráfico a los nodos del clúster si procede.

En la figura 7.1 se puede ver un diagrama que resume la arquitectura.

7.1 Contenedores

Los contenedores sirven como alternativa a las máquinas virtuales para ejecutar procesos en entornos aislados. Un contenedor es un proceso aislado del sistema anfitrión y posee todas las herramientas y dependencias necesarias para su ejecución. Los contenedores se pueden ejecutar de forma nativa en Linux y comparten el *kernel* con el sistema operativo anfitrión, al contrario que con máquinas virtuales, que es necesario un sistema operativo invitado por cada instancia.[31]

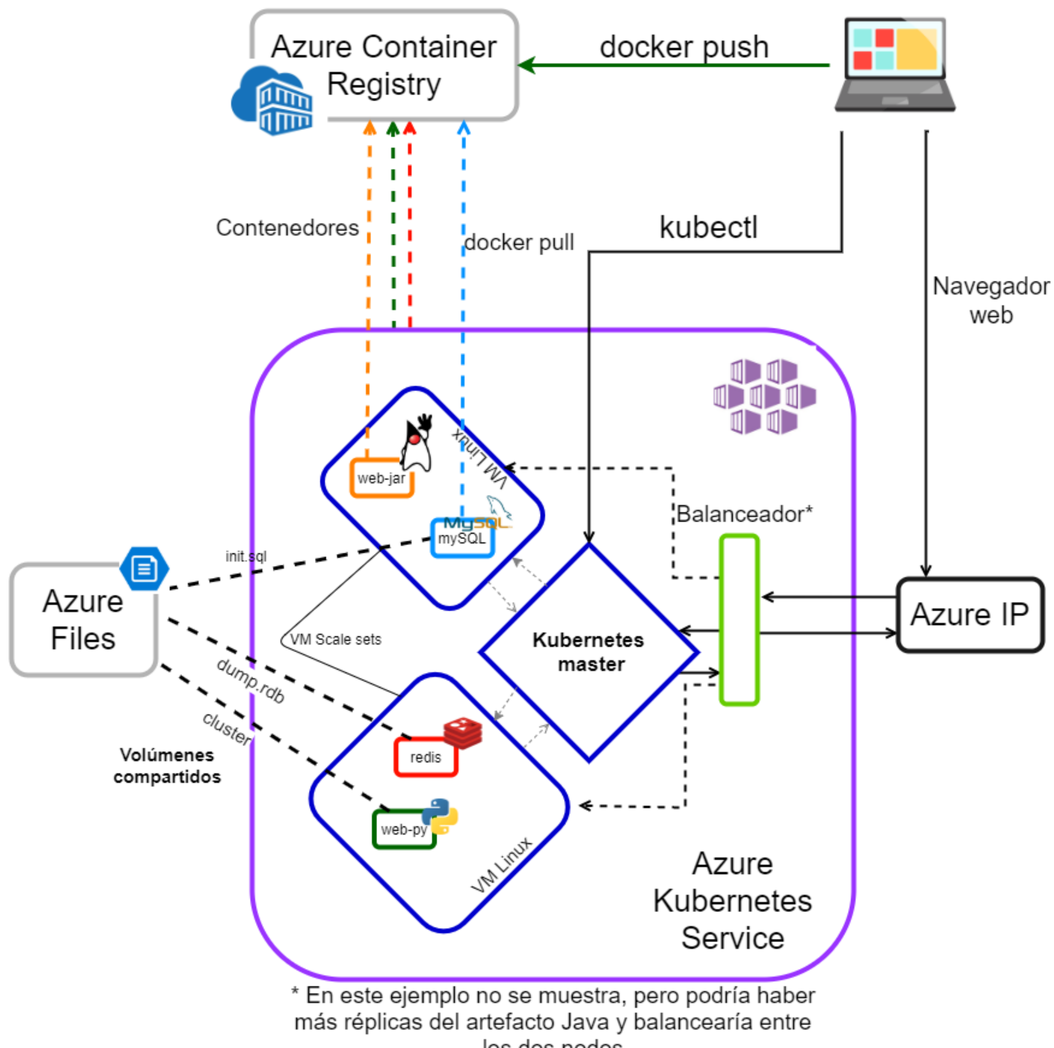
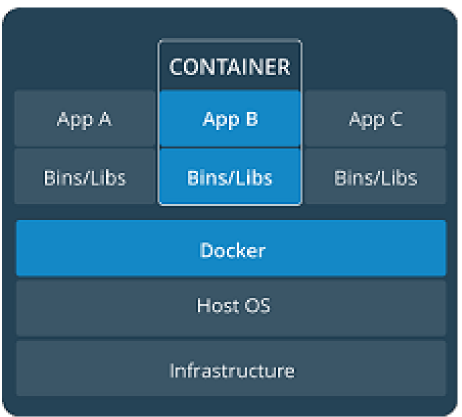


Figura 7.1: Diagrama de infraestructura de los sistemas en Azure

Si recordamos el diagrama de la figura 5.1 de la página 37, la aplicación está compuesta por cuatro artefactos: El servidor web basado en Spring boot (Java), el servicio REST de recomendaciones basado en Flask (Python), y los dos artefactos de datos, una base de datos MySQL y un almacén clave-valor Redis.

Estos cuatro artefactos serán 4 contenedores aislados desplegados en un clúster Kubernetes de Azure.



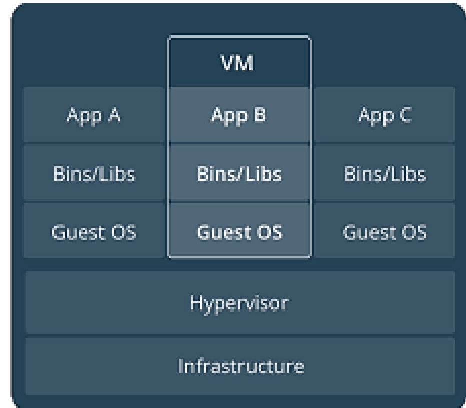


Figura 7.2: Diferencia entre el uso de contenedores o de máquinas virtuales

7.2 Docker y Kubernetes

Docker es una plataforma para desarrollar, entregar y arrancar aplicaciones. Con su modelo de contenedores permite separar las aplicaciones de la infraestructura. Se destacan dos elementos clave:

- Imágenes: Una plantilla para crear un contenedor. Por lo general las imágenes están basadas en otras imágenes y se van especializando. Por ejemplo una imagen de una aplicación puede estar basada en una imagen de Debian, con la configuración de la aplicación por encima. Para definir una imagen se utiliza una plantilla Dockerfile.

- Contenedores: Son instancias de las imágenes. Se pueden crear, borrar, arrancar o parar utilizando la interfaz de comandos *docker*. Cuando se para un contenedor se mantiene su estado.

Docker nos permite crear imágenes y levantar los contenedores en cualquier infraestructura que tenga la tecnología, además de facilitar la escalabilidad horizontal. Para construir las imágenes tenemos los archivos Dockerfile. Si queremos arrancar y configurar un conjunto de imágenes podemos utilizar Docker Compose, o Kubernetes para un entorno productivo.

7.2.1 Dockerfile

Docker dispone de una herramienta para construir imágenes a partir de un Dockerfile. Se trata de un documento de texto compuesto de una serie de comandos para ejecutar *scripts* o utilizar archivos de la máquina anfitrión.

En el ejemplo de 7.1 se puede ver un ejemplo de la estructura del fichero y los comandos más habituales. La instrucción CMD se ejecutará al arrancar el contenedor, mientras que el resto se utilizan para construir la imagen.

```
1 FROM image
2 ENV foo /bar
3 WORKDIR ${foo}   # WORKDIR /bar
4 ADD . $foo       # ADD . /bar
5 RUN chmod +x ${foo}/start-server.sh
6 CMD ${foo}/start-server.sh
```

Listing 7.1: Ejemplo Dockerfile

Dentro de este proyecto, se han definido dos Dockerfile:

- Dockerfile para el artefacto de la aplicación web, con una imagen base oficial de Openjdk [39]. Para construir esta imagen, se siguen los siguientes pasos en el fichero Dockerfile:
  - Se utiliza la imagen base *openjdk*. (Por defecto la última versión).
  - Se copia el archivo con extensión JAR a la imagen.
  - Se indica el *entrypoint* para que al levantar el contenedor se ejecute el comando *java -jar app.jar*
- Dockerfile para el artefacto de servicio de recomendaciones REST, utilizando una imagen base especializada para hacer de servidor web para flask con *uwsgi y nginx* [40].

  Para el correcto funcionamiento de esta imagen, es necesario montar un volumen con el clúster de búsqueda materializado y pasarle la ruta por variable de entorno. Se ha hecho así para no sobrecargar la imagen ya que se trata de un archivo binario de 1,5Gb.

  Para construir esta imagen, se siguen los siguientes pasos en el fichero Dockerfile:
  - Se utiliza la imagen base *tiangolo/uwsgi-nginx-flask:python3.6*.
  - Se copia el proyecto y sus dependencias en formato fichero *requirements.txt* a la nueva imagen.
  - Se instalan todas las dependencias utilizando *pip*.

  Esta imagen ya gestiona el arranque de la aplicación utilizando UWSGI a partir de un fichero *uwsgi.ini* definido previamente en el proyecto.

7.2.2 Docker Compose

Docker Compose es una herramienta para administrar y manejar aplicaciones compuestas por múltiples contenedores. Para utilizarlo, se describe un fichero en formato YAML donde se indica la configuración de los diferentes contenedores, así como las dependencias entre los mismos.

Docker Compose permite definir redes, volúmenes, dependencias, variables de entorno, imagenes base a utilizar de un repositorio remoto o local, o directorios donde construir la imagen a partir de un Dockerfile.

En este proyecto, se ha definido un fichero *docker-compose.yml* con los siguientes contenedores y sus configuraciones:

- web-jar: Aplicación web basada en Spring Boot, se configura a partir del Dockerfile del artefacto.
  - Se expone el puerto 8080 al 8080 de la máquina anfitrión.
  - Dependencia del servicio de recomendaciones y la base de datos MySQL.
  - Variables de entorno para las conexiones a la base de datos o al servicio de recomendaciones y para el nivel de log de la aplicación.
- redis: Almacén clave-valor que se utiliza en el servicio de recomendaciones para hacer correspondencias entre los identificadores de Spotify y los que usa internamente la aplicación.
  - Se utiliza la imagen base *redis:alpine*.
  - Se expone el puerto 6379 al 6379 de la máquina anfitrión para depurar en local.
  - Dependencia del servicio de recomendaciones y la base de datos MySQL.
  - Volumen compartido que contiene el almacén de datos ya cargado en Redis en /data.
- web-py: Aplicación de servicios basada en Flask, se configura a partir del Dockerfile del artefacto.
  - Se expone el puerto 80 al 5000 de la máquina anfitrión.
  - Dependencia del almacén clave-valor Redis.
  - Variables de entorno para la conexión con el servidor de Redis y el volumen compartido.
  - Volumen compartido que contiene el clúster de búsqueda de recomendaciones materializado en binario.
- db: Base de datos MySQL 5.7
  - Se utiliza la imagen base *mysql:5.7*
  - Variables de entorno para los datos del usuario y creación de la base de datos.
  - Volumen compartido que contiene el archivo de inicialización de la base de datos con las listas de reproducción y las canciones.

7.2.3 Kubernetes

Ya se ha visto qué es una imagen, qué es un contenedor y cómo se crean y configuran. A la hora de llevar todos estos elementos a producción, necesitamos una herramienta para administrarlos.

Kubernetes es una plataforma para orquestar contenedores. Permite orquestar, escalar y administrar la infraestructura de las aplicaciones de manera genérica, además de facilitar la automatización de estas tareas. Está desacoplado del desarrollo de las aplicaciones, por lo que tendrá que realizarse en otro flujo de trabajo, y las imágenes de los contenedores generadas serán las que se instancien en el clúster de Kubernetes.

Para definir los despliegues, se debe crear un documento en formato YAML similar al docker-compose. Se pueden crear varios documentos para cada tipo de servicio o se puede definir en el mismo.

En kubernetes tenemos 3 conceptos que se han utilizado en este proyecto:

- Pods: Representan a un contenedor en concreto de la aplicación. Podrían utilizarse varios contenedores en el mismo pod pero no es necesario. Los Pods son efímeros, pueden crearse y desaparecer. Utilizando la herramienta *kubectl* podemos ver los pods, ejecutar instrucciones dentro de ellos, pararlos o ver sus registros.

- Deployment: Representa un despliegue, se define en la plantilla YAML de despliegue y define cuantos pods puede tener un contenedor o grupo de contenedores, la imagen base a utilizar, etc.

- Service: Son los servicios expuestos, se configuran también en la plantilla YAML y marcan las direcciones y mapeos de puertos.

En el documento YAML de despliegue debemos definir los cuatro tipos de despliegues, y para cada uno de ellos sus servicios.

En A.1 se puede ver un ejemplo del fichero de definición de servicios de Kubernetes. En el ejemplo, se le está especificando la dirección IP pública que se ha configurado y se indica que exponga en el puerto 80 el puerto interno del contenedor 8080.

En A.2 se puede ver un YAML de despliegue en Kubernetes del artefacto redis de la aplicación. En esta configuración se puede ver: el número de réplicas, 1; la imagen base, *redis:alpine*; el volumen montado en /data y el puerto donde escucha el contenedor.

En esta solución, se ha utilizado el servicio del cloud de Azure, Azure Kubernetes Service (AKS) para la implantación del clúster de Kubernetes, que se detalla en la siguiente sección.

7.3 Despliegue en la nube: Azure

La informática en la nube permite alquilar recursos de infraestructura y plataforma a otra empresa y pagar por lo que se usa. De esta forma, nos ahorramos las siguientes necesidades:

- Comprar el hardware y actualizarlo.
- Tener una localización para los servidores.
- Seguridad y acondicionamiento de esa localización.
- Estar al día y aplicar las actualizaciones de seguridad de los sistemas operativos y las infraestructuras a bajo nivel.

Todas estas responsabilidades pasan a cargo del proveedor del servicio, que pueden ofrecer la nube de diferentes formas:

- Infrastructure as a Service (IaaS): Corresponde a todos los servicios de infraestructura, por ejemplo, la creación de máquinas virtuales donde el cliente debe administrar el sistema operativo.
- Platform as a Service (PaaS): Corresponde a los servicios de plataforma más especializados, por ejemplo, al desplegar un contenedor la responsabilidad de manejar el sistema operativo actualizado sería del proveedor. Es el modelo que se ha escogido en este proyecto ya que el clúster de Kubernetes está administrado por Azure. Se podría haber creado manualmente utilizando tecnologías de IaaS, pero no aportaba ningún beneficio.
- Software as a Service (SaaS): Corresponde a servicios de software como 365, donde los usuarios pueden manejar la aplicación a través de Internet.

Se ha utilizado Azure como proveedor debido a la familiaridad del alumno con el entorno. No se ha hecho ningún análisis de los diferentes proveedores de servicios en la nube a día de hoy para la elección. Para afrontar los costes de utilizar estos servicios, se ha utilizado una cuenta de Azure for Students, que incluye 100$ de créditos durante un año de forma gratuita.

Durante el resto de secciones, se verán los diferentes componentes de Azure que forman parte de la solución de este proyecto. Azure dispone tanto de un portal con una interfaz visual como de una interfaz por línea de comandos con muchas más utilidades. Durante el desarrollo de este proyecto se han utilizado conjuntamente dependiendo de la necesidad.

7.3.1 Azure Container Registry

En este proyecto se han elaborado dos imágenes con Docker, explicado en la sección de Dockerfile 7.2.1. Estas imágenes necesitamos subirlas a la nube par poder realizar el despliegue en Kubernetes. Para subirlas, se ha utilizado Azure Container Registry.

Utiliza Docker Registry 2.0, de código abierto, para almacenar y administrar las imágenes de los contenedores. Ofrece 3 planes de diferentes precios; Básico, Standard y Premium. Se ha utilizado el nivel básico ya que contempla los casos necesarios en este proyecto. No se necesitan ni hay presupuesto para grandes cantidades de almacenamiento ni para replicación geográfica.

Se ha creado a partir de Azure Portal. Para subir las imágenes se ha utilizado la propia herramienta de docker, que a partir de unos credenciales proporcionados al crear el Azure Container Registry permite subir las imágenes locales utilizando los comandos *docker tag* y *docker push*.

Desde Azure Portal se pueden ver las imágenes subidas al registro y programar otras acciones como webhooks.

7.3.2 Azure Files

La aplicación tiene 3 grupos de datos diferenciados, necesitamos compartir estos datos con los contenedores pero sin que formen parte de su ciclo de vida. Para ello hemos utilizado Azure Files.

Azure Files nos permite administrar varios sistemas de ficheros en la nube. Soporta unicamente el protocolo SMB3. En este proyecto se han identificado y subido a Azure Files los siguientes archivos:

- *dump.rdb*: Es todo el almacén clave-valor de redis, almacenado en formato de SNAPSHOT. Contiene las relaciones de los ids y listas de reproducción. Pesa 455 Mb.
- *multicluster*: Es un fichero binario con el árbol para la búsqueda de vecinos. Se ha materializado con pickle para ahorrar tiempos de carga. Pesa 1,575 Gb
- *init.sql*: Contiene la información entendible sobre listas de reproducción y canciones dentro del espacio de búsqueda de recomendaciones, para inicializar la base de datos MySQL. Pesa 515 Mb.

Estos ficheros se comparten con sus respectivos contenedores. Con el contenedor de redis, *dump.rdb*; con el contenedor del sistema de recomendaciones, *multicluster*; y con el contenedor MySQL, el fichero *init.sql*. Para compartirlos se montan los volúmenes en la especificación del fichero de Kubernetes, como se ve observa la figura A.2 de la página 83.

Los 3 grupos se han creado y se han subido los ficheros desde Azure Portal, a través de la interfaz web.

7.3.3 Azure Public IP

Azure proporciona direcciones IP públicas para exponer recursos a Internet. En este caso, se necesita exponer la aplicación web para poder interactuar con ella. En Azure las direcciones IP se ofrecen en diferentes tamaños o SKUs. No se pueden mezclar productos de diferentes SKUs como un balanceador de carga y una IP pública. Se ha escogido el SKU Standard.

Además, permite crear un dominio con el formato *<nombre>.<region>.cloudapp.azure.com*, en nuestro caso *recsys.westeurope.cloudapp.azure.com*.

Para crear la IP se ha utilizado el comando siguiente desde la consola:

az network public-ip create -g recsys –name PublicIP –sku Standard –allocation-method Static –dns-name recsys

Donde:

- -g recsys: Implica que el recurso lo tiene que crear en el grupo llamado recsys, creado previamente desde Azure Portal.
- –name PublicIP: Es el nombre del recurso que lo identifica dentro del grupo recsys.
- –sku Standard: Plan de venta, existe el Basic y el Standard. Se ha escogido Standard para ir acorde al SKU del balanceador de carga de AKS.
- –allocation-method Static: Para IPs dinámicas o estáticas en este caso.
- –dns-name recsys : Nombre del dominio, que corresponde a lo explicado en el punto anterior

7.3.4 Azure Kubernetes Service

Azure Kubernetes Service (AKS) es una manera simplificada para desplegar un clúster de Kubernetes en Azure. Azure se encarga de la monitorización y el mantenimiento del clúster. El clúster se ha creado a través de Azure Portal con las siguientes propiedades:

- Versión de Kubernetes 1.15.11.
- Dos nodos.
- Cada nodo tiene dos vCPUs, 8 Gb de memoria RAM y admite hasta 6400 operaciones de entrada/salida por segundo.

- Sistema operativo Linux

Podemos acceder al clúster mediante la herramienta de línea de comandos *kubectl* y comprobar el estado de los nodos, iniciar despliegues, ver los diferentes servicios o mostrar los logs de algún contenedor, entre otros.

Para crear el clúster, Azure crea la infraestructura por debajo utilizando su propio IaaS. Para ello crea un grupo de recursos donde crea los componentes necesarios para AKS.

Name ↑↓	Type ↑↓
a55e2ea5-02b7-4022-929c-f372f87bab8c	Public IP address
aks-agentpool-26729044-nsg	Network security group
aks-agentpool-26729044-routetable	Route table
aks-nodepool1-26729044-vmss	Virtual machine scale set
aks-vnet-26729044	Virtual network
kubernetes	Load balancer

Figura 7.3: Recursos creados por Azure para AKS

Entre los recursos creados, indicados en la figura 7.3, podemos identificar, principalmente:

- Un grupo de seguridad de red: Consiste en un cortafuegos sencillo que sólo deja traspasar el puerto 80, según lo indicado en el fichero de despliegue de Kubernetes (figura A.1 de la página 83).

- Un conjunto de máquinas virtuales escalables, que corresponden a los dos nodos que indicamos en la configuración inicial.

- Un balanceador de carga, también configurado en el documento de despliegue. (figura A.1 de la página 83)

Capítulo 8

Conclusiones

FINALIZADO este proyecto, se exponen los objetivos cumplidos, lo aprendido, y las posibles ampliaciones de este proyecto.

8.1 Valoración final

Se ha conseguido desarrollar la aplicación web, que puede utilizar cualquier usuario con una cuenta de Spotify y empezar a recibir recomendaciones. Se han creado distintos tipos de recomendaciones para maximizar la experiencia del usuario y se puede interactuar con las mismas para añadirlas a la cuenta de Spotify del usuario.

El sistema de recomendaciones es bueno, fiable y eficiente, ofrece recomendaciones en tiempo real y no ralentiza en exceso el funcionamiento de la página. Comparándolo con un sistema basado en k-NN, es hasta un 99,24% más rápida (Cuadro 6.2).

Se han desarrollado funcionalidades que permite que un usuario no registrado pueda navegar por la página, permitiéndole ver cómo funciona utilizando las listas de reproducción del propio conjunto de datos para obtener recomendaciones.

Se ha mantenido el motor de recomendaciones completamente separado de la aplicación web, por lo que podría sustituirse por otro completamente diferente mientras se expusiese la misma interfaz REST.

Se ha desplegado en un clúster Kubernetes en el cloud, lo que permite escalar la aplicación, e incluso facilita trasladarla a otro entorno de contenedores si fuese necesario, simplemente cambiando algunos valores de la configuración.

Uno de los principales problemas durante el desarrollo ha sido la gestión de la memoria RAM del equipo de desarrollo, ya que el proyecto consume alrededor de 5 GB de RAM en funcionamiento, por lo que sumando el sistema operativo y el entorno de desarrollo, complicó en algunas ocasiones trabajar de forma fluida.

8.2 Lecciones aprendidas

Se ha podido iniciar al *machine learning* de forma práctica, introduciendo algunos conceptos aún no vistos y otros vistos pero de forma exclusivamente teórica. Se han visto métodos de búsqueda de vecinos e implementado en concreto Approximate Neareast Neighbor.

Se ha aprendido a utilizar otros motores de base de datos NoSQL como Redis, y conectarlo con una aplicación en Python para hacer consultas clave-valor de manera rápida. Se ha aprendido a crear servicios REST en Python, utilizando Flask, y configurando las dependencias en un entorno virtual. También se ha aprendido a utilizar *pandas* para el tratamiento y transformación de datos de forma masiva.

También se ha aprendido a crear un contrato en *Swagger* a partir del cual generar un cliente de forma dinámica, integrándolo con Spring.

Se han visto conceptos de programación orientada a aspectos, y creado un aspecto para facilitar el desarrollo y no repetir código.

También se han vuelto a ver conceptos de interfaces de usuario web, y aprendido y ahondado un poco más en lenguajes como JavaScript y librerías como JQuery, algo esencial pero que se ha visto de forma muy simplificada durante el grado.

Por último, se ha aprendido a desarrollar una aplicación utilizando contenedores y a desplegar en un clúster de Kubernetes. Se ha aprendido a utilizar un entorno en la nube como el de Microsoft Azure y a aprovecharse de gran parte de sus servicios.

8.3 Líneas futuras

Un futuro posible desarrollo que se ha quedado fuera del alcance de este proyecto es aumentar dinámicamente las instancia del motor de recomendaciones que actualmente es de un millón de listas de reproducción.

A partir de los diferentes usuarios que utilizasen la página, para los que obtuvieran una peor puntuación de recomendaciones se podrían almacenar sus listas y añadirlas al conjunto de datos. Para ello, se tendrían que añadir los datos de la lista de reproducción y de las canciones que no tuviéramos registradas ya a la base de datos, se construiría una fila de nuestra matriz dispersa y se añadiría al conjunto de datos. Por último se tendría que recargar el árbol de búsqueda en segundo plano, por ejemplo en un proceso al final del día, e integrar pruebas automatizadas para estudiar el impacto en el rendimiento y la calidad de las recomendaciones. Si la evolución fuese tal que impactase el resultado del rendimiento, se podría seleccionar a partir de los *miss* y los *hits* en el almacén de Redis qué canciones y listas añadir o quitar del modelo, para quedarnos con un modelo que evolucionase según lo hiciesen los usuarios.

Además de lo explicado, habría que considerar un número máximo que se pueda ampliar

para evitar un consumo de memoria excesivo.

Otro posible desarrollo sería ampliarlo para soportar otros servicios de música como Google Play Music, para lo cual habría que establecer una relación entre los ids internos de las canciones de nuestra aplicación y un identificador de Google. Y desarrollar el servicio que se comunicaría con el API de Google Play en lugar de la de Spotify.

A nivel algorítmico, otra futura linea de desarrollo sería cambiar la representación de los objetos por representaciones densas (*embeddings*). Los modelos basados en memoria usando *embeddings* han demostrado una mejora en la calidad de recomendaciones y pueden contribuir a una reducción de las necesidades de memoria [41].

Apéndices

Apéndice A

Material adicional

Se muestran ejemplos de ficheros y cuadros detallados que no tenían cabida dentro del cuerpo principal de la memoria.

```
apiVersion: v1
kind: Service
metadata:
  name: web-jar
spec:
  loadBalancerIP: 20.50.209.148
  type: LoadBalancer
  ports:
    - name: "80"
      port: 80
      targetPort: 8080
  selector:
    app: web-jar
```

Listing A.1: Ejemplo de YAML de servicio en Kubernetes

```

apiVersion: apps/v1
kind: Deployment
metadata:
  name: redis
spec:
  replicas: 1
  strategy:
    type: Recreate
```

```
10    selector:
11      matchLabels:
12        app: redis
13    template:
14      metadata:
15        labels:
16          app: redis
17      spec:
18        containers:
19          - image: redis:alpine
20            name: redis
21            ports:
22              - containerPort: 6379
23            volumeMounts:
24              - mountPath: /data
25                name: redis-data
26        restartPolicy: Always
27        volumes:
28          - name: redis-data
29            azureFile:
30              secretName: azure-secret
31              shareName: redisdata
32              readOnly: false
```
Listing A.2: Ejemplo de YAML de despliegue en Kubernetes

k-clúster	Playlist	Top-k success	Top-2k success
5	Dinner with Friends	60,00%	100,00%
5	Atmospheric Piano	0,00%	40,00%
5	En el Trabajo	40,00%	40,00%
5	Lo-Fi Beats	0,00%	0,00%
5	Hit Alegres	20,00%	100,00%
5	Temazos Chill	40,00%	100,00%
5	Chillin' on a Dirt Road	40,00%	100,00%
5	Chopin Nocturnes	0,00%	0,00%
5	Classical Romance	60,00%	80,00%
5	Sinfonía Hipster	100,00%	100,00%
10	Dinner with Friends	60,00%	100,00%
10	Atmospheric Piano	0,00%	60,00%
10	En el Trabajo	40,00%	40,00%
10	Lo-Fi Beats	0,00%	0,00%
10	Hit Alegres	40,00%	100,00%
10	Temazos Chill	60,00%	100,00%
10	Chillin' on a Dirt Road	80,00%	100,00%
10	Chopin Nocturnes	0,00%	0,00%
10	Classical Romance	60,00%	80,00%
10	Sinfonía Hipster	100,00%	100,00%
20	Dinner with Friends	60,00%	100,00%
20	Atmospheric Piano	0,00%	60,00%
20	En el Trabajo	40,00%	80,00%
20	Lo-Fi Beats	0,00%	0,00%
20	Feel-Good Indie Rock	20,00%	20,00%
20	Hit Alegres	60,00%	100,00%
20	Temazos Chill	100,00%	100,00%
20	Chillin' on a Dirt Road	100,00%	100,00%
20	Chopin Nocturnes	0,00%	0,00%
20	Classical Romance	60,00%	80,00%
20	Sinfonía Hipster	100,00%	100,00%
40	Dinner with Friends	60,00%	100,00%
40	Atmospheric Piano	0,00%	60,00%
40	En el Trabajo	60,00%	100,00%
40	Lo-Fi Beats	0,00%	0,00%
40	Hit Alegres	100,00%	100,00%
40	Temazos Chill	100,00%	100,00%
40	Chillin' on a Dirt Road	100,00%	100,00%
40	Chopin Nocturnes	20,00%	20,00%
40	Classical Romance	60,00%	80,00%
40	Sinfonía Hipster	100,00%	100,00%

Cuadro A.1: Calidad a-NN - Desglose

Lista de acrónimos

a-NN Approximate Neareast Neighbor. 2, 3, 9, 25, 41, 63–65, 78

Ajax Asynchronous JavaScript And XML. 24, 58, 59

AKS Azure Kubernetes Service. 33, 72, 75, 76

AOP Aspect Oriented Programming. 49

API Application Programming Interface. 25, 27, 36, 38, 45, 46, 48–50, 52, 55, 79

CF Collaborative Filtering. 6, 8

CI Continuous Integration. 30

CSS Cascading Style Sheets. 24, 36

HTML HyperText Markup Language. 24, 25, 36

IaaS Infrastructure as a Service. 73, 76

IDE Integrated Development Environment. 29

IP Internet Protocol. 72, 75

JDBC Java Database Connectivity. 28, 36, 38, 61

JSON JavaScript Object Notation. 5, 26, 42

JVM Java Virtual Machine. 23

k-NN k-Neareast Neighbor. 2, 3, 7, 9, 64–66, 77

LTS Long Time Support. 23

MPD The Million Playlist Dataset. 5, 42

PaaS Platform as a Service. 73

POM Project Object Model. 31

REST Representational state transfer. v, 25, 27, 36, 38, 45–47, 50, 54, 68, 77, 78

SaaS Software as a Service. 73

SMB3 Server Message Block v3. 74

SQL Structured Query Language. 25, 43

URI Unified Resource Identifier. 89

VCS Version Control System. 30

W3C World Wide Web Consortium (W3C). 24

WSR Weighted Sum Recommender. 7, 8, 10, 40, 41, 46, 57

Glosario

backend Es la parte de desarrollo de una aplicación que no está expuesta al usuario, implica toda la lógica de negocio, la interacción con los diferentes sistemas, etc.. 23, 25–27

bytecode Código independiente de máquina que generan compiladores de determinados lenguajes (Java, Erlang,...) y que es ejecutado por su correspondiente intérprete.. 23

endpoint Son los servicios y recursos que expone una aplciación en formato de Unified Resource Identifier (URI)s. 28, 48, 61

framework Marco de desarrollo. Corresponde a un conjunto de herramientas, conceptos y metodologías que se llevan a cabo a la hora de desarrollar un proyecto dentro de una casuística habitual.. 3, 7, 23, 27, 28

frontend Es la parte de desarrollo de una aplicación que está expuesta al usuario, implica toda la lógica de las pantallas, la interfaz y el comportamiento en general con el usuario.. 24

interfaz responsive Corresponde a una interfaz web que se adapta a la pantalla el usuario. Utilizado sobre todo para elaborar interfaces que funcionen tanto en formato móvil como desde un escritorio.. 28

matriz dispersa Es una matriz de gran tamaño compuesta en su mayoría por ceros. Se representan teniendo en cuenta los puntos en los que hay datos para ahorrar memoria y recursos.. 8, 25, 26, 38, 45, 64

namespace Espacio de nombres. Es un conjunto de nombres donde los nombres definidos son únicos.. 25

stub Es un fragmento o conjunto de fragmentos de código temporal que sirven como plantilla o para imitar un funcionamiento todavía no implementado.. 27

Bibliografía

[1] S. G. Consumer, “¿en qué países se compran más productos en amazon?” https://es.statista.com/grafico/18609/uso-de-amazon-por-paises/, 2019, accedido el 16-06-2020.

[2] P. V. Kessel, “10 facts about americans and youtube,” https://www.pewresearch.org/fact-tank/2019/12/04/10-facts-about-americans-and-youtube/, 2019, accedido el 16-06-2020.

[3] N. Mohan, “Youtube’s ai is the puppet master over most of what you watch,” https://www.cnet.com/news/youtube-ces-2018-neal-mohan/, 2018, accedido el 16-06-2020.

[4] J. Mangalindan, “Amazon’s recommendation secret,” https://fortune.com/2012/07/30/amazons-recommendation-secret/, 2012, accedido el 16-06-2020.

[5] McKinsey&Company, “How retailers can keep up with consumers,” https://www.mckinsey.com/industries/retail/our-insights/how-retailers-can-keep-up-with-consumers, 2013, accedido el 16-06-2020.

[6] D. Valcarce, J. Parapar, and Álvaro Barreiro, “Language models for collaborative filtering neighbourhoods,” in *Advances in Information Retrieval - 38th European Conference on IR Research, ECIR 2016, Padua, Italy, March 20-23, 2016. Proceedings*, ser. Lecture Notes in Computer Science, N. Ferro, F. Crestani, M. Moens, J. Mothe, F. Silvestri, G. M. D. Nunzio, C. Hauff, and G. Silvello, Eds., vol. 9626. Springer, 2016, pp. 614–625. [Online]. Available: https://doi.org/10.1007/978-3-319-30671-1_45

[7] ——, “Finding and analysing good neighbourhoods to improve collaborative filtering,” *Knowl. Based Syst.*, vol. 159, pp. 193–202, 2018. [Online]. Available: https://doi.org/10.1016/j.knosys.2018.06.030

[8] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval.* Cambridge University Press, 2008.

[9] S. Arya and D. M. Mount, "Approximate nearest neighbor queries in fixed dimensions," in *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, ser. SODA '93. USA: Society for Industrial and Applied Mathematics, 1993, p. 271–280.

[10] C.-W. Chen, P. Lamere, M. Schedl, and H. Zamani, "Recsys challenge 2018: Automatic music playlist continuation," in *Proceedings of the 12th ACM Conference on Recommender Systems*, ser. RecSys '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 527–528. [Online]. Available: https://doi.org/10.1145/3240323.3240342

[11] A. Said, "A short history of the recsys challenge," *AI Magazine*, vol. 37, no. 4, pp. 102–104, Jan. 2017. [Online]. Available: https://aaai.org/ojs/index.php/aimagazine/article/view/2693

[12] F. Ricci, L. Rokach, and B. Shapira, *Introduction to Recommender Systems Handbook*. Boston, MA: Springer US, 2011, pp. 1–35. [Online]. Available: https://doi.org/10.1007/978-0-387-85820-3_1

[13] D. Valcarce, J. Parapar, and Álvaro Barreiro, "A mapreduce implementation of posterior probability clustering and relevance models for recommendation," *Engineering Applications of Artificial Intelligence*, vol. 75, pp. 114 – 124, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0952197618301696

[14] ——, "Efficient pseudo-relevance feedback methods for collaborative filtering recommendation," in *Advances in Information Retrieval*, N. Ferro, F. Crestani, M.-F. Moens, J. Mothe, F. Silvestri, G. M. D. Nunzio, C. Hauff, and G. Silvello, Eds. Cham: Springer International Publishing, 2016, pp. 602–613.

[15] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft, "When is nearest neighbor" meaningful?" *ICDT 1999. LNCS*, vol. 1540, 12 1997. [Online]. Available: https://www.researchgate.net/publication/2845566_When_Is_Nearest_Neighbor_Meaningful

[16] F. Chierichetti, A. Panconesi, P. Raghavan, M. Sozio, A. Tiberi, and E. Upfal, "Finding near neighbors through cluster pruning," in *Proceedings of the Twenty-Sixth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, ser. PODS '07. New York, NY, USA: Association for Computing Machinery, 2007, p. 103–112. [Online]. Available: https://doi.org/10.1145/1265530.1265545

[17] K. Schwaber, *Agile Project Management With Scrum*. USA: Microsoft Press, 2004.

[18] R. Pressman, *Software Engineering: A Practitioner's Approach*, 7th ed. USA: McGraw-Hill, Inc., 2009.

[19] Microsoft, "Predict costs and optimize spending for azure," https://docs.microsoft.com/en-us/learn/modules/predict-costs-and-optimize-spending, accedido el 16-06-2020.

[20] M. Lutz, *Learning Python*, 4th ed. O'Reilly, 2014.

[21] M. Pilgrim, *HTML5 Up and running*. O'Reilly, 2010.

[22] J. D. Gauchat, *El gran libro de HTML5, CSS3 y Javascript*, 1st ed. Marcombo, 2012.

[23] D. Flanagan, *JavaScript: The Definitive Guide*, 6th ed. O'Reilly, 2006.

[24] J. Bloch, *Effective Java*, 3rd ed. Addison-Wesley Professional, 2017.

[25] S. Beecher, "Approximate nearest neighbor search for sparse data in python!" https://github.com/facebookresearch/pysparnn, accedido el 16-06-2020.

[26] N. Community, "Numpy user guide," https://numpy.org/doc/stable/user/index.html, 2020, accedido el 16-06-2020.

[27] C. Walls, *Spring Boot in Action*, 4th ed. Manning, 2014.

[28] T. Agile, "Learn about taiga," https://taiga.pm/learn-about-taiga-2/, accedido el 16-06-2020.

[29] JetBrains., https://www.jetbrains.com/, accedido el 16-06-2020.

[30] A. S. Fundation, https://www.apache.org, accedido el 16-06-2020.

[31] D. Community, "Docker engine overview," https://docs.docker.com/engine/, accedido el 16-06-2020.

[32] ——, "Overview of docker compose," https://docs.docker.com/compose/, accedido el 16-06-2020.

[33] ——, "Docker hub quickstart," https://docs.docker.com/docker-hub/, accedido el 16-06-2020.

[34] D. Valcarce, J. Parapar, and Álvaro Barreiro, "A distributed recommendation platform for big data," *J. UCS*, vol. 21, no. 13, pp. 1810–1829, 2015. [Online]. Available: http://www.jucs.org/jucs_21_13/a_distributed_recommendation_platform

[35] A. Parecki, *OAuth 2.0 Simplified*. Lulu.com, 2018.

[36] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[37] D. Valcarce, A. Bellogín, J. Parapar, and P. Castells, "On the robustness and discriminative power of information retrieval metrics for top-n recommendation," in *Proceedings of the 12th ACM Conference on Recommender Systems*, ser. RecSys '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 260–268. [Online]. Available: https://doi.org/10.1145/3240323.3240347

[38] ——, "Assessing ranking metrics in top-n recommendation," *Information Retrieval Journal*, 2020. [Online]. Available: https://doi.org/10.1007/s10791-020-09377-x

[39] D. Community, "docker-library/openjdk," https://github.com/docker-library/openjdk, accedido el 16-06-2020.

[40] S. Ramírez, "uwsgi-nginx-flask," https://github.com/tiangolo/uwsgi-nginx-flask-docker, accedido el 16-06-2020.

[41] D. Valcarce, J. Parapar, and Álvaro Barreiro, "Collaborative filtering embeddings for memory-based recommender systems," *Eng. Appl. Artif. Intell.*, vol. 85, pp. 347–356, 2019. [Online]. Available: https://doi.org/10.1016/j.engappai.2019.06.020