Universidad Autónoma de Madrid

Escuela Politécnica Superior

23|24

Trabajo fin de máster

Desarrollo de una aplicación de recomendación de música para grupos

Miguel García González

Escuela Politécnica Superior
Universidad Autónoma de Madrid
C\Francisco Tomás y Valiente nº 11

Campus Internacional excelencia UAM CSIC+

www.uam.es

UNIVERSIDAD AUTÓNOMA DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

Máster Universitario en Ingeniería Informática

TRABAJO FIN DE MÁSTER

Desarrollo de una aplicación de recomendación de música para grupos

Autor: Miguel García González
Tutor: Alejandro Belllogín Kouki

enero 2024

Miguel García González
Desarrollo de una aplicación de recomendación de música para grupos

Miguel García González
C\ Ana María Nº 51

IMPRESO EN ESPAÑA – PRINTED IN SPAIN

A mis padres, por su apoyo incondicional.

Quien recibe un beneficio nunca debe olvidarlo;
quien lo otorga, nunca debe recordarlo.

Pierre Charron

Prefacio

Miguel García González

Agradecimientos

En primer lugar, quiero agradecer a mi familia, en especial a mis padres, todo el apoyo que me brindan día a día de manera incondicional. Las oportunidades que me han dado, como la de irme a estudiar al extranjero durante este máster, me han dado momentos, experiencias y aprendizaje que no olvidaré. A mis hermanos quiero agradecerles su experiencia y consejo, los cuales en ciertas ocasiones me han ayudado a tomar decisiones importantes, que me han llevado hasta aquí. Su reciente experiencia en estas etapas de la vida ha sido siempre y será un apoyo imposible de ignorar.

Quiero dar las gracias a mis amigos, por ser el círculo de influencia que son, porque me ayudan a seguir motivado y concentrado en lo que hay que hacer, y porque me han dado momentos de diversión y desconexión que me han ayudado a seguir adelante.

Quiero dar las gracias también por los compañeros del máster, con los que he aprendido mucho y he compartido muchas horas. Al final, el máster te brinda la experiencia de vivir la universidad de una forma bastante diferente al grado en mi opinión, y doy gracias por haberla vivido. Quiero dar las gracias a en general a todas las personas en las cuales consigo encontrar motivación, ya sea por su ejemplo, por su experiencia o por su consejo.

Por último, quiero dar las gracias a Alejandro Bellogín, mi tutor, por haberme dado la oportunidad de trabajar en este proyecto, por su apoyo, disponibilidad absoluta y por sus propuestas y guía durante la realización del trabajo.

Resumen

Que la tecnología inunda nuestras vidas tras haber progresado de manera vertiginosa es una obviedad. La música no es una excepción, y la aparición de plataformas de *streaming* como *Spotify* ha cambiado la forma en la que escuchamos música. La obtención de datos de escucha de los usuarios permite que se puedan perfilar sus gustos de música, dejando de estar todo el peso de la elección en el usuario, que antes tendría que buscar manualmente la música que él previamente tendría que haber descubierto que le gustase; a estar una gran parte del peso en el sistema, que recomendará la música que más se ajuste a los gustos registrados del usuario. A día de hoy *Spotify* traduce esta recomendación en listas de reproducción (*playlists*) generadas automáticamente y que se actualizan automáticamente. Los usuarios han pasado a tener un papel totalmente pasivo en estas recomendaciones, y aunque no se quiere poner en duda su atractivo o utilidad y eficacia, en este trabajo se quiere explorar una alternativa intermedia; en la cual, usuario y sistema tengan un papel en la generación de recomendaciones.

Para ello, se expondrá en este documento el desarrollo de una aplicación web que permite a los usuarios de *Spotify* generar listas de reproducción (*playlists*) de música que se basen en los gustos de un grupo de usuarios de *Spotify*. Se utilizará la API de *Spotify* para obtener datos de usuarios, canciones más escuchadas, y recomendaciones. Como *framework* de desarrollo se utilizará *Flutter* (desarrollado por *Google*), que permite desarrollar aplicaciones multiplataforma, con una misma base de código. En este caso se hará uso de la versión web de la aplicación.

Se llevarán a cabo pruebas con usuarios para evaluar la aplicación y distintas formas de generar *playlists*. Tras ello, se analizarán los resultados obtenidos para obtener *feedback*, conclusiones y posible trabajo futuro sobre el trabajo realizado.

Palabras clave

Flutter, *Spotify*, recomendación, recomendación de grupos, estrategias de agregación

Abstract

That technology inundates our lives after having progressed rapidly is an obvious fact. Music is no exception, and the emergence of streaming platforms like Spotify has changed the way we listen to music. Obtaining listening data from users allows their music preferences to be profiled, shifting the weight of choice from the user, who previously had to manually search for music they would have previously discovered they liked, to the system, which recommends music that best matches the user's registered preferences. Currently, Spotify translates this recommendation into automatically generated and updated playlists. Users have become completely passive in these recommendations, and although their appeal, usefulness, and effectiveness are not questioned, this work aims to explore an intermediate alternative; one in which the user and the system play a role in generating recommendations.

To this end, this document presents the development of a web application that allows Spotify users to generate playlists based on the preferences of a group of Spotify users. The Spotify API will be used to obtain user data, most listened songs, and recommendations. The development framework used will be Flutter (developed by Google), which allows for cross-platform application development with a single codebase. In this case, the web version of the application will be used.

Tests will be conducted with users to evaluate the application and different ways of generating playlists. Afterwards, the obtained results will be analyzed to gather feedback, draw conclusions, and identify possible future work based on the work done.

Keywords

Flutter, Spotify, recommendation, group recommendation, aggregation strategies

Índice

Listas

Lista de algoritmos

Lista de códigos

Lista de cuadros

Lista de ecuaciones

Lista de figuras

Lista de tablas

Lista de cuadros

1

Introducción

En este capítulo se expondrán la motivación que ha hecho que este trabajo se lleve a cabo, los obejtivos que se proponen lograr con él, y finalmente, se explicará la estructura que seguirá este documento.

1.1. Motivación del proyecto

La motivación de este trabajo viene por dos vertientes principales. Por un lado, *Flutter* [1] es un *framework* de desarrollo de aplicaciones multiplataforma bastante nuevo, desarrollado por *Google* que ha ganado cierta popularidad y que ha llamado nuestra atención. Así que hacer un proyecto con él es una buena forma de ponerlo a prueba y ver qué tal funciona.

Por el otro lado, escuchar música forma parte de nuestro día a día, de manera directa, a través de la aplicación de *Spotify* [2]. Añadido a esto, el autor siente o percibe que hoy en día en muchas aplicaciones, productos o servicios, el usuario tienen un papel muy pasivo, y que en el caso de *Spotify* esto se ha acentuado con el paso del tiempo. Los usuarios reciben distintos tipos de *playlists* generadas periódicamente: novedades de la semana, fusionadas con otros usuarios que se actualizan semanalmente, predeterminadas para estados de ánimo, etc. Por supuesto, los usuarios pueden crear sus propias *playlists*, pero la forma de hacerlo es bastante manual, y desde luego la adición de canciones se hace de una en una.

Por ello, se quiere explorar una alternativa intermedia, en la cual, usuario y sistema comparten el peso de la creación de las *playlists*. El usuario podrá juntar a varios usuarios a la vez para crear una *playlist* que trate de ajustarse a los gustos de todos. Incluso, generar una *playlist* que se base en sus, por ejemplo, tres canciones más escuchadas recientemente. La cosa es ofrecer al usuario, y sobre todo a aquel que le guste cobrar un papel más activo en sus elecciones, una forma fácil de ofrecer canciones a escuchar.

Finalmente, en este trabajo entra en escena la recomendación, y más específicamente, la dirigida a grupos, ya que se buscará también hacer un pequeño estudio sobre las distintas formas de generar *playlists* a partir de los gustos de un grupo de usuarios. Se utilizará la API de *Spotify* [3] para obtener

datos de usuarios, canciones más escuchadas, y recomendaciones, que aportarán la base para la generación de *playlists*.

1.2. Propuesta y objetivos

Para este trabajo se tiene como objetivo principal el desarrollo de una aplicación web que permita a grupos de usuarios de *Spotify* crear (*playlists*) que combinen sus gustos musicales, ofreciendo una alternativa a las *playlists* que la aplicación de *Spotify* les pueda ofrecer. También se querrá llevar a cabo el estudio y posterior comparación de distintas estrategias de agregación para la generación de *playlists*.

Derivados de este objetivo principal, se plantean los siguientes objetivos específicos:

- Estudiar sobre el desarrollo de aplicaciones con *Flutter*, y familiarizarse con la base de su desarrollo, los *widgets*.
- Estudiar la API de *Spotify*, leyendo su documentación y haciendo pruebas, para acabar desarrollando el módulo de llamadas a la API.
- Desarrollar la aplicación web, ejecutada en el lado del cliente, que permita a grupos de usuarios de *Spotify* crear *playlists* que combinen sus gustos musicales.
- Realizar un pequeño estudio sobre la aplicación desarrollada, llevando a cabo pruebas con usuarios reales, para obtener conclusiones sobre la usabilidad de la aplicación y las potenciales estrategias para la generación de *playlists*.

1.3. Estructura del documento

ESTRUCTURA DEL DOCUMENTO

2

Estado del arte

En este capítulo se hará un recorrido por los actores principales de este trabajo: *Flutter*, la API de *Spotify*, y la recomendación dirigida a grupos. Se explicará qué son, cómo funcionan, y una aproximación de cómo hemos aplicado estos conceptos en este trabajo. Estos componentes son los que han conformado el proyecto de aplicación que finalmente se ha desarrollado, y que se explicará en el capítulo 3.

2.1. *Framework* de desarrollo de aplicaciones: *Flutter*

Flutter [1] es un *framework* de desarrollo de código abierto de aplicaciones multiplataforma desarrollado por *Google*. Es una alternativa que tiene cierta popularidad, y que, como comentábamos en la sección 1.1 ha llamado nuestra atención.

Con este *framework*, se pueden desarrollar aplicaciones para diversas plataformas, con una misma base de código. Desde aplicaciones móviles en *iOS* y *Android*, hasta aplicaciones de escritorio para *Windows*, *Linux* o *MacOS*. Sin olvidarnos de las aplicaciones *Web* (como la aplicación desarrollada en este trabajo), o incluso dispositivos embebidos, como los que pueden ser el infoentretenimiento de un coche [4].

Aunque solamente este hecho aporta un atractivo muy considerable a esta tecnología, *Flutter* cuenta con otras características a tener en cuenta:

- **Rendimiento:** utiliza el lenguaje de programación *Dart* [5], que se compila a código nativo, lo que aporta una ventaja en rendimiento clara. Comentaremos más sobre las características de este lenguaje en la sección 2.1.1.
- **Productividad:** cuenta con *Hot Reload* [6], que permite ver los cambios en la aplicación en tiempo real, sin necesidad de reiniciar la aplicación, y sin perder el estado de esta, agilizando mucho el desarrollo.
- **Personalización:** su diseño de la interfaz basada en *widgets* personalizables, que permiten crear interfaces de usuario atractivas y consistentes.
- **Comunidad:** existe una comunidad con buena actividad, la cual hace que prolifere el desarrollo de *plugins* que se pueden encontrar en *pub.dev* [7].

2.1.1. *Dart*, el lenguaje detrás de *Flutter*

Flutter se basa en el lenguaje de programación *Dart* [5], también desarrollado por *Google*. Aunque este es un lenguaje relativamente nuevo, y con un uso que no está tan extendido como otros lenguajes, como *Java*, *Python* o *JavaScript*, podemos destacar varias características de este lenguaje que lo hacen atractivo. Por otro lado, haberse familiarizado con este lenguaje no ha sido un problema para el desarrollo en este trabajo, debido a una sintaxis que recuerda a otros lenguajes como *Java*, *C++* o *JavaScript*. Algunas características para una visión general de *Dart* [8] son:

- **Tipado estático:** es un lenguaje de tipado estático, lo que significa que una vez las variables se declaran de un tipo, este no puede cambiar durante la ejecución del programa.
- **Tipado dinámico:** aunque sea un lenguaje de tipado estático, también permite el uso del tipo `dynamic`, que permite flexibilidad en el tipo que deba tener una variable.
- **Programación orientada a objetos:** es un lenguaje orientado a objetos, lo que lo hace muy amigable si se tiene experiencia con otros lenguajes orientados a objetos.
- **Seguridad frente a nulos:** tiene un sistema de seguridad frente a nulos, que permite evitar que variables puedan tener un valor nulo en cualquier momento en tiempo de ejecución.
- **Asincronía:** permite el uso de funciones asíncronas, que permiten ejecutar código en segundo plano, sin bloquear la ejecución del programa.
- **Compilación a código nativo:** se compila a código nativo, lo que aporta un rendimiento muy bueno.
- **Compilación nativa:** permite la compilación *JIT* (*just-in-time*) y *AOT* (*ahead-of-time*), que permite producir código máquina para plataformas nativas.
- **Compilación *Web*:** para aplicaciones *Web*, se compila a código *JavaScript*, que es lo que ocurre con la aplicación desarrollada en este trabajo.

Dart es un lenguaje optimizado para el cliente que permite desarrollar aplicaciones rápidas en cualquier plataforma. Su objetivo es ofrecer el lenguaje de programación más productivo para el desarrollo multiplataforma, junto con una plataforma de ejecución flexible para marcos de aplicaciones. Dart también constituye la base de Flutter. Dart proporciona el lenguaje y los tiempos de ejecución que impulsan las aplicaciones de Flutter, pero Dart también admite muchas tareas básicas del desarrollador, como formatear, analizar y probar el código [8].

2.1.2. Instalación y primer uso básico

Para poder desarrollar con *Flutter*, es necesario instalar el *SDK* de *Flutter*, que se explica en [9] para diferentes plataformas.

Una vez instalado, se puede comprobar que todo funciona correctamente ejecutando el comando `flutter doctor`. Para crear un proyecto, se ejecuta el comando `flutter create <nombre_proyecto>`. Una vez creado, tras situarse en el directorio del nuevo proyecto, se puede ejecutar con el comando `flutter run`. Ejecutando el comando `flutter devices` se puede comprobar qué dispositivos

están conectados y disponibles para la ejecución. En la figura 2.1 se puede ver un ejemplo de ejecución de un proyecto *Flutter*.

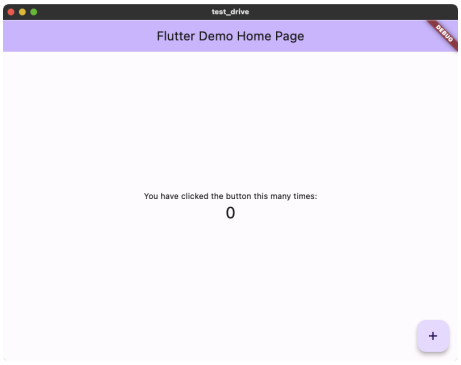


Figura 2.1: Ejecución de un proyecto *Flutter*
Imagen extraída de flutter.dev

2.1.3. *Widgets*

Los *widgets* son el elemento básico de la interfaz de usuario en *Flutter*. Todo lo que se ve en pantalla es un *widget*, y estos se combinan para crear elementos gráficos más complejos; pero siempre hay que tener en mente que todo está formado por *widgets*, los cuales se organizan en forma de árbol. En la figura 2.2 se puede ver un ejemplo de vista de árbol de *widgets*.

Figura 2.2: Ejemplo de árbol de *widgets*

Según [10], los widgets de Flutter se construyen utilizando un marco moderno que se inspira en *React* [11]. La idea central es construir la interfaz de usuario a partir de widgets. Los widgets describen el aspecto que debería tener su vista en función de su configuración y estado actuales. Cuando cambia el estado de un widget, este reconstruye su descripción, que el framework compara con la descripción anterior para determinar los cambios mínimos necesarios en el árbol de renderizado subyacente para

pasar de un estado al siguiente.

Existen dos tipos de *widgets*: los *StatelessWidget* y los *StatefulWidget*. Los primeros son aquellos que no contemplan cambios en su estado porque no tienen; es decir, no almacenan un estado que pueda variar. Mientras que los segundos sí almacenan un estado que pueda variar, y por ejemplo, pueden cambiar su aspecto en función de este estado.

Para hacer ver esto de una forma más cercana a lo técnico y al código, vamos a aclarar un par de cosas. Cuando se crea un *widget*, se define una clase que hereda de *StatelessWidget* o de *StatefulWidget*, y se implementa el método `build()`. Este método es el que se encarga de devolver un *widget* (que, como ya hemos comentado, puede estar formado por numerosos *widgets* estructurados en un árbol) que será lo que se muestre en la interfaz de usuario.

En el caso de los *StatefulWidget*, debe implementarse el método `createState()`, que devolverá un objeto que hereda de *State*. Este será el que se encargue de almacenar el estado del *widget*, y actualizarlo cuando se necesite. En el caso de un *StatefulWidget*, el método `build()` puede implementarse tanto en el *widget* como en el *state*, pero es recomendable hacerlo en el *state*. Veamos la comparación en código de un *StatelessWidget* y un *StatefulWidget* en la figura 2.3.

```dart
class MiStatelessWidget extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Container(
      // Aquí va el contenido del widget
    );
  }
}
```

(a) *StatelessWidget*

```dart
class MiStatefulWidget extends StatefulWidget {
  @override
  _MiStatefulWidgetState createState() => _MiStatefulWidgetState();
}

class _MiStatefulWidgetState extends State<MiStatefulWidget> {
  @override
  Widget build(BuildContext context) {
    return Container(
      // Aquí va el contenido del widget
    );
  }
}
```

(b) *StatefulWidget*

Figura 2.3: Implementación en código de un *StatelessWidget* y un *StatefulWidget*

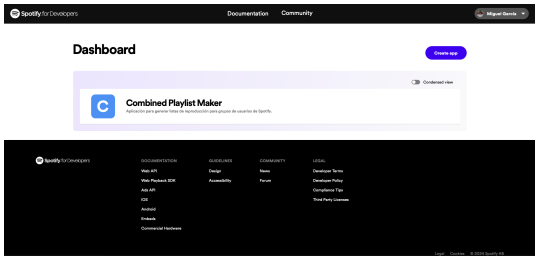
2.2. API Web de Spotify

En esta sección realizaremos una visión general de la API de *Spotify* [3], e iremos pasando por los *endpoints* más destacados, que serán útiles para el desarrollo de la aplicación. No obstante, en el capítulo 3 se ahondará más en el uso de cada uno de ellos.

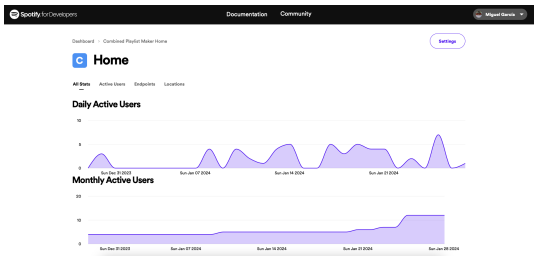
2.2.1. Visión general de la API

La API de *Spotify* [3] es una API *REST*, que utiliza el formato *JSON* para el intercambio de datos. Es importante destacar que para hacer uso de esta API, es necesario darse de alta como desarrollador,

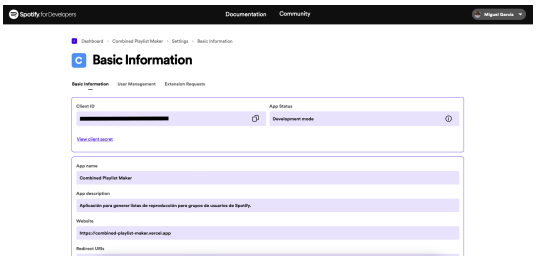
asumiendo que se tiene una cuenta de *Spotify*. Una vez hecho esto, se podrán dar de alta aplicaciones en la cuenta de desarrollador desde el *Dashboard*, y tener así los datos y credenciales necesarios para hacer uso de la API. En el *Dashboard* se pueden ver las aplicaciones dadas de alta, así como crear nuevas. Además, se puede entrar en cada aplicación para ver datos de utilización de *endpoints* y editar ajustes sobre ella, como la dirección del sitio web, las posibles *redirect URIs* para el flujo de autenticación, etc. Obsérvese la figura 2.4 a modo de orientación.



(a) *Dashboard*



(b) Información aplicación



(c) Editar ajustes aplicación

Figura 2.4: Visión general *Spotify for Developers*

Imágenes extraídas de developer.spotify.com

Para hacerse una idea inicial de qué tipos de *endpoints* nos harán falta, se puede hacer una primera aproximación a la aplicación que se quiere desarrollar. Vamos a suponer que un grupo de usuarios de *Spotify* está reunido e interesado en crear una playlist común que combine sus gustos musicales para escuchar en ese momento (o en un futuro). Teniendo en cuenta esta situación, y el resultado que se quiere obtener, podemos listar qué tipos de *endpoints* o interacciones con la API nos harán falta:

- **Datos de cuenta de los usuarios**, es decir, necesitaremos que los usuarios hagan *login* con sus cuentas de *Spotify*
- **Canciones más escuchadas** porque es necesario saber sobre los gustos de los usuarios y, obtener las canciones más escuchadas de cada uno de los usuarios que forman el grupo, parece una buena forma de hacerlo.
- **Recomendaciones**, porque así, podremos aprovecharnos de las estrategias de recomendación individuales de *Spotify*, y obtener recomendaciones de canciones que puedan gustar a los usuarios que forman el grupo, para así generar la playlist común.
- **Crear playlist**, porque así los usuarios podrán guardar en sus cuentas de *Spotify* la playlist que se haya generado.

2.2.2. Datos de cuenta de los usuarios

Para que los usuarios puedan autenticarse aplicaciones con sus cuentas de *Spotify*, y así poder crear playlists combinadas con los usuarios de su grupo, se llevará a cabo el flujo de autenticación PKCE (*Proof Key for Code Exchange*) definido en [12], el cual es el recomendado por *Spotify* para aplicaciones móviles u aplicaciones web de una sola página, donde la clave secreta del cliente no puede ser protegida de manera segura. Se puede ver un gráfico del flujo en la figura 2.5.

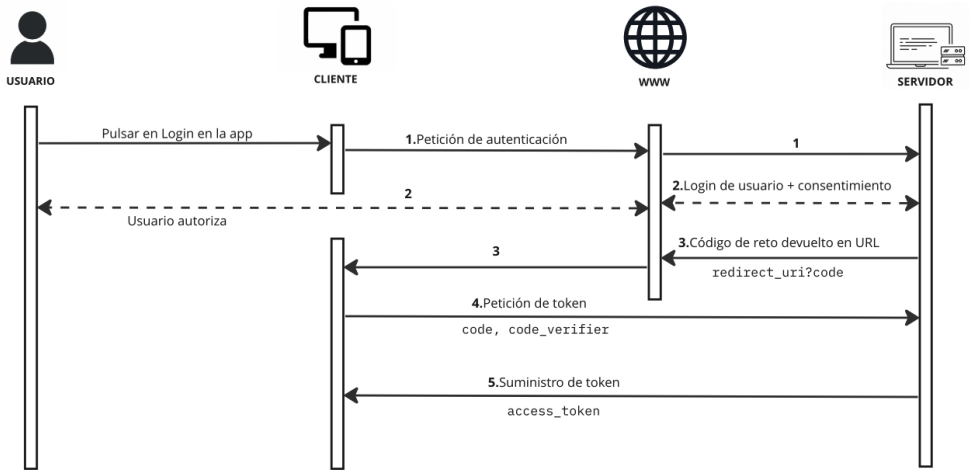


Figura 2.5: Flujo de autenticación PKCE

Para llevar a cabo esta autenticación, la app tendrá que lanzar la URL de autenticación de *Spotify* ("https://accounts.spotify.com/authorize"). Para más detalles sobre cómo se lleva a cabo este flujo en nuestra app, se tratará en el capítulo 3.

2.2.3. Canciones más escuchadas

Una vez se lleva a cabo la autenticación del usuario con su cuenta de *Spotify*, ya podemos hacer llamadas a la API gracias al token de acceso que hemos obtenido.

Comenzando por las canciones más escuchadas por el usuario, descubrimos que hay una *endpoint* que nos puede devolver los datos que necesitamos. En la documentación de la API [3] está definido como *Get User's Top Items* [13]. Y es que, resulta que podremos obtener tanto las canciones más escuchadas, como los artistas más escuchados. Veremos cómo haremos uso de este *endpoint* para la implementación de nuestra app en el capítulo 3.

Como se puede observar en la figura 2.6, `GET` es el método a utilizar y se especifican los siguientes parámetros:

- `type`: para indicar si queremos obtener artistas o canciones.

- `time_range`: el período de tiempo sobre el que queremos obtener la lista *items*.
- `limit`: el número de *items* que querríamos obtener.
- `offset`: por si quisieramos obtener la lista a partir de una determinada posición.

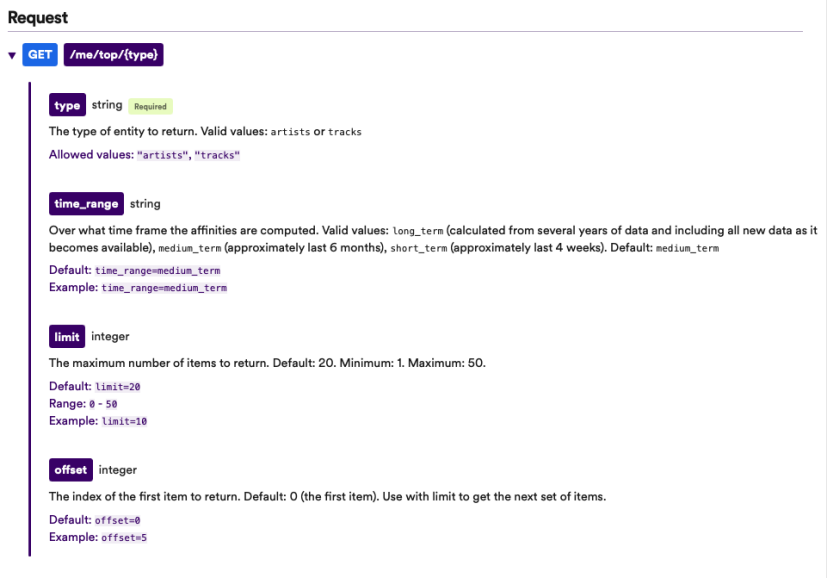


Figura 2.6: Obtener canciones o artistas más escuchados

Imagen extraída de developer.spotify.com

2.2.4. Recomendaciones

Para obtener recomendaciones de canciones, *Spotify* nos ofrece un *endpoint* llamado *Get Recommendations* [14]. En el capítulo 3 ahondaremos más en detalle sobre cómo se ha hecho uso de este *endpoint* para la implementación de la app. En la figura 2.7, además de que `GET` es el método a utilizar, se pueden observar cinco parámetros que se pueden especificar:

- `limit`: el número de canciones que se quieren obtener.
- `market`: el mercado en el que deben estar presentes las canciones recomendadas, si se quisiera restringir.
- `seed_artists`: una lista de identificadores de artistas, a partir de los cuales, *Spotify* obtendrá canciones para recomendar.
- `seed_genres`: una lista de géneros, que funciona igual que la anterior.
- `seed_tracks`: una lista de identificadores de canciones, que funciona igual que las anteriores.

Para este *endpoint*, existen más parámetros opcionales que se pueden especificar, relacionados con la la musicalidad, la energía, el tempo, la popularidad de las canciones, etc. Realmente tienen mucho potencial para la obtención de recomendaciones, pero no se han utilizado en la aplicación desarrollada. No obstante, está claro que deben tenerse en cuenta para trabajo futuro, y lo comentaremos en el capítulo 5.

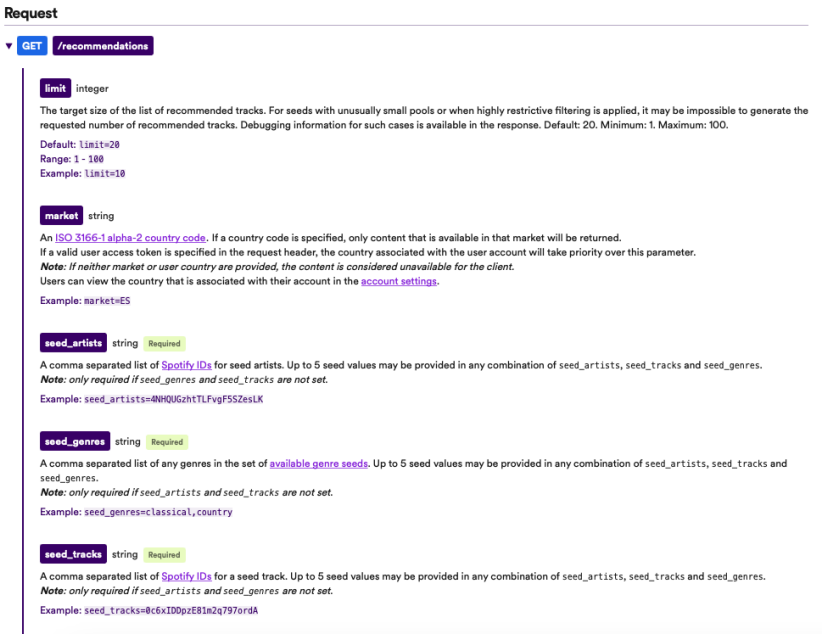


Figura 2.7: Obtener recomendaciones

Imagen extraída de developer.spotify.com

2.2.5. Crear playlist

Para crear una playlist, *Spotify* nos ofrece un *endpoint* llamado *Create Playlist* [15]. Como con los anteriores, ahondaremos más en cómo es utilizado en el capítulo 3. En la figura 2.8, además de `POST` es el método a utilizar, se pueden observar los parámetros a especificar en el cuerpo de la petición:

- `name`: el nombre de la playlist.
- `public`: si la playlist es pública o no.
- `collaborative`: si la playlist es colaborativa o no.
- `description`: la descripción de la playlist.

Con este *endpoint* se lleva a cabo la creación de la *playlist*, que se guardará en la cuenta de *Spotify* del usuario que se elija. Pero aún faltaría añadir las canciones a la *playlist* recién creada. Para ello, *Spotify* nos ofrece un *endpoint* llamado *Add Items to Playlist* [16]. En la figura 2.9, además de observar que `POST` es el método a utilizar, se pueden observar de nuevo los parámetros a especificar en el cuerpo de la petición:

- `playlist_id`: el identificador de la *playlist* a la que se quieren añadir las canciones.
- `position`: la posición en la que se quieren añadir las canciones.
- `uris`: una lista de identificadores de canciones, que se añadirán a la *playlist*.

Debe observarse además, que las `uris` se pueden pasar en la URL de la petición si no son demasiadas; si no, se recomienda pasarlas en el cuerpo de la petición.

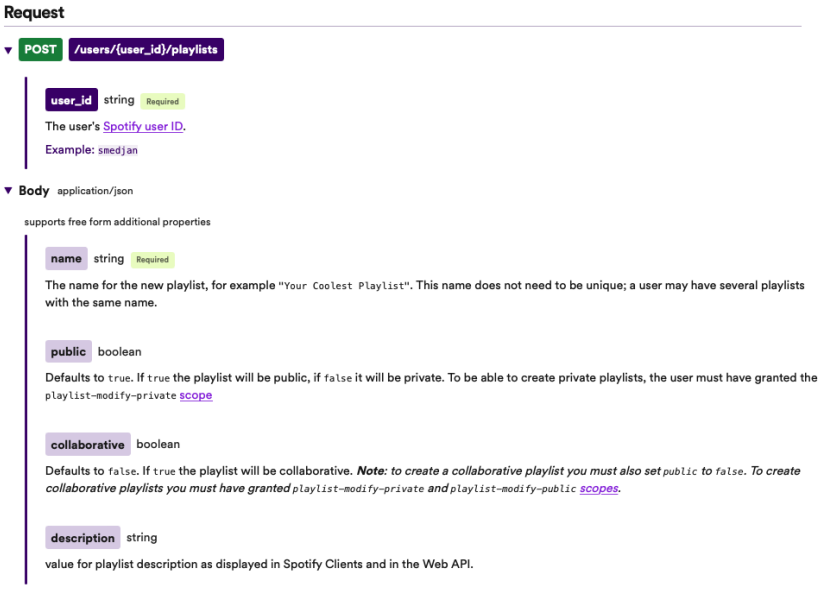


Figura 2.8: Crear playlist

Imagen extraída de developer.spotify.com

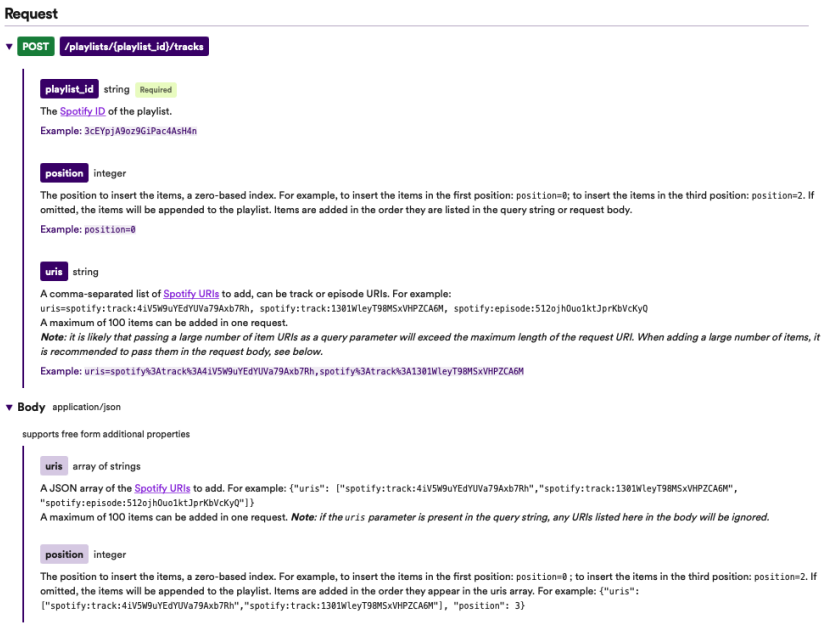


Figura 2.9: Añadir canciones a *playlist*

Imagen extraída de developer.spotify.com

2.3. Recomendación

Como introducción a esta sección, un fragmento del prefacio del libro *Recommender Systems Handbook* [17] traducido por nosotros al español aporta una visión bastante actual de qué debe venirnos a la cabeza cuando hablamos de sistemas de recomendación:

"Los sistemas de recomendación son herramientas y técnicas de software que ofrecen sugerencias de artículos útiles para un usuario. Las sugerencias de un sistema de recomendación están pensadas para ayudar al usuario a tomar decisiones como qué comprar, qué música escuchar o qué noticias leer. Los sistemas de recomendación son un valioso medio para que los usuarios de Internet hagan frente a la sobrecarga de información y les ayuden a elegir mejor. En la actualidad son una de las aplicaciones más populares de la inteligencia artificial, ya que ayudan a descubrir información en la Red. Se han propuesto varias técnicas de generación de recomendaciones y, en las dos últimas décadas, muchas de ellas se han implantado con éxito en entornos comerciales. Hoy en día, todos los grandes actores de Internet adoptan técnicas de recomendación. El desarrollo de sistemas de recomendación es un esfuerzo multidisciplinar en el que participan expertos de diversos campos, como la inteligencia artificial, la interacción persona-ordenador, la minería de datos, la estadística, los sistemas de apoyo a la decisión, el marketing y el comportamiento del consumidor."

El estudio sobre sistemas de recomendación es un campo bastante más nuevo que otras técnicas y herramientas clásicas de los sistemas de información, como las bases de datos, o los motores de búsqueda. A pesar de ser un área de estudio independiente relativamente joven, el interés en estos sistemas ha crecido de manera muy significativa. Simplemente, un claro signo de ello es cómo se hace uso de los sistemas de recomendación en la actualidad, como por ejemplo en las plataformas de *streaming* de música o vídeo (como *Spotify* o *Netflix*), o en las plataformas de comercio electrónico (como *Amazon* o *eBay*). Reflexionemos sobre el motivo de su extendido uso.

Partes interesadas en un sistema de recomendación

Según [18], en un sistema de recomendación influyen tres partes interesadas: los usuarios (o consumidores), los proveedores (o suministradores) y los propietarios del sistema. Los usuarios son los que reciben las recomendaciones (basándonos en este trabajo, digamos que los usuarios de *Spotify*), los proveedores son los que ofrecen los productos o servicios que se recomiendan (digamos que los artistas que publican su música en *Spotify*), y el propietario del sistema (en este caso, *Spotify*) que ofrece la plataforma en la que los usuarios son acercados a los proveedores (artistas). Sabiendo esto, podemos ver que las tres partes se ven beneficiadas ante un escenario ideal: los usuarios reciben recomendaciones de productos o servicios que les interesan (descubren nuevas canciones, géneros y artistas que les gustan), los proveedores ven aumentada su visibilidad (los artistas pueden volverse más conocidos) y, por tanto, sus ventas, y el propietario del sistema ve aumentado su número de usuarios y, por tanto, sus ingresos (más suscripciones de *Spotify*). Este beneficio claro para las tres partes

diríamos que es claramente uno de los mayores motivos por los que los sistemas de recomendación son tan populares y utilizados en la actualidad.

Fuentes de datos y conocimiento en un sistema de recomendación

Por otro lado, no podemos dejar de lado la suma importancia que tienen los datos de los que hace uso un sistema de recomendación. En [18], se destacan tres elementos como las fuentes de conocimiento y datos de las que un sistema de recomendación se nutre: los usuarios, los productos (o ítems) y las interacciones entre los dos anteriores. Los **ítems**, que son los objetos de las recomendaciones, deben contar con una descripción detallada y exhaustiva, abarcando desde características generales hasta metadatos específicos. Algoritmos especializados extraen y analizan estas características, los cuales son especialmente necesarios y útiles para ítems complejos como imágenes o textos, utilizando técnicas avanzadas de procesamiento de imágenes y lenguaje natural. En cuanto a los **usuarios**, cada usuario se modela de manera única, integrando una diversidad de factores como intereses, edad y género, entre otros. Pero más allá de las preferencias que se puedan mostrar explícitamente (como que un usuario indique sus géneros de música favoritos de manera activa), el modelo de un usuario también se enriquece con información contextual, permitiendo así que el sistema de recomendación ajuste sus sugerencias a las circunstancias específicas y momentáneas del usuario (como un descubrimiento de un nuevo artista o género que está escuchando recientemente). Por último, las **interacciones** entre usuarios e ítems son la fuente de datos más importante para un sistema de recomendación. Desde calificaciones explícitas hasta comportamientos implícitos de navegación, cada acción del usuario aporta información valiosa, contribuyendo a la comprensión profunda de sus preferencias y comportamientos.

2.3.1. Métodos y técnicas de recomendación

Filtrado basado en contenido

Filtrado colaborativo

Sistemas híbridos

2.3.2. Tipos de sistemas de recomendación

Recomendación de música

Recomendación a grupos

2.3.3. Estrategias de agregación

3

Desarrollo

Este capítulo hará un recorrido por el proceso de desarrollo de la aplicación, tratando de explicar y clarificar los detalles de implementación y de interacción con el usuario más importantes.

- mencionar implementación de autenticacion, porque digo que se explica en este capítulo
- mencionar implementación de llamadas a la API, porque digo que se explica en este capítulo
- mencionar gestion de los usuarios de la aplicacion con el modo desarrollo de spotify, que obliga a dar de alta previamente a hasta 25 usuarios.

4

Pruebas y resultados

En este capítulo se hablará sobre pruebas con los usuarios, y sus resultados. Puede que también se hable de las pruebas que se hicieron para decidir qué estrategias de agregación se iban a comparar.

5

Conclusiones y trabajo futuro

En este capítulo se expondrán las conclusiones que se han obtenido tras la realización de este trabajo y las últimas pruebas con usuarios, y se van a proponer posibles rumbos a seguir para trabajo futuro en el proyecto.

Bibliografía

[1] Google, "Flutter." github.com/flutter/flutter, 2017.

[2] S. AB, "Spotify." https://open.spotify.com/, 2024.

[3] S. AB, "Web api, spotify for developers." https://developer.spotify.com/documentation/web-api, 2024.

[4] Google, "Toyota showcase." https://flutter.dev/showcase/toyota, 2021.

[5] Google, "Dart." https://github.com/dart-lang, 2011.

[6] Google, "Hot reload." https://docs.flutter.dev/tools/hot-reload, 2023.

[7] Google, "pub.dev." https://pub.dev, 2023.

[8] Google, "Dart overview." https://dart.dev/overview, 2023.

[9] Google, "Flutter install." https://docs.flutter.dev/get-started/install, 2024.

[10] Google, "Building user interfaces with flutter." https://docs.flutter.dev/ui, 2023.

[11] Meta, "React." https://react.dev, 2024.

[12] S. AB, "Authorization code with pkce flow." https://developer.spotify.com/documentation/web-api/tutorials/code-pkce-flow, 2024.

[13] S. AB, "Get user's top items." https://developer.spotify.com/documentation/web-api/reference/get-users-top-artists-and-tracks, 2024.

[14] S. AB, "Get recommendations." https://developer.spotify.com/documentation/web-api/reference/get-recommendations, 2024.

[15] S. AB, "Create playlist." https://developer.spotify.com/documentation/web-api/reference/create-playlist, 2024.

[16] S. AB, "Add items to playlist." https://developer.spotify.com/documentation/web-api/reference/add-tracks-to-playlist, 2024.

[17] F. Ricci, L. Rokach, and B. Shapira, eds., *Recommender Systems Handbook*. Springer US, 2022.

[18] F. Ricci, L. Rokach, and B. Shapira, "Recommender systems: introduction and challenges," in *Recommender systems handbook*, pp. 1–34, Springer, 2022.


UÁM
Universidad Autónoma
de Madrid