

MEMORIA DE LA APLICACIÓN.

CURSO 2020-2021

SISTEMA DE RECOMENDACIÓN MUSICAL

SISTEMAS DE INFORMACIÓN DE GESTIÓN
Y BUSINESS INTELLIGENCE

ANTÍA PÉREZ-GOROSTIAGA GONZÁLEZ

ÍNDICE

1. INTRODUCCIÓN.	3
2. DESCRIPCIÓN DEL PROBLEMA.	4
2.1. ¿POR QUE ESCOGÍ ESTE TEMA?	5
2.2. IDEA PRINCIPAL.	5
3. HERRAMIENTAS UTILIZADAS.	6
3.1. HERRAMIENTAS PARA EL DESARROLLO DEL CONJUNTO DE DATOS.	6
3.2. HERRAMIENTAS PARA EL DESARROLLO DE LA BASE DE DATOS.	8
3.3. HERRAMIENTAS PARA EL DESARROLLO DEL BACKEND.	8
3.4. HERRAMIENTAS PARA EL DESARROLLO DEL FRONTEND.	9
4. FUNCIONAMIENTO DE LA APLICACIÓN.	10
4.1. BASE DE DATOS.	10
4.2. BACKEND.	15
4.3. FRONTEND.	17
5. ALGORITMO DE RECOMENDACIÓN.	27
5.1. ALGORITMO DE BÚSQUEDA.	27
5.2. ALGORITMO DE RECOMENDACIÓN ALEATORIA.	30
5.3. ALGORITMO DE RECOMENDACIÓN PERSONALIZADA.	32
5.4. ALGORITMO DE RECOMENDACIÓN POR FILTRADO COLABORATIVO.	35
6. ANÁLISIS DE RESULTADOS.	43
6.1. EJEMPLO 1.	43
6.2. EJEMPLO 2.	52
7. DAFO.	58
8. LÍNEAS DE FUTURO.	59
9. LECCIONES APRENDIDAS.	60
10. GUÍA DE INSTALACIÓN DE LA APLICACIÓN.	61
11. REFERENCIAS.	64
12. BIBLIOGRAFÍA Y WEBGRAFÍA.	65

1. INTRODUCCIÓN.

El mundo de la música es muy amplio, y a veces, de la cantidad de géneros y artistas diferentes que hay, nos cuesta mucho conocer música nueva. Por este mismo motivo, esta aplicación web nos ayudará a conocer canciones nuevas que según nuestros gustos es muy probable que nos gusten.

En primer lugar, realizaremos una exposición del problema que intentaremos resolver con nuestra aplicación, y a continuación, explicaremos las tecnologías y herramientas que hemos usado, y la composición de la aplicación, desde como hemos construido la base de datos y como funciona, hasta el funcionamiento de la aplicación explicando tanto el backend como el frontend, mostrando finalmente nuestra interfaz de usuario con sus partes detalladas.

También realizaremos un análisis de los algoritmos de recomendación usados, y analizaremos los resultados obtenidos por nuestro sistema con un par de ejemplos, comentando como funciona la aplicación tanto por fuera (la interfaz) como por dentro (lo que el usuario no ve).

Concluiremos esta memoria con un análisis crítico de las ventajas y limitaciones del trabajo, y comentaremos tanto las líneas de futuro como las lecciones aprendidas.

2. DESCRIPCIÓN DEL PROBLEMA.

Hoy en día la música es esencial para muchas personas, y tiene un rol fundamental entre muchas relaciones. La música tiene una capacidad intrínseca para regular emociones, y ciertas canciones se quedan incrustadas entre nuestros recuerdos más importantes, tanto positivos como negativos. Y nos ayuda a hablar o a sentirnos identificados con ciertos temas que de otra forma no sabríamos como afrontar.

Personalmente, la música me ha acompañado toda la vida, y durante toda la pandemia pude observar que la música, tanto para mí como para otras personas que conozco, fue una pequeña escapatoria, un pequeño rayo de luz durante esos meses en los que no podíamos salir de casa. Fue una terapia muy buena para ayudarnos a nosotros y a nuestra salud mental, y esto los artistas lo saben.

Muchos de ellos, a pesar de que no podrían dar conciertos o aún sabiendo que la gente no podría ir a comprar sus discos en formato físico, sacaron muchos temas y discos para poder aportar así su granito de arena a la situación. A su vez, pudimos observar cómo muchos realizaban conciertos online gratuitos para que así la gente pudiese desconectar aunque fuese un rato de todo. Incluso hubo grupos que se pusieron de acuerdo para realizar pequeños festivales virtuales para que así todo el mundo pudiese disfrutar de ello.

La música une a la gente, y es por eso que la idea de esta aplicación surgió en base a todo este tema. Durante ese período de tiempo, personalmente escuchaba mi música, y la que pude conocer era de artistas que ya conocía. Al tener amigos con gustos muy similares a los míos, esta opción de conocer música diferente se complicaba. Desde el comienzo de este nuevo curso escolar, conocí a gente que me enseñó música nueva y artistas que desconocía. Es por esto, por lo que decidí crear esta aplicación.

La aplicación nos permitirá conocer canciones nuevas por medio de 3 tipos de recomendaciones diferentes (aleatoria, personalizada, según lo que le ha gustado, o por medio de recomendaciones de otros usuarios con gustos similares), permitiéndole a su vez al usuario buscar temas por género, artista o palabras clave del título, y pudiendo ordenarlos por medio de ciertos parámetros, como la popularidad de la canción, la bailabilidad (dependiendo de si quieren bailar o no), la energía (si quieren una canción más enérgica más tranquila), o lo animada que sea la canción. Según como se sienta, o lo que quiera conseguir con la canción, podrá buscarla. Y podrá también indicar si una canción le ha gustado o no, podrá ver la lista de canciones que le han gustado, y podrá escuchar una preview de la canción.

2.1. ¿POR QUE ESCOGÍ ESTE TEMA?

Escogí este tema por que me parece un tema muy interesante, como ya comenté, la música está presente en mi día a día, y me acompaña siempre, según cómo me encuentre de ánimos, puedo poner diferentes playlist, así que en el momento que el profesor comentó que teníamos que hacer un sistema de recomendación de algo, y nos recomendó que fuese un tema que nos guste, no tuve ninguna duda en escoger este tema.

2.2. IDEA PRINCIPAL.

Mi idea siempre fue crear un sistema de recomendación que pudiese recomendarte música de forma aleatoria, para así descubrir una canción completamente diferente a lo que sueles escuchar, pero también, un sistema en el que pudieses buscar tú por alguna palabra clave del título, géneros, artistas, o que pudieses ordenar las canciones por algún criterio como la popularidad de la canción o la disponibilidad de la misma.

Al ser un sistema de recomendación, debía tener alguna forma de recomendar canciones acordes a los gustos del usuario, por lo que desarrollé, en un primer momento, un método que te recomienda una canción acorde a las que más de te han gustado (bien sea de un género similar, o de un artista, o ambas). Pero este método no es del todo fiable, ya que no hay nadie que corrobore si esa canción que te ha recomendado el sistema se adapta a lo que buscas. Además, las canciones serán siempre del mismo género, por lo que pierdes la oportunidad de conocer géneros nuevos. Entonces, ¿Cuál sería una buena solución?

Decidí implementar otro método que incluye un algoritmo de filtrado colaborativo. Aunque mas adelante explicaremos todo con más detalle, podemos adelantar que el filtrado colaborativo es una técnica que se encarga de filtrar información usando técnicas que implican la colaboración de múltiples agentes. Y en nuestro caso, estos agentes son los usuarios. Es decir, este sistema nos ayuda a recomendar a cierto usuario una serie de canciones que le pueden gustar, según sus gustos y con ayuda de otros usuarios.

3. HERRAMIENTAS UTILIZADAS.

A la hora de realizar este trabajo, la única exigencia por parte del profesor de la asignatura fue que debíamos usar Neo4J para la realización de la base de datos de la aplicación. Para el resto de la aplicación nos daba una gran libertad, podíamos realizar una aplicación tanto de escritorio como web, en el lenguaje que más nos gustase.

Gracias a esta libertad, elegí realizar una aplicación web, aprovechando así para poder indagar más sobre ciertas herramientas. De esta forma, dividiremos este apartado en 3 puntos para explicar así las herramientas usadas en cada uno:

3.1. HERRAMIENTAS PARA EL DESARROLLO DEL CONJUNTO DE DATOS.

En primer lugar, vamos a hablar de la creación del archivo de datos. Decidí que los datos que usaré para la base de datos los incluiría en un archivo en formato .csv, en el que todos mis datos estarían separados por el separador “,”, ya que resultaría más cómodo recorrer línea a línea este archivo obteniendo así los datos. En un primer momento, decidí escoger un conjunto de datos de “[Kaggle](#)” el cual contenía varios datos muy interesantes para poder desarrollar mi aplicación.

El problema que me encontré con esta base, es que era demasiado pequeño, contenía tan solo 50 canciones, por lo que no existía mucha variedad.

Por tanto, decidí ampliar mi conjunto de datos manualmente, por lo que busqué uno a uno los datos necesarios para mi aplicación, pero ciertos valores como la bailabilidad, la energía, la popularidad o la valencia no sabía como sacarlos, por lo que investigando un poco encontré una librería de Python llamada “[Spotipy](#)” que usa [Spotify](#), con la que podía obtener esos datos por medio de unas claves obtenidas desde la parte de [Desarrollador de Spotify](#). Y también la herramienta “[Pandas](#)” que es una herramienta de manipulación de datos de alto nivel que nos permite almacenar y manipular los datos para poder crear nuestro propio DataFrame.

Posteriormente, volví a editar mi base de datos para añadir, de forma manual, la url de la canción para poder escuchar la preview, y otro enlace para poder mostrar la imagen.

En el momento en el que decidí implementar el filtrado colaborativo, creé una nueva base de datos de forma manual con los usuarios y las canciones que les gustan, y decidí a su vez ampliar otra vez la base de datos. Por tanto, creé una playlist en Spotify con las canciones que quería incluir en mi programa, y un nuevo programa en Python desde el que obtengo 3 archivos, uno con los artistas y

ciertos datos, otro con las canciones y sus datos (como la url y la imagen), y otro con las características de las canciones (con su bailabilidad, energía, etc).

Finalmente, reuní los datos necesarios de cada archivo en un único .csv, y añadí de forma manual el género de cada canción, ya que es un dato que no conseguí obtener, ya que según investigué, Spotify no tiene almacenado este dato en todas sus canciones.

Por tanto, la primera herramienta usada es **Python** para la creación del conjunto de datos.



En esta imagen, podemos ver los conjuntos de datos “en crudo” tanto el de los usuarios y sus canciones favoritas ([datosUsuarios.csv](#)), como el de las canciones y los datos que contiene cada una ([baseDatosMusica.csv](#)). Disponemos actualmente de 16 usuarios diferentes, con un total de 327 canciones favoritas, y de 204 canciones diferentes.

Conjuntos de datos

3.2. HERRAMIENTAS PARA EL DESARROLLO DE LA BASE DE DATOS.

Para poder crear y gestionar la base de datos, utilizamos **Neo4J**. Esta herramienta de bases de datos en forma de grafos, es muy visual. Nos permite crear diferentes nodos para cada tipo de datos y relacionarlos entre sí por medio de relaciones. Los nodos pueden tener propiedades que sirven para diferenciarlos y para poder trabajar con ellos.



Antes de empezar con la base de datos, tuve que aprender cómo funciona esta nueva herramienta. Para esto, lo primero que hice fue realizar el curso básico de Neo4J para poder aprender algo más sobre esta herramienta y sobre el lenguaje Cypher, ya que van a ser las herramientas básicas para la realización del proyecto final de esta asignatura.

También, intenté realizar el examen para conseguir el certificado de Neo4j, tan sólo hice un intento, pero por suerte pude adquirir muchos conocimientos sobre Neo4j los cuales me resultaron de gran utilidad a la hora de realizar mi proyecto, y preferí ponerme a desarrollar la base de datos en vez de centrarme en el certificado ya que hablando con el profesor me comentó que no era obligatorio.

A pesar de que son muchas las páginas que me ayudaron a aprender y entender el funcionamiento de Neo4J y del lenguaje Cypher (todas están referenciadas en la webgrafía), viendo los proyectos de mis compañeros del curso 2019-2020, encontré una guía para aprender Cypher, la cual me resultó de mucha utilidad.

3.3. HERRAMIENTAS PARA EL DESARROLLO DEL BACKEND.

El backend de la aplicación es la parte encargada de que toda la lógica de una página funcione. Consiste en el conjunto de acciones que pasan dentro de una web, pero que no podemos ver. Es decir, recibe las consultas realizadas por la parte del cliente, enviadas desde el frontend, realiza estas consultas a la base de datos, obteniendo así los datos solicitados, los cuales devuelve al frontend.

Para crear el backend de nuestra aplicación, usamos un entorno de programación en tiempo real llamado **NodeJS**, el cual establece la estructura del backend, junto con el lenguaje **JavaScript**, que se encarga de darle funcionalidad al backend.



Este lenguaje es el que usamos para realizar las consultas a la base de datos, y para poder desarrollar la lógica de la aplicación.



Dentro de este entorno, hemos usado principalmente las siguientes herramientas:

Un framework web llamado **ExpressJS**. Nos permite recibir consultas del frontend.

Un driver llamado **Neo4J driver** que nos permite realizar la conexión del backend desde NodeJS con la base de datos creada en Neo4J, y también nos permite realizar peticiones a la base de datos para poder obtener así los datos necesarios.

3.4. HERRAMIENTAS PARA EL DESARROLLO DEL FRONTEND.

El frontend es la parte del sitio web que interactúa con los usuarios, la parte del cliente, contiene el código que forma la interfaz gráfica, y la parte de la lógica de la aplicación. Realiza las peticiones que posteriormente enviaremos al backend para poder obtener así los datos de la base de datos y poder realizar las operaciones necesarias para, finalmente, mostrar por los resultados en la interfaz.

Las herramientas que usamos para crear el frontend son:

JavaScript, que al igual que en el backend, le da la funcionalidad necesaria al frontend para poder realizar las operaciones necesarias para su correcto funcionamiento.



Vue.js, es un framework que nos permite la creación de aplicaciones web de una forma sencilla, aportando la estructura y parte de la funcionalidad de la interfaz.



Vuetify, es una extensión de Vue.js centrada en la interfaz de usuario.



Axios, es una librería de JavaScript que nos permite realizar peticiones de tipo HTTP desde el cliente al backend.



Electron, es un plugin que usamos para transformar la aplicación web en una aplicación instalable.



4. FUNCIONAMIENTO DE LA APLICACIÓN.

Un usuario que use nuestra aplicación, se comunicará directamente con el frontend, que contiene los elementos necesarios para componer la interfaz gráfica de usuario por medio de la cual se comunican. Desde ahí según las acciones que realice el usuario, el frontend le enviará una serie de datos y peticiones al backend para que realice las consultas necesarias en la base de datos y que finalmente le pueda devolver al frontend la respuesta obtenida, y este la pueda mostrar por pantalla.

Para poder probar la aplicación, debemos, en primer lugar, activar la base de datos desde Neo4J y debemos tener cargados en ella los datos necesarios. A continuación, debemos activar el backend y finalmente ejecutar el frontend. Todo esto lo detallaremos a continuación:

4.1. BASE DE DATOS.

Lo primero que tenemos que hacer es crear un nuevo proyecto en Neo4J pulsando sobre “+ New”. A continuación, añadimos una Base de datos de tipo grafo pulsando sobre “+ Add Database” y a continuación sobre “Create a Local Database”. Le ponemos nombre y contraseña, y pulsamos en los puntos de la derecha y le damos a “Manage”. Pulsamos sobre la flecha de “Open Folder” y pulsamos en “Import”. Arrastramos hasta esa carpeta nuestros archivos .csv, e iniciamos la base de datos pulsando sobre “Start” y finalmente, abrimos el browser de NEO4J y ejecutamos la query dada.

Ya hemos comentado cómo son los archivos de datos, pero ahora vamos a ver con más detalle sus campos.

Mi base tiene 6 tipos de nodos:

Estilos Musicales - es un nodo que agrupa los tipos de estilos o géneros, se relaciona con los géneros por medio de IS_STYLE_OF.

Géneros - agrupa a los artistas por medio de los géneros que cantan, y se relaciona con ellos por medio de GENRE_SINGED

Artistas - se relacionan por medio de MADE con las canciones.

Canciones - Contienen los datos de las canciones, como el nombre, la imagen, la url de la preview, y demás datos necesarios para desarrollar la aplicación.

UserName - se relacionan con las canciones por medio de LIKES o HATES, y con Usuarios por medio de IS_USER (para así poder tenerlos agrupados).

Usuarios - es uno de los nodos principales.

El nodo usuarios lo añadí al final para poder tener agrupados los usuarios y así que nos resulte más cómodo verlo de forma visual.

Los nodos con los que trabaja principalmente nuestra aplicación son los nodos canciones, ya que son los que contienen todos los datos necesarios para poder realizar las consultas básicas de nuestra aplicación. También son muy importantes los nodos de usuarios ya que contienen la Información necesaria para poder realizar la recomendación personalizada por medio de nuestro algoritmo de filtrado colaborativo.

En la primera base de datos, la de canciones, podemos observar que tratamos con 4 tipos de nodos diferentes, pero ninguno a parte del nodo Canciones, tiene más atributos que el nombre. Los datos que contiene el nodo canciones son los siguientes:

name: Título de la canción.

artist: Nombre del artista principal que canta la canción.

genre: Género al que pertenece la canción.

energy: Valor que indica la energía que desprende la canción.

danceability: Valor que indica loailable que es la canción.

valence: Valor que indica lo positiva que es la canción.

popularity: Valor que indica lo popular que es la canción.

cover: Enlace a la carátula de la canción.

preview: Enlace a la preview para poder escuchar la canción.

fav: Valor que indica que representa la canción para el usuario actual (el que maneja la aplicación) 0 si no la valoró, 1 si es su favorita, y 2 si no le gusta (este valor desde que se crearon los usuarios ya no es necesario).

Para transformar la base de datos “en crudo” en un grafo, construí una sentencia CYPHER que toma los diferentes datos del archivo .csv para poder transformarlos en nodos y para poder añadirseles como atributos a cada uno de los nodos creados. También, va creando las relaciones que existen entre los diferentes tipos de nodos, en este caso, creamos la relación MADE entre un artista y una canción, y las relaciones GENRE_SINGED entre un género y un artista e IS_STYLE entre un estilo y un género, para poder usar estilo como uno de los nodos principales.

```

LOAD CSV WITH HEADERS FROM 'file:///baseDatosMusica.csv' AS line WITH
line LIMIT 1000

MERGE (s:Style {name: "Estilos Musicales"})

MERGE (c:Canciones { name: line.TrackName, artist: line.ArtistName,
genre: line.Genre, energy: toInteger(line.Energy), danceability:
toInteger(line.Danceability), valence: toInteger(line.Valence),
popularity: toInteger(line.Popularity), cover: line.TrackCover,
preview: line.TrackPreview})

FOREACH (n IN (CASE WHEN line.ArtistName IS NULL THEN [] ELSE [1] END)
| MERGE (a:ArtistName {name: line.ArtistName})

    MERGE (a)-[:MADE]->(c)

    FOREACH (m IN (CASE WHEN line.Genre IS NULL THEN [] ELSE [1] END)
    | MERGE (g:Genre {name: line.Genre})

        MERGE (g)-[:GENRE_SINGED]->(a)

        MERGE (s)-[:IS_STYLE]->(g)

    )

)

```

Sentencia para la creación de la base de datos de música

En la segunda base de datos, la de usuarios, podemos observar que simplemente creamos a los diferentes usuarios y les asignamos los atributos nombre, dni, contraseña y email, y creamos 2 relaciones, LIKES entre usuario y canción, e IS_USER entre un usuario y el nodo usuarios. En esta base de datos en crudo solo están almacenadas las canciones que les gustan a los usuarios.

```

LOAD CSV WITH HEADERS FROM 'file:///datosUsuarios.csv' AS line WITH
line LIMIT 1000

MERGE (r:Users {name: "Usuarios"})

FOREACH (n IN (CASE WHEN line.UserName IS NULL THEN [] ELSE [1] END) |
MERGE (u:UserName {name: line.UserName, dni: line.Dni, passw:
line.Passw, email: line.Email})

    MERGE (t:Canciones {name: line.TrackName})

    MERGE (u)-[:LIKES]->(t)

    MERGE (u)-[:IS_USER]->(r)

)

```

Sentencia para la creación de la base de datos de usuarios

Acabamos de ver las dos sentencias usadas para generar la base de datos, y voy a comentar brevemente como funcionan.

En primer lugar, accedemos a los diferentes archivos .csv de los que necesitamos obtener los datos. Para esto, indicamos que vamos a llamar “line” a las diferentes líneas que contiene el archivo. Ponemos un límite, en este caso de 1000 líneas para facilitar la lectura de la base. A continuación, creamos los nodos “Style”, “Canciones” y “Usuarios”, y le añadimos todos los atributos necesarios. Finalmente, con un “FOREACH” comprobamos que los datos no sean nulos, y si no lo son, les añadimos las relaciones necesarias. En todo momento uso “MERGE” en vez de “CREATE” para evitar tener datos duplicados.

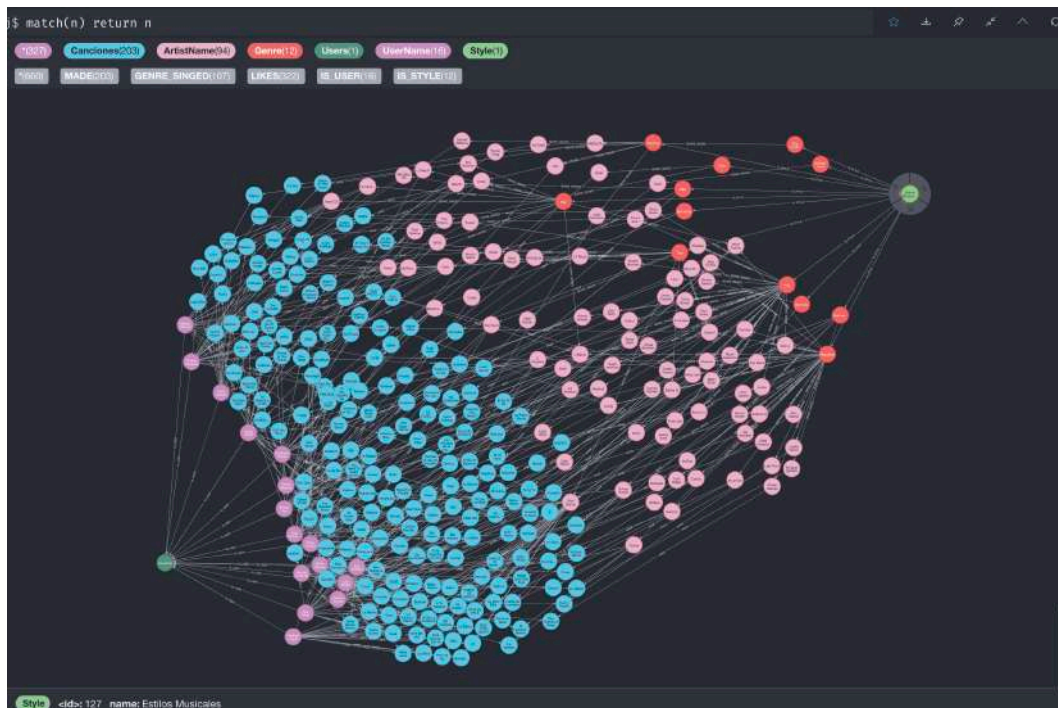


Imagen de la base de datos.

Como comenté anteriormente, tuve que ampliar varias veces la base de datos ya que no me parecían suficientes los datos obtenidos de [Kaggle](#) para poder realizar este trabajo. Por tanto, decidí ampliarla manualmente, pero me topé con un pequeño inconveniente: no sabía como obtener los datos de disponibilidad, energía, valencia y popularidad, por lo que investigando, encontré que en la página de desarrolladores de [Spotify](#), te permitían obtener unas claves con las cuales podía mediante la librería [spotipy](#) de python obtener estos datos de las canciones.

```

import pandas as pd
import spotipy

sp = spotipy.Spotify()
from spotipy.oauth2 import SpotifyClientCredentials
cid = "client_id"
secret = "client_secret"
client_credentials_manager = SpotifyClientCredentials(client_id=cid,
client_secret=secret)
sp = spotipy.Spotify(client_credentials_manager=client_credentials_manager)
sp.trace=False

playlist = sp.user_playlist("client_id", "playlist_id", fields="tracks,next")
tracks = playlist["tracks"]
songs = tracks["items"]

ids = []
song = []
artist = []

for i in range(len(songs)):
    s = songs[i]["track"]
    ids.append(s["id"])
    song.append(["https://open.spotify.com/embed/track/"+s["id"],s["name"],s["popularity"],s["album"]["images"][0]
["url"],s["artists"][0]["id"],s["artists"][0]["name"]])
    a = sp.artist(s["artists"][0]["id"])
    img = ""
    if len(a["images"]) >0:
        img = a["images"][0]["url"]

    artist.append([a["id"],a["name"],a["popularity"],a["followers"]
["total"],img])

dataArtists = pd.DataFrame(artist)

features = sp.audio_features(ids)
df = pd.DataFrame(features)

dataSongs = pd.DataFrame(song)
while tracks["next"]:

    tracks = sp.next(tracks)
    songs = tracks["items"]

    ids = []
    song2 = []
    artist2 = []

    for i in range(len(songs)):
        s = songs[i]["track"]
        ids.append(s["id"])
        song2.append(["https://open.spotify.com/embed/track/"+s["id"],s["name"],s["popularity"],s["album"]["images"][0]
["url"],s["artists"][0]["id"],s["artists"][0]["name"]])

        a = sp.artist(s["artists"][0]["id"])
        img = ""
        if len(a["images"]) >0:
            img = a["images"][0]["url"]
        artist2.append([a["id"],a["name"],a["popularity"],a["followers"]
["total"],img])

```

```

features2 = sp.audio_features(ids)
df2= pd.DataFrame(features2)
df = pd.concat([df,df2])

dataSongs2 = pd.DataFrame(song2)
dataSongs = pd.concat([dataSongs,dataSongs2])

dataArtists2 = pd.DataFrame(artist2)
dataArtists = pd.concat([dataArtists,dataArtists2])

df.to_csv("caracteristicasCanciones", sep=',', encoding='utf-8')
dataSongs.to_csv("datosCanciones", sep=',', encoding='utf-8')
dataArtists.to_csv("datosArtistas", sep=',', encoding='utf-8')

```

Código de Python para la obtención de los datos de las canciones

4.2. BACKEND.

El Backend es el encargado de mediar entre la base de datos y el frontend, procesando las peticiones que recibe por parte del frontend, solicitando y obteniendo estos datos de la base de datos, y finalmente devolviendo de nuevo al frontend los datos que solicitó.

Para implementarlo, utilizamos la herramienta Node js y lo programamos con el lenguaje JavaScript utilizando los drivers de express js y neo4j. Todo el Backend de la aplicación se encuentra ubicado en la carpeta Backend, y el código en el fichero llamado “backend.js” que es el que se encarga de realizar todas las funciones del Backend.

En él, tenemos muchos tipos de peticiones diferentes:

Peticiones para obtener datos: Se encargan de devolver los tipos de datos solicitados para poder realizar diferentes tipos de operaciones. Tenemos las siguientes:

- **getGeneros** - devuelve todos los géneros diferentes que hay almacenados en la base de datos, la usamos para, por ejemplo, rellenar el desplegable para poder buscar por géneros.
- **getArtistas** - devuelve todos los artistas diferentes que hay almacenados en la base de datos, la usamos para, por ejemplo, rellenar el desplegable para poder buscar por artistas.
- **getArtistasG** - devuelve todos los artistas que hay de un género en concreto que hay almacenados en la base de datos, la usamos para, por ejemplo, rellenar el desplegable para poder buscar por artistas una vez seleccionado un género y así evitar errores.
- **getFavs y getHateds** - devuelven una lista con las canciones que el usuarioActual indicó que le gustan o que no le gustan.

- **getNeighbours** - devuelve a los usuarios vecinos del usuarioActual, aquellos a los que le gusten las mismas canciones que a él.
- **getRandomSongs** - devuelve una canción aleatoria.
- **getSongs** - devuelve las canciones que coincidan con los criterios de búsqueda.
- **getPersonalizedSongs** - devuelve una canción recomendada de forma personalizada según los gustos que tiene el usuarioActual.
- **getSongsColaborativeFilter** - devuelve las canciones recomendadas para el usuarioActual dependiendo de sus gustos y basándose en usuarios con gustos similares.

Peticiones para añadir datos: Se encargan de crear una relación o un usuario nuevo a la base de datos. Hay dos que se encargan de sobre escribir un dato, pero estas podríamos obviarlas.

Tenemos las siguientes peticiones:

- **postNewUser** - se encarga de crear el nuevo usuario y añadirlo a la base (la usamos para añadir al usuarioActual una vez realiza alguna operación de añadir a favoritos o indicar que no le gusta cierta canción).
- **postNewSongLiked y postNewSongHated** - se encargan de crear una relación (LIKES o HATES) entre el usuarioActual y una canción, y añadirla a la base de datos.
- **addFav y addDislike** - estas peticiones son las que podríamos eliminar, en las primeras versiones servían para indicar si una canción era del gusto del usuarioActual o no, ahora que las almacenamos en la base de datos no son estrictamente necesarias. Al igual que las anteriores indican si al usuario le gusta o no una canción y lo indica sobre escribiendo el atributo “fav”.

Peticiones para eliminar datos: Se encargan de eliminar una cierta relación creada en la base de datos.

Tenemos las siguientes peticiones:

- **postDeleteRelation** - se encarga de eliminar la relación creada entre el usuarioActual y una canción. Esta relación puede ser tanto LIKES como HATES.
- **deleteFavDislike** - al igual que en con addFav y addDislike, no es estrictamente necesaria, y modifica el atributo “fav” poniéndolo a 0 para así indicar que ya no es de sus favoritas ni de las canciones que no le gustan.

Ahora, explicaremos brevemente cómo se realizan en general las consultas a la base de datos.

Con cada petición, generalmente, queremos obtener una lista de elementos dependiendo, o no, de algún dato de entrada.

En primer lugar, abrimos la session y creamos una lista vacía, la cual rellenaremos con los valores obtenidos y se la devolveremos al Frontend.

Sólo si nos han mandado algún dato, los guardamos en variables para poder usarlos mas cómodamente.

A continuación, escribimos la query y se la mandamos a nuestra base de datos por medio del método run de la session.

Como resultado, obtenemos 3 situaciones diferentes:

onNext: se encarga de devolver fila a fila la tabla con todos los nodos de la base de datos que cumplan las condiciones de la query. Guardamos las propiedades de cada fila en la lista que vamos a devolver al frontend.

onCompleted: una vez terminó de devolver todos los datos referentes a la consulta se ejecuta este apartado. Aquí devolvemos la lista creada al frontend y cerramos la session.

onError: este apartado sólo se ejecuta si se produce un error.

En próximos apartados, explicaremos en profundidad el funcionamiento de los algoritmos de recomendación, y también, explicaremos con más detalle como funcionan algunas de estas peticiones.

4.3. FRONTEND.

El Frontend es la parte de la aplicación que ve el usuario y con la que interactúa. La creamos con ayuda de Vuetify, que es una herramienta que contiene los componentes que necesitamos para poder crear la interfaz web y para permitirle al usuario interactuar con ella. Usamos Vuetify a partir de VueJs, que es el framework original que lo hace funcionar. También usamos Javascript para realizar toda la lógica de la aplicación, y axios para poder enviar y recibir las peticiones desde Frontend al Backend utilizando peticiones HTTP.

En mi opinión, la interfaz debe ser atractiva para el usuario. Si la interfaz es bonita, sencilla de usar e intuitiva, el usuario querrá probarla, ya que, al no tener conocimiento previo de cómo funciona, se guía por lo que puede ver.

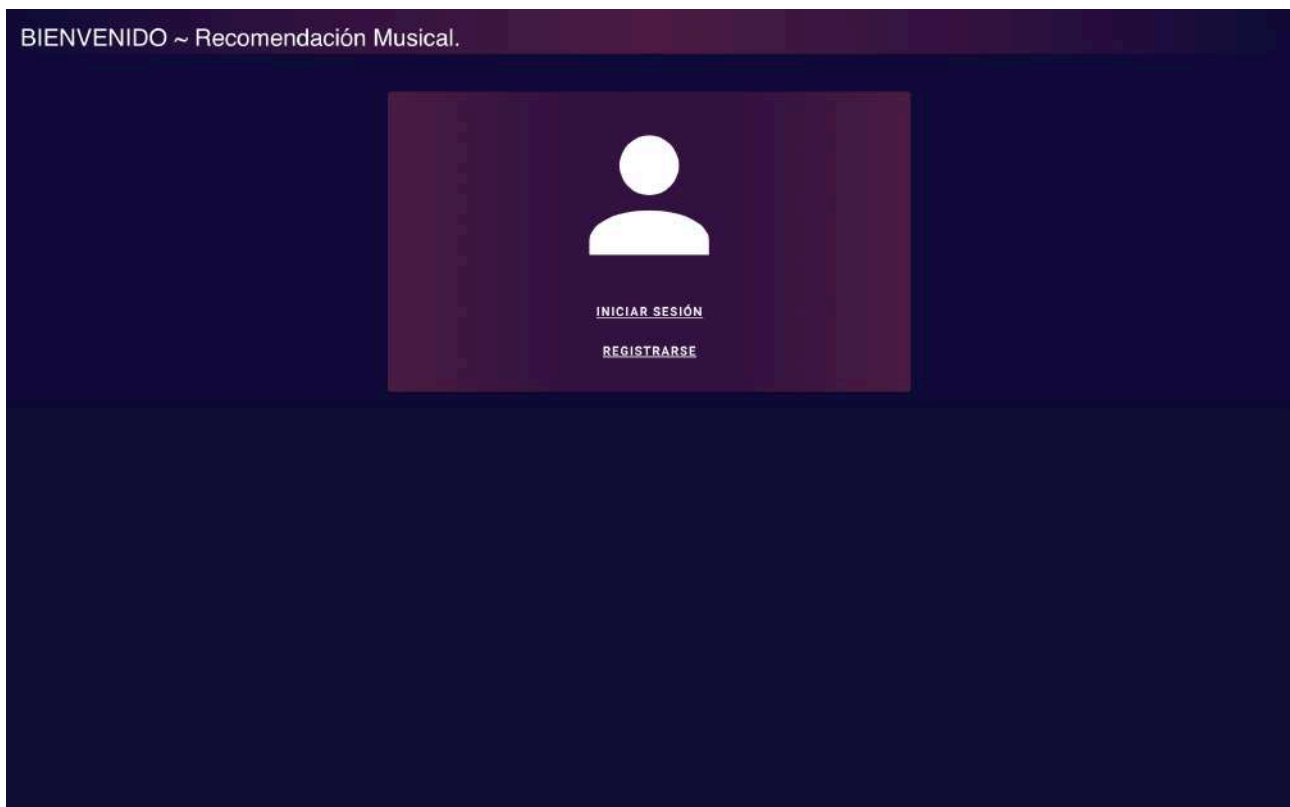
Personalmente, me parecen mas atractivas las páginas en tonalidades oscuras, por lo que tenía claro que me gustaría que mi aplicación fuese de este estilo. Pero no quería que fuese la típica con el fondo negro. El morado me pareció un color muy adecuado para este proyecto.

La idea principal era usarlo como color base para las barras de la aplicación y para el fondo, en distintos tonos, y darle un efecto mas llamativo por medio de las tarjetas que contienen las canciones, las cuales tendrán diferentes colores. Finalmente, decidí crear un efecto degradado de un tono morado oscuro, con un sub tono más azulado, hasta un tono morado que tira mas hacia el borgoña.

Para las diferentes tarjetas que contienen las canciones, tenía claro que quería usar colores mas vivos, pero para mejorar la estética, no quería que fuesen en cualquier orden. Por tanto, hice una lista que contenía una selección de 5 colores, los cuales combinan bien entre ellos y con la aplicación.

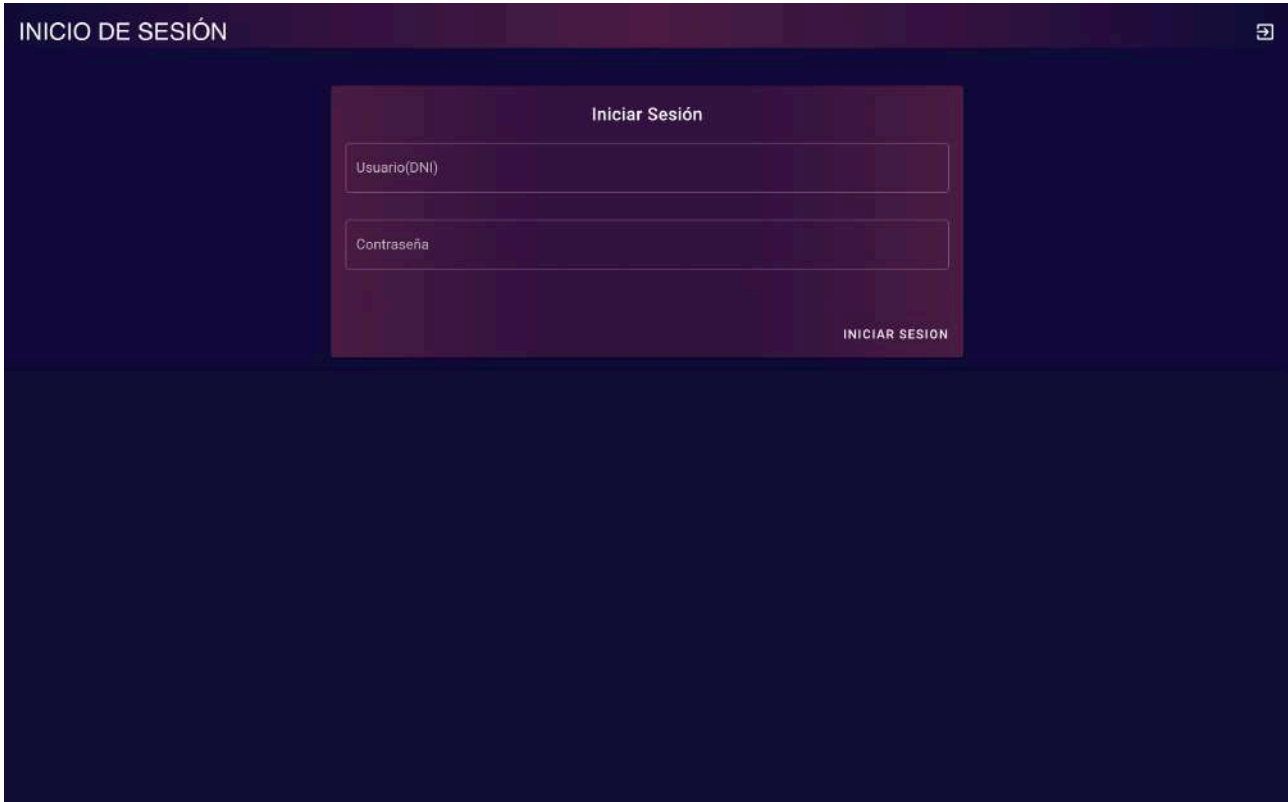
A pesar de que quiero que mi interfaz sea bonita y atractiva, no podía descuidar la parte de la accesibilidad. La aplicación debe ser accesible para todo el mundo, las personas daltónicas no deben tener ningún tipo de problema a la hora de visualizar la aplicación. Para comprobar esto, instalé una aplicación para simular como una persona daltónica vería la interfaz, y los resultados fueron bastante satisfactorios. A pesar de que no es lo mismo, considero que sigue siendo bastante atractiva. También decidí mostrarle la interfaz a unos amigos que sufren esta alteración en diferentes grados, y también me dieron su visto bueno.

La primera interfaz que podemos ver cuando accedemos a la aplicación, es la de entrada, en la cual debemos pulsar uno de los dos botones principales, login o registro.



Pantalla de Entrada de la Aplicación.

Si pulsamos sobre Login, nos lleva a la interfaz de Login donde tenemos dos campos para rellenar con nuestro dni (se comprobará que el dni introducido sea válido con un formato de 8 números y 1 letra mayúscula) y nuestra contraseña, un botón de iniciar sesión desde donde se comprobará si los datos son correctos, y de ser así, se accedería a la pantalla de inicio de la aplicación, y en la barra superior un icono para volver a la pantalla de entrada.



Pantalla de Inicio de Sesión de la Aplicación.

Si pulsamos sobre Registro, nos llevará a una interfaz con varios campos a rellenar, que son: nombre, dni (se comprobará que el dni introducido sea válido con un formato de 8 números y 1 letra mayúscula, y que no esté ya registrado en la base de datos), email (se comprobará que el email introducido sea de tipo xxx@xxx.xx y tenga mínimo 10 caracteres en ese orden, es decir, el nombre debe tener longitud mínima 3, seguido de un '@' y el servidor de longitud mínima 3, seguido de un '.' y el dominio de longitud mínima 2 y máxima 5), y una contraseña de mínimo 3 caracteres. Una vez pulsamos sobre el botón de registrarse, si todos los datos son correctos, se creará el usuario y se accederá directamente a la pantalla de inicio de la aplicación. En la barra superior tenemos también un icono para volver a la pantalla de entrada.

Pantalla de Registro de la Aplicación.

Podemos dividir la interfaz principal, o interfaz de inicio, en 5 secciones:

Barra Superior. Esta barra, que se usará a lo largo de las siguientes pantallas, contiene, de izquierda a derecha, un menú lateral desplegable, que contiene diferentes secciones según en que interfaz estés (Inicio, que nos dirige a la página en la que estamos, Favoritos, que nos dirige a una página donde podemos ver la lista de canciones que nos han gustado, Ayuda, que nos dirige a una página donde podemos ver como funciona la aplicación, y Cerrar Sesión, que cierra nuestra sesión), a continuación, se encuentra el nombre de la pantalla en la que estas, salvo en la de inicio que pone MUSIC, y en la de inicio y en favoritos aparece también el nombre del usuario, y finalmente un icono en forma de corazón que nos dirige a la página donde podemos ver la lista de canciones que nos han gustado, o si estamos en esta página, o en la de ayuda, veremos un icono en forma de casa que nos dirige a la página de inicio de la aplicación, y un icono para cerrar la sesión.

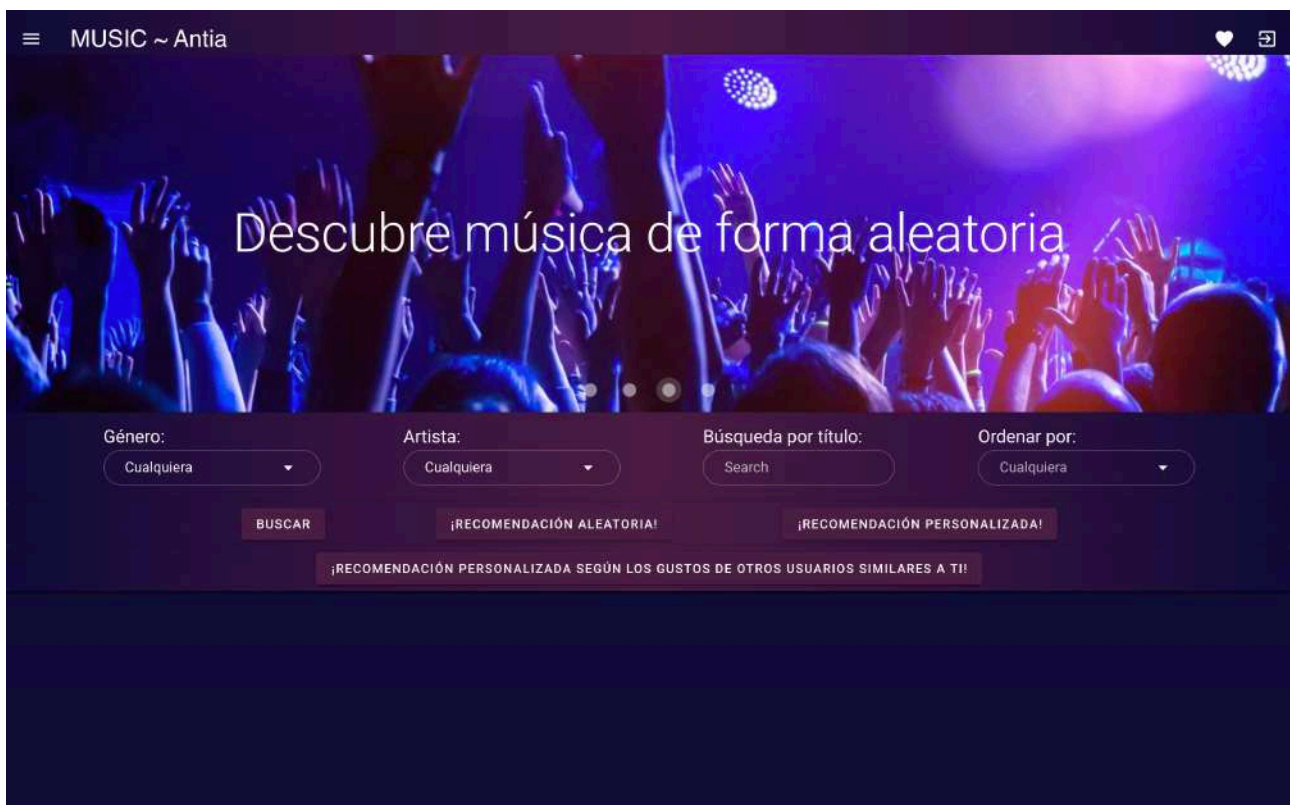
Carrusel de imágenes. Quería incluir una serie de imágenes relacionadas con la página, que contuviesen unas frases para, de alguna forma, explicar lo que se puede hacer allí. Las imágenes son atractivas a la vista, y siempre hacen que el usuario muestre más interés.

Sección de filtros. Algo necesario para esta aplicación son los filtros de búsqueda. Tenemos dos menús desplegables que permiten buscar

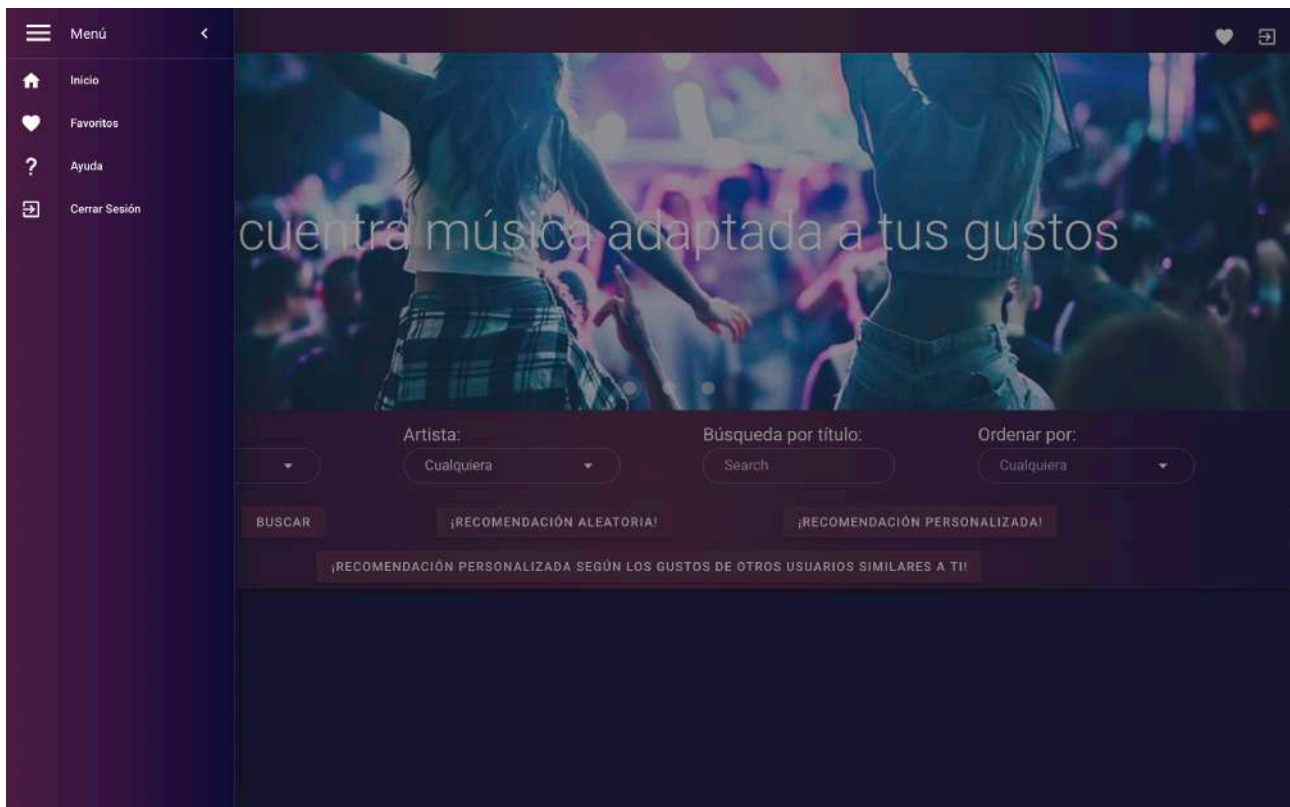
canciones por género y por artista, y tenemos también una barra de búsqueda que nos permite buscar canciones por el título. Para poder organizar un poco esta información, tenemos otro menú desplegable que nos permite seleccionar un máximo de dos formas de ordenación, para poder obtener las canciones ordenadas en función de esos parámetros.

Sección de botones. Para que todo esto funcione, tenemos el botón de Buscar que nos devolvería las canciones que cumplen las condiciones que el usuario indicó por medio de los filtros. También, tenemos un botón que nos permite obtener una canción de forma completamente aleatoria, otro que nos permite obtener una canción en función de los gustos del usuario, y un último botón que nos permite obtener una lista de canciones recomendadas para el usuario, basándose en sus gustos y en usuarios con gustos similares.

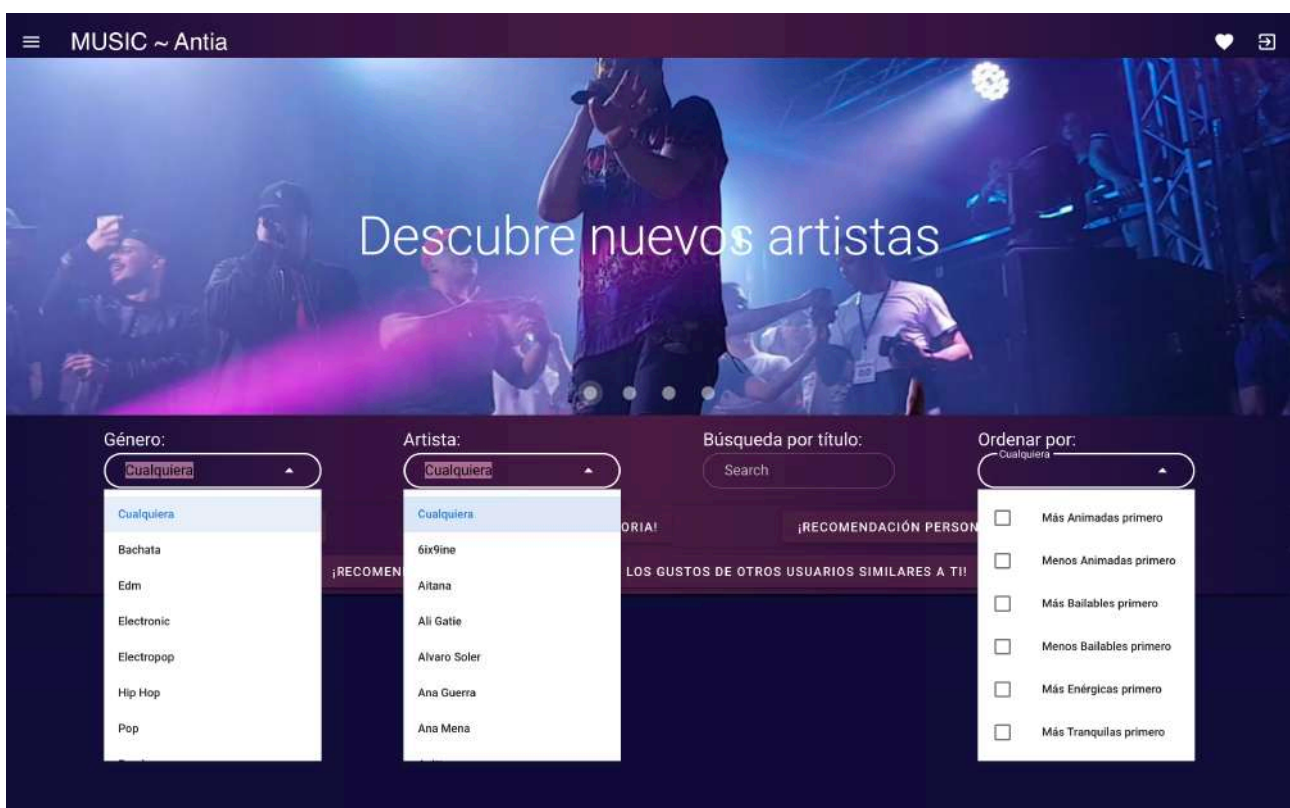
Sección de canciones. Muestra por medio de tarjetas las canciones que obtenemos al realizar algún tipo de búsqueda o consulta. Estas tarjetas son de 5 colores diferentes, siguiendo un orden, y contienen el título de la canción, el artista, el género al que pertenece, la preview de Spotify, un botón de no me gusta, otro de añadir a favoritos, los cuales se muestran rellenos si la canción ya la añadimos a alguna de las dos listas, y solo el contorno si no las añadimos aún, y por último la carátula de la canción.



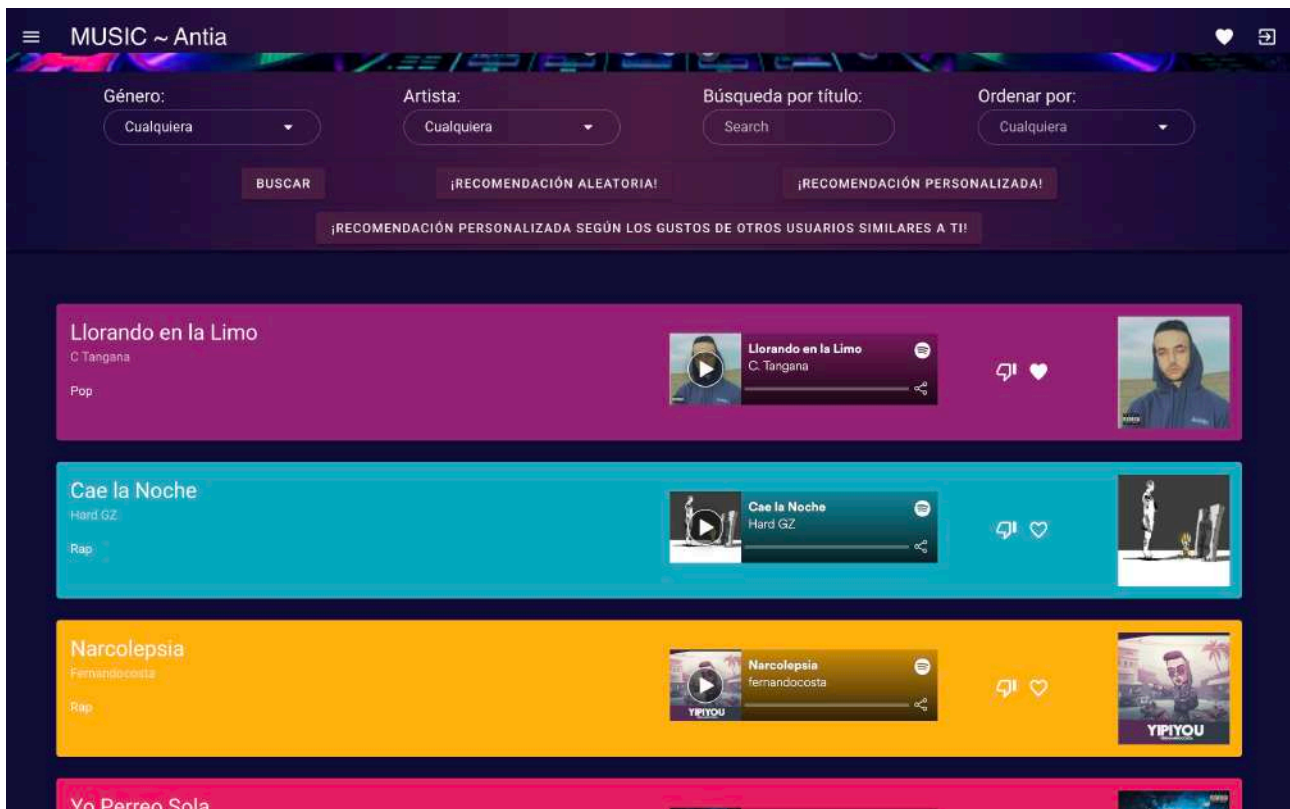
Pantalla de Inicio de la Aplicación.



Menú Lateral Desplegado.



Filtros de búsqueda.

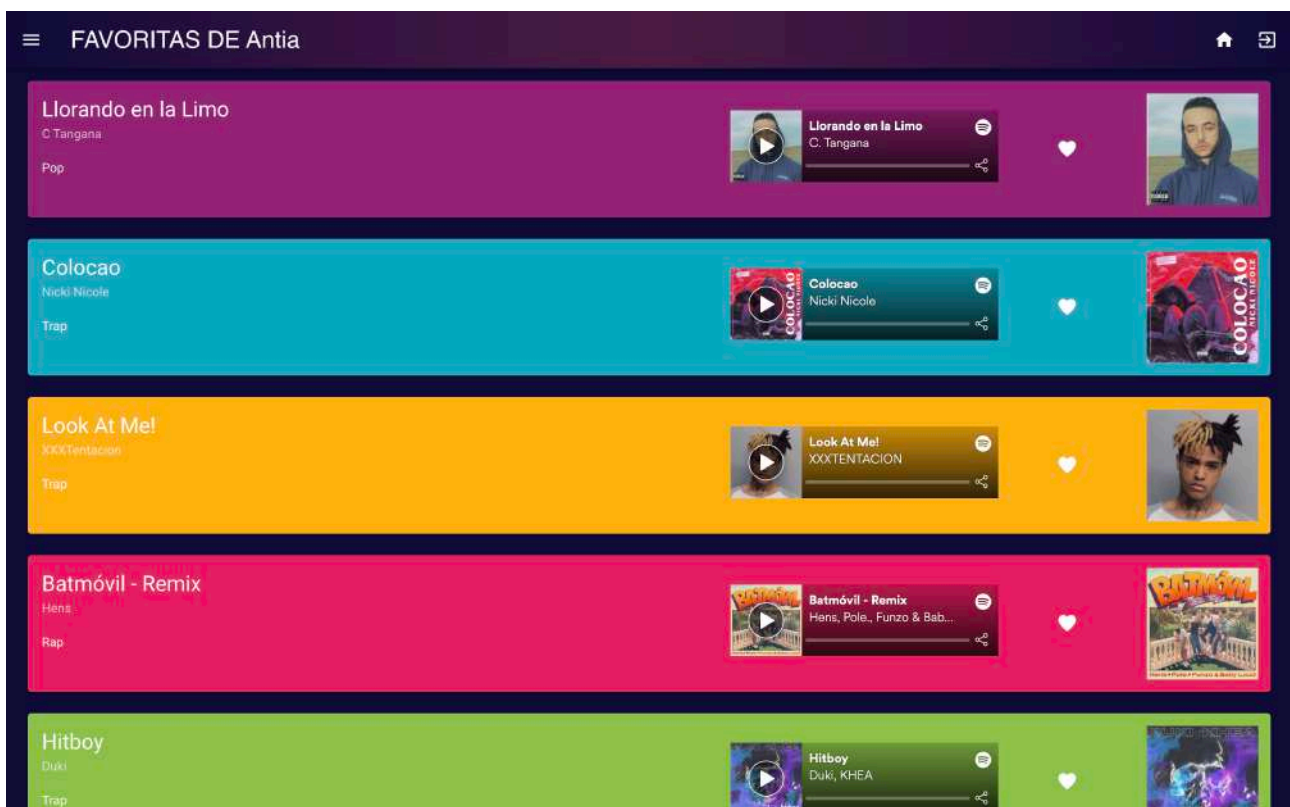


Sección de Canciones.

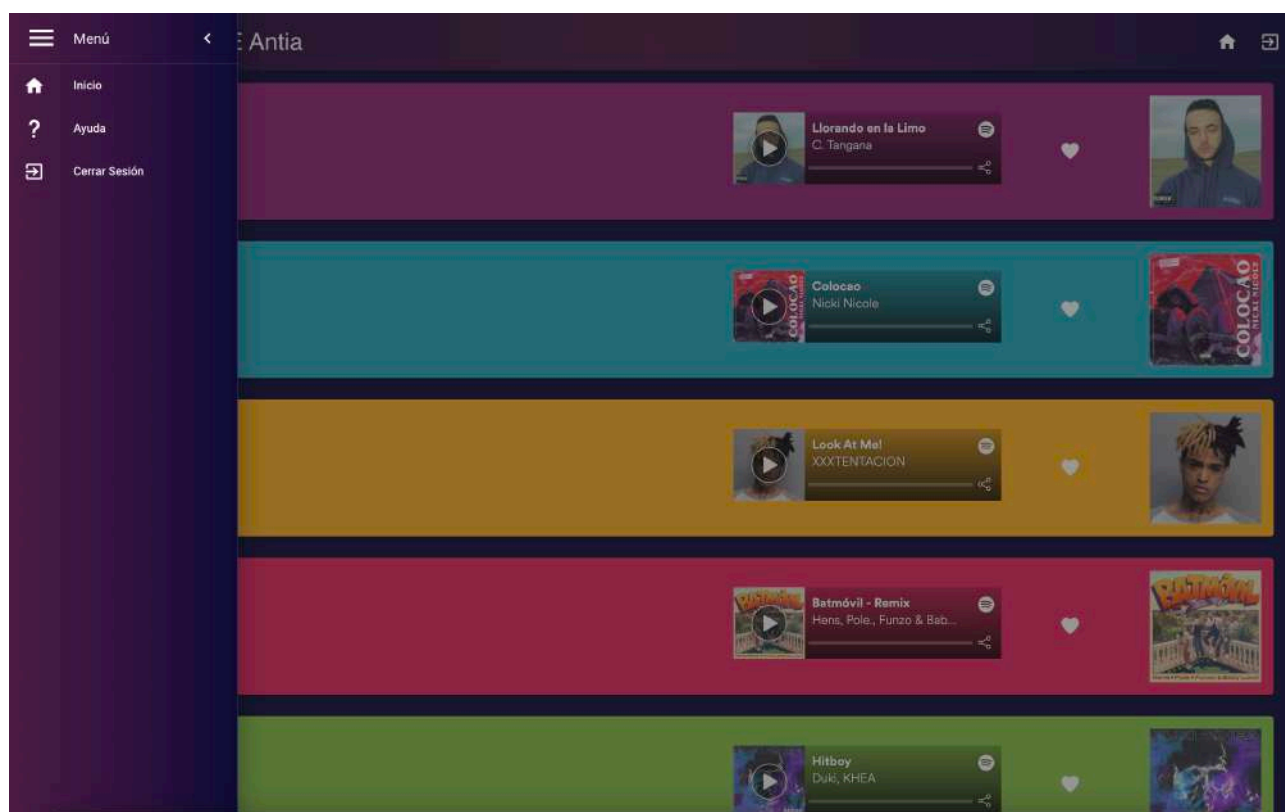
En la parte de sección de canciones, podemos ver que los botones de me gusta y no me gusta se muestra solo el contorno si esa canción no está añadida a la lista de canciones que me gustan o que no me gustan, y relleno en caso contrario. El botón cambia en el momento en el que lo pulsamos, y si marcamos como favorita una canción que teníamos como que no nos gustaba, nos muestra un mensaje de que se elimina dicha canción de la lista de canciones que no te gustan y se añade a favoritos, lo mismo sucede en el caso contrario. Además, si pulsamos otra vez sobre el botón de me gusta, de una canción que ya está añadida a favoritas, salta otro mensaje diciendo que se retira de esa lista, y lo mismo ocurre con el botón de no me gusta.

Al pulsar sobre alguno de los botones de recomendación, debemos saber que las canciones que nos muestran no se encuentran entre las canciones que nos gustan ni entre las que no nos gustan, pero cuando buscamos una canción si que pueden aparecer, y como comentamos, aparece reflejado en los botones.

Si queremos ver las canciones que hemos añadido a favoritos, accedemos a la pantalla de Favoritas, por medio de alguno de los dos botones disponibles (el de podemos ver en la barra superior, o el que aparece en el menú lateral). Una vez ahí, podemos ver la misma barra superior que en la pantalla de inicio, tan solo cambia uno de los iconos, en vez de el corazón para acceder a favoritos aparece una casa para volver a la pantalla de inicio. En el menú desplegable, tampoco aparece la opción de ir a la pantalla de Favoritas. Además de esta barra, veremos una sección donde, al igual que en la pantalla anterior, se nos mostrarán todas las canciones que nos han gustado en diferentes tarjetas. Estas tarjetas son de 5 colores diferentes, siguiendo un orden, y contienen el título de la canción, el artista, el género al que pertenece, la preview de Spotify, un botón de añadir a favoritos, desde el que podemos quitarlo o añadirlo de la lista (se mostrarán rellenos, y si pulsamos sobre uno de ellos, se quedará solo el contorno, la canción seguirá mostrándose en pantalla hasta que cambiemos, por si la quisieses volver a añadir), y por último la carátula de la canción.

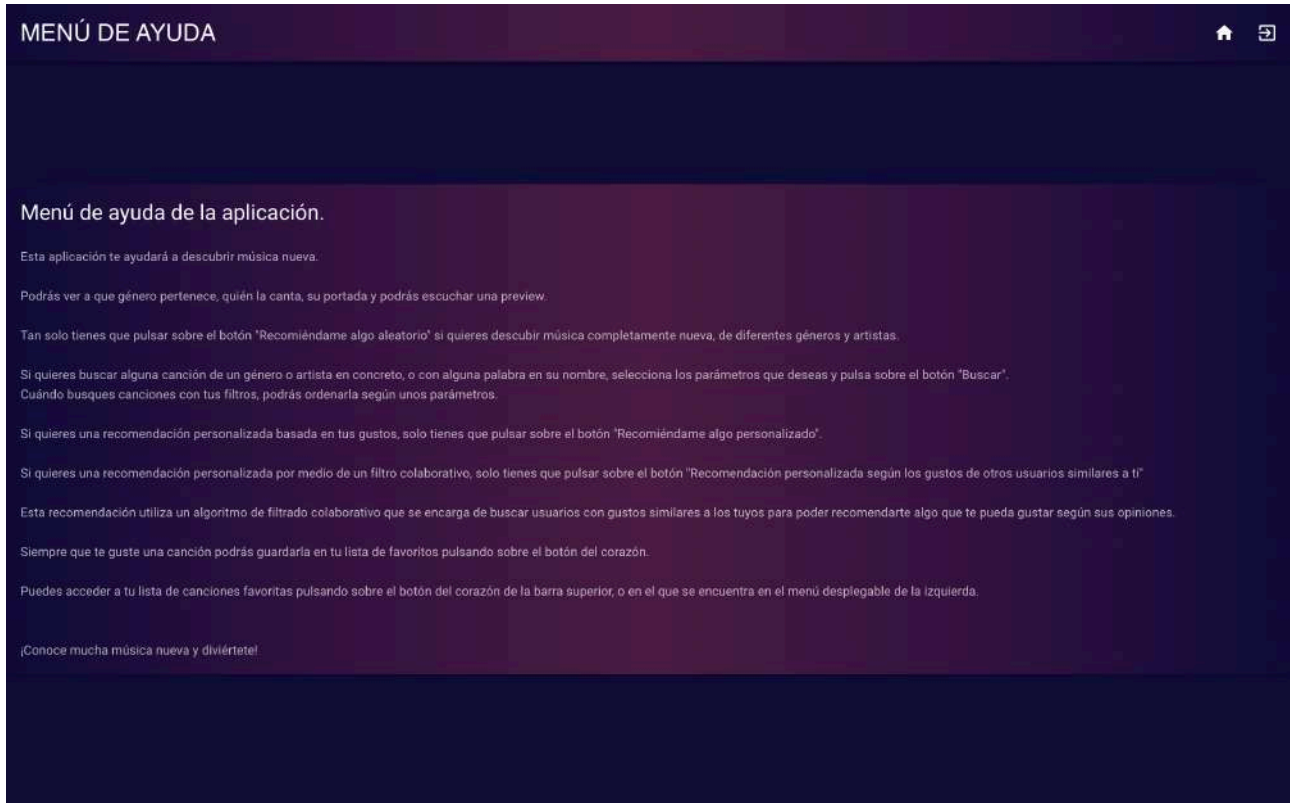


Pantalla de Favoritas de la Aplicación.



Menú Lateral Desplegado (Favoritas).

Finalmente, por medio del menú lateral podemos acceder a la pantalla de ayuda, en ella se muestra una barra superior con el nombre de la pantalla, el icono de regreso a la pantalla de inicio, y el icono de cerrar sesión. Además de esto, podemos ver un contenedor con una explicación de cómo funciona la aplicación.



Pantalla de Ayuda de la Aplicación.

5. ALGORITMO DE RECOMENDACIÓN.

En este apartado intentaremos detallar el funcionamiento de los algoritmos de recomendación que empleamos en la aplicación.

5.1. ALGORITMO DE BÚSQUEDA.

Este primer algoritmo se encarga de obtener una canción dependiendo de los parámetros proporcionados por el usuario. Para esto, realizamos una consulta al Backend enviándole todos los datos que nos proporcionó el usuario. Una vez estamos en el Backend, construimos la **query**. El usuario puede proporcionarnos un sólo dato, varios datos, o ningún dato. Por este mismo motivo, decidí que debería crear la **query** por partes.

```

/***** GET FILTERED SONGS *****/
app.get("/getSongs", (req, res) => {
  console.log("Estoy en funcion /getSongs");
  const session = driver.session();
  var genre = req.query.genre;
  var busqueda = req.query.busqueda;
  var artist = req.query.artist;
  var type = [];
  type = req.query.type;
  var lista = [];
  var query = "MATCH (c: Canciones) ";

  //FILTRAR POR GÉNERO
  if (genre != "Cualquiera" && genre != "") {
    query += "where c.genre='" + genre + "' ";
  }

  //FILTRAR POR ARTISTA
  if (artist != "Cualquiera" && artist != "") {
    if (query.includes("where")) {
      query += "AND c.artist='" + artist + "' ";
    } else {
      query += "where c.artist='" + artist + "' ";
    }
  }

  //FILTRAR POR NOMBRE BUSCADO
  if (busqueda != "") {
    if (query.includes("where")) {
      query += " AND (c.name =~ '(?i).*' + busqueda + '.*') ";
    } else {
      query += " where (c.name =~ '(?i).*' + busqueda + '.*') ";
    }
  }

  query += "return c";
  //ORDENAR SEGUN FILTROS
  console.log(type);
  if (type[0] == "No") {
    for (var i = 0; i < type.length; i++) {
      if (type[i] == "+animada") {
        if (i == 0) {
          query += " order by c.valence desc";
        } else {
          query += ", c.valence desc";
        }
      } else if (type[i] == "-animada") {
        if (i == 0) {
          query += " order by c.valence";
        } else {
          query += ", c.valence";
        }
      } else if (type[i] == "+bailable") {
        if (i == 0) {
          query += " order by c.danceability desc";
        } else {
          query += ", c.danceability desc";
        }
      }
    }
  }
}

```

```

    }
    else if(type[i] == "-bailable"){
      if(i == 0){
        query += " order by c.danceability";
      }
      else{
        query += ", c.danceability";
      }
    }
    else if(type[i] == "+energica"){
      if(i == 0){
        query += " order by c.energy desc";
      }
      else{
        query += ", c.energy desc";
      }
    }
    else if(type[i] == "-energica"){
      if(i == 0){
        query += " order by c.energy";
      }
      else{
        query += ", c.energy";
      }
    }
    else if(type[i] == "+popular"){
      console.log(++POPULARR);
      if(i == 0){
        query += " order by c.popularity desc";
      }
      else{
        query += ", c.popularity desc";
      }
    }
    else if(type[i] == "-popular"){
      if(i == 0){
        query += " order by c.popularity";
      }
      else{
        query += ", c.popularity";
      }
    }
  }
}
console.log(query);
const resultadoPromesa = session.run(query).subscribe({
  onNext: function (result) {
    lista.push(result.get(0).properties);
    console.log(result.get(0).properties);
  },
  onCompleted: function () {
    res.send(lista);
    console.log(lista);
    session.close();
  },
  onError: function (error) {
    console.log(error + "No hay canciones que cumplan estos criterios de búsqueda.");
  }
})
console.log("SALGO de /getSongs\n\n");
});

```

Función getSongs (Backend)

```

/***** GET SONG *****/
getSong() {
  console.log("ESTOY EN GET SONG");
  this.songs = [];
  this.getFavs();
  this.getHateds();
  for(var j = 0; j < this.type.length; j++){
    for(var i = 0; i < this.types.length; i++){
      if(this.type[j] == this.types[i]){
        console.log(this.type[j]);
        console.log(this.types[i]);
        if(i == 0){
          this.aux[j] = "+animada";
        }
        else if(i == 1){
          this.aux[j] = "-animada";
        }
        else if(i == 2){
          this.aux[j] = "+bailable";
        }
        else if(i == 3){
          this.aux[j] = "-bailable";
        }
        else if(i == 4){
          this.aux[j] = "+energica";
        }
        else if(i == 5){
          this.aux[j] = "-energica";
        }
        else if(i == 6){
          this.aux[j] = "+popular";
        }
        else if(i == 7){
          this.aux[j] = "-popular";
        }
      }
    }
  }
  if(this.type.length == 0){
    this.aux[0] = "No";
  }
  axios.get(direccionIp + "/getSongs", {
    params: {
      genre: this.genre,
      artist: this.artist,
      type: this.aux,
      busqueda: this.busqueda,
    },
  }).then(
    respuesta => {
      if(respuesta.data[0] == "No hay canciones que cumplan estos criterios de búsqueda."){
        alert(respuesta.data[0]);
      }else{
        this.num = 0;
        for(var i = 0; i < respuesta.data.length; i++){
          this.songs.push(respuesta.data[i]);
          this.songs[i].color = this.colors[this.num];
          this.songs[i].iconF = 'mdi-heart-outline';
          this.songs[i].iconD = 'mdi-thumb-down-outline';
          this.num = this.num + 1;
          if(this.num == this.colors.length){
            this.num = 0;
          }
        }
        for(var k = 0; k < this.songs.length; k++){
          if(this.favs.length > 0){
            for(var l = 0; l < this.favs.length; l++){
              if(this.songs[k].name == this.favs[l].name){
                this.songs[k].iconF = 'mdi-heart';
                l = this.favs.length;
              }
            }
          }
          if(this.hateds.length > 0){
            for(var m = 0; m < this.hateds.length; m++){
              if(this.songs[k].name == this.hateds[m].name){
                this.songs[k].iconD = 'mdi-thumb-down';
                m = this.hateds.length;
              }
            }
          }
        }
      }
    }
  );
  this.aux = [];
  this.type = [];
  this.artist = "Cualquiera";
  this.busqueda = "";
  this.genre = "Cualquiera";
  console.log("SALGO DE GET SONG\n\n");
}

```

5.2. ALGORITMO DE RECOMENDACIÓN ALEATORIA.

Este algoritmo se encarga de obtener una canción de forma completamente aleatoria. Este es el algoritmo más sencillo. Al pulsar en el botón de recomendación aleatoria, se inicia la función **getRandomSong** la cual obtiene por medio de **Math.random** un número aleatorio que se corresponde con un género. Realizamos una solicitud al Backend por medio de **axios**, pasándole dicho género como dato, y recibimos como respuesta una lista con las canciones. A continuación, obtenemos otro número aleatorio, y almacenamos en la lista songs la canción que ocupa la posición que nos indica el número obtenido. Esa será la canción aleatoria que le recomendamos al usuario.

```

/***** GET RANDOM SONGS *****/
app.get("/getRandomSongs", (req, res) => {
  console.log("Estoy en función /getRandomSongs");
  const session = driver.session();
  var genre = req.query.genre;
  var dni = req.query.dni;
  var query = "Match(c:Canciones), (a:UserName) where c.genre='" + genre + "' and a.dni='"
    + dni + "' AND NOT (a)-[:LIKES]->(c) AND NOT (a)-[:HATES]->(c) ";
  var lista = [];
  query += "return c";
  const resultadoPromesa = session.run(query).subscribe({
    onNext: function (result) {
      lista.push(result.get(0).properties);
    },
    onCompleted: function () {
      res.send(lista);
      session.close();
    },
    onError: function (error) {
      console.log(error + " ERROR");
    }
  })
  console.log("SALGO de /getRandomSongs\n\n");
});

```

Función getRandomSongs (Backend)


```

/***** RANDOM SONG *****/
randomSong() {
  console.log("ESTOY EN RANDOM SONG");
  this.songs = [];
  this.getFavs();
  this.getHateds();
  setTimeout(() =>{
    var random = Math.floor(Math.random() * this.generos.length);
    var g = this.generos[random];
    axios.get(direccionIp + "/getRandomSongs",{
      params:{
        genre: g,
        dni: this.dni,
      },
    }).then(response => {
      if(response.data[0] == " ERROR"){
        alert(response.data[0]);
      }
      else {
        if(response.data[0] == undefined){
          this.randomSong();
        }
        else {
          var valida = false;
          do{
            var randomm = Math.floor(Math.random() * response.data.length);
            var inFav = false;
            var inHated = false;
            for(var x = 0; x < this.favs; x++){
              if(response.data[randomm].name == this.favs[x].name){
                inFav = true;
                break;
              }
            }
            if(inFav == false){
              for(var y = 0; y < this.hateds; y++){
                if(response.data[randomm].name == this.hateds[y].name){
                  inHated = true;
                  break;
                }
              }
            }
            if(inFav == false && inHated == false){
              this.songs.push(response.data[randomm]);
              valida = true;
            }
          } while(valida == false);
          this.songs[0].color = this.colors[Math.floor(Math.random() * this.colors.length)];
          this.songs[0].iconF = 'mdi-heart-outline';
          this.songs[0].iconD = 'mdi-thumb-down-outline';
        }
      }
    });
  }, 200);
  this.type = [];
  this.artist = "Cualquiera";
  this.búsqueda = "";
  this.genre = "Cualquiera";
  console.log("SALGO DE RANDOM SONG\n\n");
},

```

Función randomSong (Frontend)

5.3. ALGORITMO DE RECOMENDACIÓN PERSONALIZADA.

Este es el primer algoritmo de recomendación que hice. Mi principal objetivo era poder obtener canciones según los gustos del usuario. Si un usuario indica que le gustan 3 canciones de un género x, y tan solo una de otro género, podremos suponer que una canción del primer género le va a gustar con una mayor probabilidad que otra del segundo género.

Por tanto, en esta función, lo primero que hacemos es realizar un conteo de los géneros que tiene el usuario añadidos a favoritos y de los artistas. A continuación, vemos si el artista pertenece al género que suponemos que le gusta. Si es así, y si tiene más canciones que la que le gusta, buscamos una canción de ese artista y género. Si no tenemos alguno de los datos, los calculamos aleatoriamente.

Finalmente, realizamos una solicitud al Backend por medio de **axios**, pasándole los datos de género y artista (indicamos que la canción no debe ser favorita).

```

/***** GET PERSONALIZED SONGS *****/
app.get("/getPersonalizedSongs", (req, res) => {
  console.log("Estoy en funcion /getPersonalizedSongs");
  const session = driver.session();
  var genre = req.query.genre;
  var dni = req.query.dni;
  var artist = req.query.artist;
  var query = "Match(c:Canciones), (a:UserName) where c.genre='" + genre + "' and a.dni='"
+ dni + "' AND NOT (a)-[:LIKES]->(c) AND NOT (a)-[:HATES]->(c)";
  var lista = [];
  query += "return c";
  const resultadoPromesa = session.run(query).subscribe({
    onNext: function (result) {
      lista.push(result.get(0).properties);
    },
    onCompleted: function () {
      res.send(lista);
      session.close();
    },
    onError: function (error) {
      console.log(error + "No hay canciones que cumplan estos criterios de búsqueda.");
    }
  })
  console.log("SALGO de /getPersonalizedSongs\n\n");
});

```

Función getPersonalizedSongs (Backend)


```

/***** PERSONALIZED SONG *****/
personalizedSong(){
  console.log("Estamos en la función PERSONALIZED SONG");
  var freqGenre = "";
  var freqArtist = "";
  var mostFreqGenre = 0;
  var mostFreqArtist = 0;
  var posibleGenres = [];
  var posibleArtists = [];
  this.songs = [];
  this.getFavs();
  this.getHateds();
  setTimeout(() =>{
    for (var i = 0; i < this.favs.length; i++) {
      var countGenre = 1;
      var countArtist = 1;
      for (var j = i; j < this.favs.length; j++) {
        if (this.favs[i].genre == this.favs[j].genre) {
          countGenre++;
        }
        if (mostFreqGenre < countGenre) {
          mostFreqGenre = countGenre;
          freqGenre = this.favs[i].genre;
        }
        if (this.favs[i].artist == this.favs[j].artist) {
          countArtist++;
        }
        if (mostFreqArtist < countArtist) {
          mostFreqArtist = countArtist;
          freqArtist = this.favs[i].artist;
        }
      }
    }
    this.rellenaArtistaG(freqGenre);
    var generoArtista = false;
    setTimeout(()=> {
      for(var w = 0; w < this.artistas.length; w++){
        if(this.artistas[w] == freqArtist){
          console.log(this.artistas[w]);
          generoArtista = true;
          break;
        }
      }
      if(generoArtista == false){
        freqArtist = "";
      }
      if (freqGenre == "" && freqArtist == "") {
        console.log("No se puede dar una recomendación personalizada ya que no hay datos suficientes."
          + "La recomendación será aleatoria.");
        var random = Math.floor(Math.random() * this.generos.length);
        freqGenre = this.generos[random];
        this.rellenaArtistaG(freqGenre);
        var random2 = Math.floor(Math.random() * this.artistas.length);
        freqArtist = this.artistas[random2];
      }
      else if(freqArtist == ""){
        console.log("No hay artista frecuente");
        axios.get(direccionIp + "/getArtistasG", {
          params:{
            genre: freqGenre,
          },
        }).then(respuesta => {
          if(respuesta.data[0] == undefined){
            this.personalizedSong();
          }
          else{
            var au = 0;
            for (var i = 0; i < respuesta.data.length; i++) {
              posibleArtists.push(respuesta.data[i]);
              au = i;
            }
            if(au >= 1){
              var random3 = Math.floor(Math.random() * posibleArtists.length);
              freqArtist = posibleArtists[random3];
            }
            else {
              freqArtist = "";
            }
          }
        });
      }
    });
  });
}

```

```

    }
    else if(freqGenre == ""){
        console.log("No hay genero frecuente");
        axios.get(direccionIp + "/getGenerosA", {
            params:{
                artist: freqArtist,
            },
        }).then(respuesta => {
            for (var i = 0; i < respuesta.data.length; i++) {
                posibleGenres.push(respuesta.data[i]);
            }
        });
        var random4 = Math.floor(Math.random() * posibleGenres.length);
        freqGenre = posibleGenres[random4];
    }
    if(this.art != ""){
        this.artist = freqArtist;
    }
    else{
        this.artist = "";
        this.art = "";
    }
    this.genre = freqGenre;
    console.log("GENERO FRECUENTE: " + this.genre);
    console.log("ARTISTA FRECUENTE: " + this.artist);
    axios.get(direccionIp + "/getPersonalizedSongs", {
        params:{
            genre: this.genre,
            artist: this.artist,
            dni: this.dni,
        },
    }).then(response => {
        if(response.data[0] == "No hay canciones que cumplan estos criterios de búsqueda."){
            alert(response.data[0]);
        }else{
            if(response.data[0] == undefined){
                this.personalizedSong();
            }
            else{
                var valida = false;
                do{
                    var randomm = Math.floor(Math.random() * response.data.length);
                    var inFav = false;
                    var inHated = false;
                    for(var x = 0; x < this.favs; x++){
                        if(response.data[randomm].name == this.favs[x].name){
                            inFav = true;
                            break;
                        }
                    }
                    if(inFav == false){
                        for(var y = 0; y < this.hateds; y++){
                            if(response.data[randomm].name == this.hateds[y].name){
                                inHated = true;
                                break;
                            }
                        }
                    }
                    if(inFav == false && inHated == false){
                        this.songs.push(response.data[randomm]);
                        valida = true;
                    }
                } while(valida == false);
                this.songs[0].color = this.colors[Math.floor(Math.random() * this.colors.length)];
                this.songs[0].iconF = 'mdi-heart-outline';
                this.songs[0].iconD = 'mdi-thumb-down-outline';
            }
        }
    });
    this.type = [];
    this.artist = "Cualquiera";
    this.busqueda = "";
    this.genre = "Cualquiera";
}, 100);
}, 200);
console.log("Salimos de la función PERSONALIZED SONG");
},

```

5.4. ALGORITMO DE RECOMENDACIÓN POR FILTRADO COLABORATIVO.

Este algoritmo es con diferencia el más complejo de todos, y por este mismo motivo, y dado que es la parte más importante de este trabajo, nos vamos a detener bastante más en él. Tuve que modificar la base de datos y crear una nueva con los usuarios y sus canciones favoritas. Para poder crear este algoritmo, tuve que realizar una investigación ya que en el primer momento desconocía el funcionamiento de estos algoritmos. Para poder crearlo, me creé un pequeño excel en el que intento explicar el funcionamiento:

- Primero, recojo la lista de canciones favoritas del usuario principal, y con ellas, recojo a los vecinos (los usuarios que tengan como canciones favoritas alguna de las que tiene el usuario principal).

	Usuario Principal	Usuario1	Usuario2	Usuario3	Usuario4	Usuario5	Usuario6	Usuario7	Usuario8
Canción 1									
Canción 2									
Canción 3									
Canción 4									
Canción 5									
Canción 6									
Canción 7									
Canción 8									
Canción 9									
Canción 10									
Canción 11									
Canción 12									
Canción 13									
Canción 14									
Canción 15									
Nº de canciones que gustan similares a las del usuario principal		6	7	5	5	4	6	6	6

- A continuación, depuramos los vecinos, y contamos el número de canciones favoritas no coincidentes con las del usuario principal. Seleccionamos a los que no difieran mucho, ni por encima ni por debajo.

	Cogemos los mas relevantes (MAX. 5)					
	Usuario Principal	Usuario1	Usuario2	Usuario6	Usuario7	Usuario8
Canción 1						
Canción 2						
Canción 3						
Canción 4						
Canción 5						
Canción 6						
Canción 7						
Canción 8						
Canción 9						
Canción 10						
Canción 11						
Canción 12						
Canción 13						
Canción 14						
Canción 15						
Nº Canciones diferentes de us principal que gusten (NO las de us principal en rojo)		2	1	2	0	4

- Finalmente, hacemos un recuento de las canciones seleccionadas como "Más recomendadas para el usuario principal", y mostramos por pantalla como máximo las 5 más recomendadas, pero si pulsan sobre el botón "Mostrar Más", podrán ver todas las recomendaciones.

	Descartamos los que no difieren, y los que difieren por mucho				
	Usuario Principal	Usuario1	Usuario2	Usuario6	REPETICIONES
Canción 1					
Canción 2					
Canción 3					
Canción 4					
Canción 5					
Canción 6					2
Canción 7					2
Canción 8					
Canción 9					
Canción 10					
Canción 11					
Canción 12					
Canción 13					
Canción 14					1
Canción 15					
CANCIONES RECOMENDADAS	6 y 7				

Una vez explicado el algoritmo en papel, comenzamos a programar el algoritmo. Lo primero que hacemos tras entrar en la función **colaborativeFilter**, es obtener una lista con las canciones favoritas del usuario actual por medio de la función **getFavs**.

```

/***** GET FAVORITAS *****/
app.post("/getFavs", function (req, res) {
  console.log("ENTRO en /getFavs\n\n");
  var favoritas = [];
  var dni = req.body.dni;
  var query = "MATCH (u:UserName)-[rel:LIKES]->(c:Canciones) WHERE u.dni='" + dni + "' ";
  query += "return c.name, c.artist, c.genre, c.cover, c.preview, c.danceability, c.energy, c.popularity, c.valence";
  const session = driver.session();
  const resultPromise = session.run(query);
  resultPromise.then(result => {
    if (result.records.length == 0) {
      res.json({
        msg: 'Error'
      })
    }
    else {
      for(var i = 0; i < result.records.length; i++){
        var cancion = {
          name: result.records[i]._fields[0],
          artist: result.records[i]._fields[1],
          genre: result.records[i]._fields[2],
          cover: result.records[i]._fields[3],
          preview: result.records[i]._fields[4],
          danceability: result.records[i]._fields[5],
          energy: result.records[i]._fields[6],
          popularity: result.records[i]._fields[7],
          valence: result.records[i]._fields[8],
        }
        console.log(cancion);
        favoritas.push(cancion);
      }
      res.send(favoritas)
    }
    session.close();
  })
  .catch((error) => {
    res.json({
      msg: 'Error'
    });
    console.log(error)
    session.close();
  })
  console.log("SALGO de /getFavs\n\n");
});

```

Función getFavs (Backend)

```

/***** GET FAVS *****/
getFavs() {
  this.favs = [];
  console.log("ESTOY en GETFAVS");
  axios.post(direccionIp + "/getFavs",{
    dni: this.dni,
  }).then(respuesta => {
    var json = {msg: 'Error'};
    if(JSON.stringify(respuesta.data) == JSON.stringify(json)){
      console.log('No tienes canciones favoritas.')
    }else{
      this.favs = respuesta.data;
    }
  });
},

```

Función getFavs (Frontend)

Una vez tenemos las canciones favoritas, buscamos a los vecinos, es decir, usuarios a los que les gustan las mismas canciones que al usuario Actual. Para esto, llamamos a la función **buscarUsuariosVecinos**.

```

/***** BUSCAR USUARIOS VECINOS *****/
buscarUsuariosVecinos(){
  for(var i = 0; i < this.favs.length; i++){
    axios.post(direccionIp + "/getNeighbours", {
      name: this.favs[i].name,
    }).then((response) => {
      var mensaje = {msg: 'Error, no hay usuarios vecinos!'};
      if(JSON.stringify(response.data[0]) == JSON.stringify(mensaje)){
        alert('Ups, se detectó un error, no hay usuarios vecinos.')
      }else{
        for(var j = 0; j < response.data.length; j++){
          this.usuariosVecinos.push(response.data[j]);
        }
      }
    })
    .catch((error) => {
      console.log(error);
    });
  }
},

```

Función buscarUsuariosVecinos (Frontend)

En esta función, para cada canción de la lista de favoritas, buscamos los vecinos por medio de **axios** y llamando a **getNeighbours**.

```

/***** GET NEIGHBOURS *****/
app.post("/getNeighbours", function(req, res) {
  console.log("ENTRO en Detectar Vecinos");
  const session = driver.session();
  console.log(req.body.name);
  var vecinos = [];
  var query = "MATCH (u:UserName), (u)-[:LIKES]->(c) WHERE c.name='" + req.body.name + "' RETURN u.dni";
  const resultPromise = session.run(query);
  resultPromise.then(result => {
    if (result.records.length == 0) {
      res.json({
        msg: 'Error, no hay usuarios vecinos'
      })
    }
    else {
      for(var i = 0; i < result.records.length; i++){
        var usuario = {
          name: result.records[i]._fields[0],
        }
        vecinos.push(result.records[i]._fields[0]);
      }
      res.send(vecinos)
    }
    session.close();
  })
  .catch((error) => {
    res.json({
      msg: 'Error, en getNeighbours'
    });
    console.log(error)
    session.close();
  })
});

```

Función getNeighbours (Backend)

Una vez tenemos la lista con todos los vecinos, depuramos los resultados obtenidos y nos quedamos con los que han aparecido más veces, por medio de la función **depurarUsuariosVecinos**. En esta función, recorremos la lista de usuarios, y buscamos todas sus coincidencias añadiéndolos a una nueva lista si aparece al menos 2 veces.

```

/***** DEPURAR USUARIOS VECINOS *****/
depurarUsuariosVecinos(){
  console.log("VECINOS DEPURADOS");
  for(var i = 0; i < this.usuariosVecinos.length; i++){
    var aux = 0;
    for(var j = 0; j < this.usuariosVecinos.length; j++){
      if(this.usuariosVecinos[i] == this.usuariosVecinos[j]){
        aux++;
      }
    }
    //Buscamos vecinos con al menos 2 coincidencias
    if(aux >= 2){
      var existe = false;
      if(this.vecinosDepurados.length > 0){
        for(var k = 0; k < this.vecinosDepurados.length; k++){
          if(this.vecinosDepurados[k] == this.usuariosVecinos[i]){
            existe = true;
            break;
          }
        }
      }
      if(existe == false){
        console.log(this.usuariosVecinos[i]);
        this.vecinosDepurados.push(this.usuariosVecinos[i]);
      }
    }else{
      console.log(this.usuariosVecinos[i]);
      this.vecinosDepurados.push(this.usuariosVecinos[i]);
    }
    break;
  }
}

//Eliminamos al propio usuario de la lista, al usuario actual
var borrar = this.vecinosDepurados.indexOf(this.dni);
this.vecinosDepurados.splice( borrar, 1 );
console.log("VECINOS DEPURADOS = "+this.vecinosDepurados.length);
},

```

Función depurarUsuariosVecinos (Frontend)

A continuación, buscamos el número de canciones favoritas que tiene cada usuario, contando solo las que sean diferentes de las que ama y odia el usuario Actual. Esto lo hacemos desde la función **depurarUsuariosPorCanciones**.

```

/***** DEPURAR USUARIOS POR CANCIONES *****/
depurarUsuariosPorCanciones(){
  this.songsNoDepuradas = [];
  for(var i = 0; i < this.vecinosDepurados.length; i++){
    axios.post(direccionIp + "/getSongsColaborativeFilter", {
      usuario: this.vecinosDepurados[i],
      dni: this.dni,
    })
    .then((response) => {
      var mensaje = {
        msg: 'Error, no hay canciones recomendadas por filtrado colaborativo'
      };
      if(JSON.stringify(response.data[0]) == JSON.stringify(mensaje)){
        alert('No hay canciones que mostrar por filtrado colaborativo.')
      } else {
        for(var j = 0; j < response.data.length; j++){
          console.log("USUARIOS DEPURADOS POR CANCIONES " + response.data[j]);
          this.songsNoDepuradas.push(response.data[j]);
        }
      }
    })
    .catch((error) => {
      console.log(error);
    });
  }
},

```

Función depurarUsuariosPorCanciones (Frontend)

En esta función, para cada vecino depurado, obtengo la lista de canciones que le gustan, las cuales no coincidan con canciones que le gustan o que odia el usuario principal, todo esto, llamando a **getSongsColaborativeFilter**.

```

/***** GET SONGS COLABORATIVE FILTER *****/
app.post("/getSongsColaborativeFilter", (req, res) => {
  console.log("ENTRO en /getSongsColaborativeFilter\n\n");
  var usuarioVecino = req.body.usuario;
  var dni = req.body.dni;
  var canciones = [];
  var query = "MATCH (c:Canciones), (u:UserName), (a:UserName), (u)-[:LIKES]->(c)"
    + " WHERE u.dni = '" + usuarioVecino + "' AND a.dni = '" + dni + "' AND NOT (a)-[:LIKES]->(c) AND NOT (a)-[:HATES]->(c) "
    + "RETURN c.name, c.artist, c.genre, c.cover, c.preview, c.danceability, c.energy, c.popularity, c.valence ";
  const session = driver.session();
  const resultPromise = session.run(query);
  resultPromise.then(result => {
    if (result.records.length == 0) {
      res.json({
        msg: 'Error, no hay canciones recomendadas por filtrado colaborativo'
      })
    }
    else {
      console.log(query);
      for (var i=0; i<result.records.length; i++){
        var cancion = {
          name: result.records[i]._fields[0],
          artist: result.records[i]._fields[1],
          genre: result.records[i]._fields[2],
          cover: result.records[i]._fields[3],
          preview: result.records[i]._fields[4],
          danceability: result.records[i]._fields[5],
          energy: result.records[i]._fields[6],
          popularity: result.records[i]._fields[7],
          valence: result.records[i]._fields[8],
        }
        console.log(cancion.name);
        canciones.push(cancion);
      }
      res.send(canciones)
    }
    session.close();
  })
  .catch((error) => {
    res.json({
      msg: 'Error'
    });
    console.log(error)
    session.close();
  })
  console.log("SALGO de /getSongsColaborativeFilter\n\n");
});

```

Función getSongsColaborativeFilter (Backend)

Finalmente, buscamos las canciones que más se repiten entre los usuarios que nos quedan por medio de la función **depurarCancionesRecomendadas**. En ella, recorremos la lista de canciones contando las veces que aparece cada una, y si aparecen 2 veces o mas, las añadimos a una nueva lista, e incluimos como atributo el número de apariciones que tuvo.


```

/***** DEPURAR CANCIONES RECOMENDADAS *****/
depurarCancionesRecomendadas() {
  console.log("Estamos en la función DEPURAR CANCIONES RECOMENDADAS");
  this.songsDepuradas = [];
  for(var i = 0; i < this.songsNoDepuradas.length; i++){
    var aux = 0;
    for(var j = 0; j < this.songsNoDepuradas.length; j++){
      if(this.songsNoDepuradas[i].name == this.songsNoDepuradas[j].name){
        aux++;
      }
    }
    //Buscamos vecinos con al menos 2 coincidencias
    if(aux >= 2){
      var existe = false;
      var cont = 0;
      for(var x = 0; x < this.songsNoDepuradas.length; x++){
        if(this.songsNoDepuradas[i].name == this.songsNoDepuradas[x].name){
          cont++;
        }
      }
      if(this.songsDepuradas.length > 0){
        for(var k = 0; k < this.songsDepuradas.length; k++){
          if(this.songsDepuradas[k].name == this.songsNoDepuradas[i].name){
            existe = true;
            break;
          }
        }
      }
      if(existe == false){
        this.songsNoDepuradas[i]['num'] = cont;
        this.songsDepuradas.push(this.songsNoDepuradas[i]);
      }
    }else{
      this.songsNoDepuradas[i]['num'] = cont;
      this.songsDepuradas.push(this.songsNoDepuradas[i]);
    }
    break;
  }
}
for(var y = 0; y < this.songsDepuradas.length; y++){
  console.log(this.songsDepuradas[y].name+' '+this.songsDepuradas[y].num);
}
console.log("Salimos de la función DEPURAR CANCIONES RECOMENDADAS");
},

```

Función depurarCancionesRecomendadas (Frontend)

Una vez tenemos la lista final de canciones recomendadas para el usuario, las reordenamos por medio de la función **reordenarRecomendacion**, la cual ordena las canciones de más recomendadas (mas apariciones) a menos.

```

/***** REORDENAR RECOMENDACIÓN *****/
reordenarRecomendacion() {
  this.songsDepuradas.sort(function (a, b) {
    if (a.num < b.num) {
      return 1;
    }
    if (a.num > b.num) {
      return -1;
    }
    return 0;
  });
},

```

Función reordenarRecomendacion (Frontend)

```

//===== COLLABORATIVE FILTER =====/
colaborativeFilter: function() {
  console.log("Estamos en la función COLLABORATIVE FILTER Máximo de 5 canciones recomendadas");
  // 1. Sacamos las canciones favoritas del usuario Principal
  var oldFavs = this.favs.length;
  this.getFavs();
  setTimeout(() => {
    console.log(this.favs);
    if(this.favs.length >= 3){
      if(this.favs.length == oldFavs){
        this.num = 0;
        this.songs = [];
        if(this.songsDepuradas.length > 5){
          this.visible = true;
          alert("Mostrando las 5 canciones más recomendadas ordenadas de más a menos recomendadas. Pulsa el botón 'Mostrar Más' para ver todas las canciones recomendadas.");
        }
        else{
          if(this.songsDepuradas.length >= 2){
            alert("Mostrando las canciones más recomendadas ordenadas de más a menos recomendadas. Sigue descubriendo música para obtener más recomendaciones.");
          }
          else{
            alert("No hay canciones recomendadas para ti.");
          }
        }
        //Recomendamos las 5 canciones con más coincidencias
        for(var i = 0; i < this.songsDepuradas.length; i++){
          this.songs.push(this.songsDepuradas[i]);
          this.songs[i].color = this.colors[this.num];
          this.songs[i].iconF = 'mdi-heart-outline';
          this.songs[i].iconD = 'mdi-thumb-down-outline';
          this.songs[i].iconB = 'mdi-thumb-down-outline';
          this.num = this.num + 1;
          if(this.num == this.colors.length){
            break;
          }
        }
      }
    }
    else{
      this.usuariosVecinos = [];
      //2. Buscamos a los vecinos (usuarios que les gustan las mismas canciones que a nosotros)
      this.buscarUsuariosVecinos();
      this.vecinosDepurados = [];
      setTimeout(() => {
        //3. Nos quedamos con los que mas veces aparecen
        this.depurarUsuariosVecinos();
        setTimeout(() => {
          if(this.vecinosDepurados.length < 3){
            alert("No hay suficientes usuarios con gustos similares a los tuyos como para hacer una recomendación.");
          }
          else{
            //4. Buscamos los que tengan un num de canciones favoritas (distintas de las que uno y odia el us principal) similar
            this.depurarUsuariosPorCanciones();
            setTimeout(() => {
              //5. Buscamos las canciones mas coincidentes entre los usuarios
              this.depurarCancionesRecomendadas();
              setTimeout(() => {
                this.songs = [];
                //Reordenamos las canciones
                this.reordenarRecomendacion();
                //Pasamos color a las canciones
                this.num = 0;
                if(this.songsDepuradas.length > 5){
                  this.visible = true;
                  alert("Mostrando las 5 canciones más recomendadas ordenadas de más a menos recomendadas. Pulsa el botón 'Mostrar Más' para ver todas las canciones recomendadas.");
                }
                else{
                  if(this.songsDepuradas.length >= 1){
                    alert("Mostrando las canciones más recomendadas ordenadas de más a menos recomendadas. Sigue descubriendo música para obtener más recomendaciones.");
                  }
                  else{
                    alert("No hay canciones recomendadas para ti.");
                  }
                }
                //Recomendamos las 5 canciones con más coincidencias
                for(var i = 0; i < this.songsDepuradas.length; i++){
                  this.songs.push(this.songsDepuradas[i]);
                  this.songs[i].color = this.colors[this.num];
                  this.songs[i].iconF = 'mdi-heart-outline';
                  this.songs[i].iconD = 'mdi-thumb-down-outline';
                  this.songs[i].iconB = 'mdi-thumb-down-outline';
                  this.num = this.num + 1;
                  if(this.num == this.colors.length){
                    break;
                  }
                }
                //, 200);
                //, 200);
                //, 300);
                //, 300);
              }, 500);
            }
          }
          alert("No tiene suficientes canciones favoritas como para realizar una recomendación personalizada.");
        }, 500);
        console.log("Salimos de la función COLLABORATIVE FILTER");
      }, 500);
    }
  }, 500);
}

```

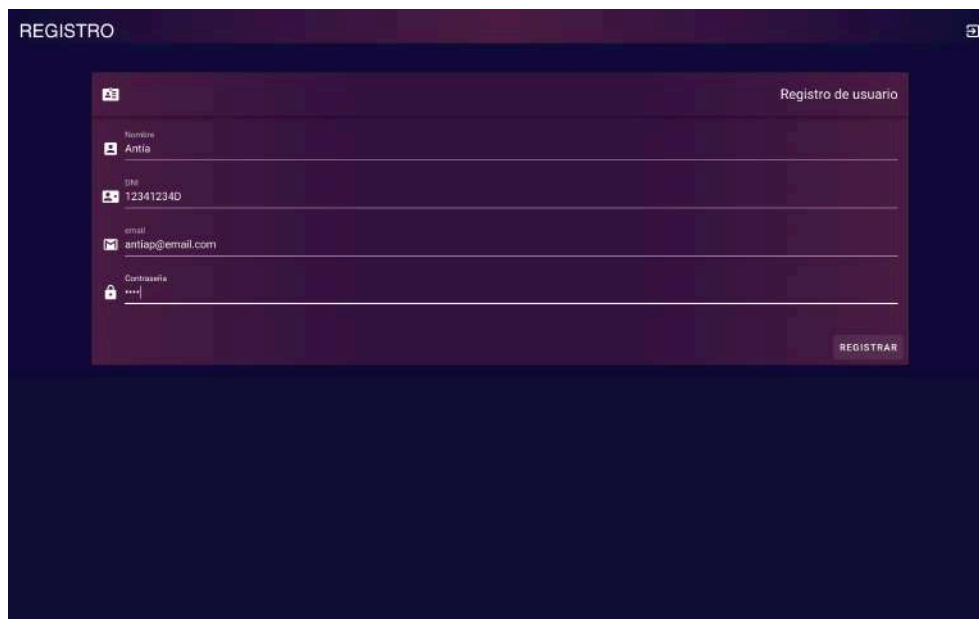
Función collaborativeFilter (Frontend)

6. ANÁLISIS DE RESULTADOS.

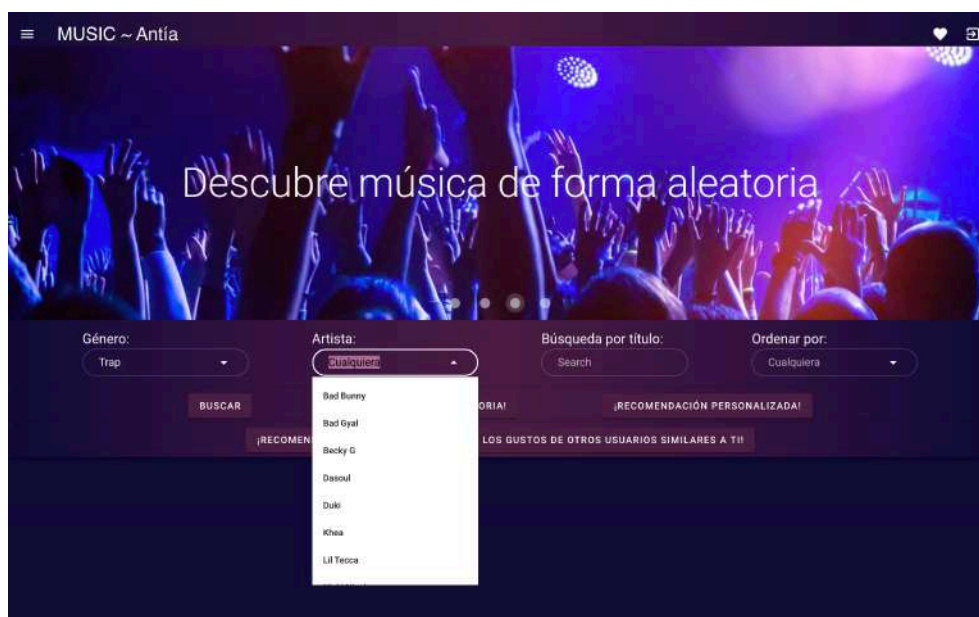
En este apartado, analizaremos los resultados obtenidos por nuestra aplicación partiendo de dos usuarios con gustos completamente distintos. Para ello, crearemos 2 usuarios los cuales navegarán por la aplicación añadiendo a sus listas de favoritos aquellas canciones que más les gustan.

6.1. EJEMPLO 1.

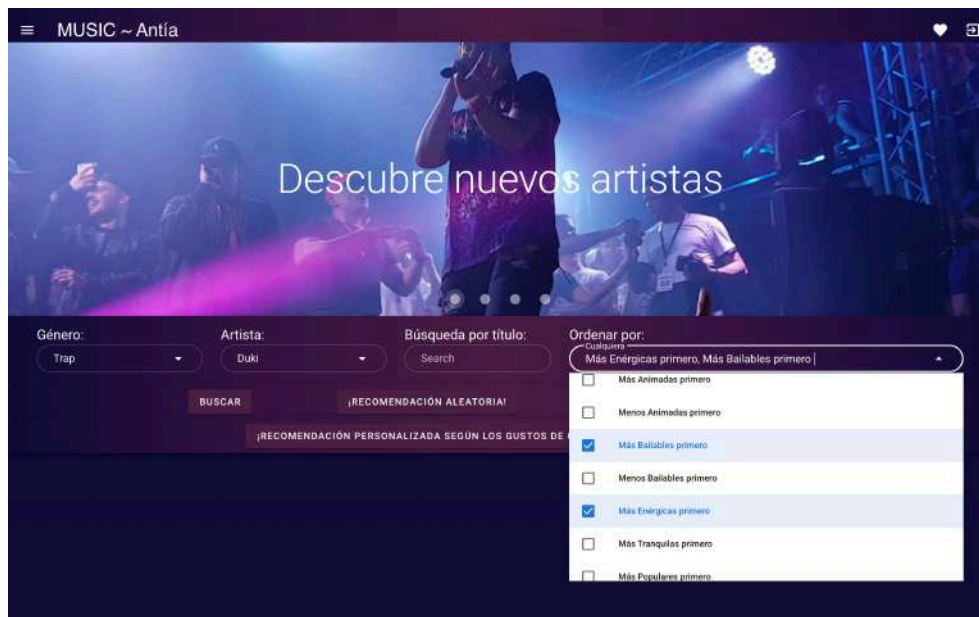
Accedemos a la aplicación, pulsamos sobre registrarse, y creamos un usuario:



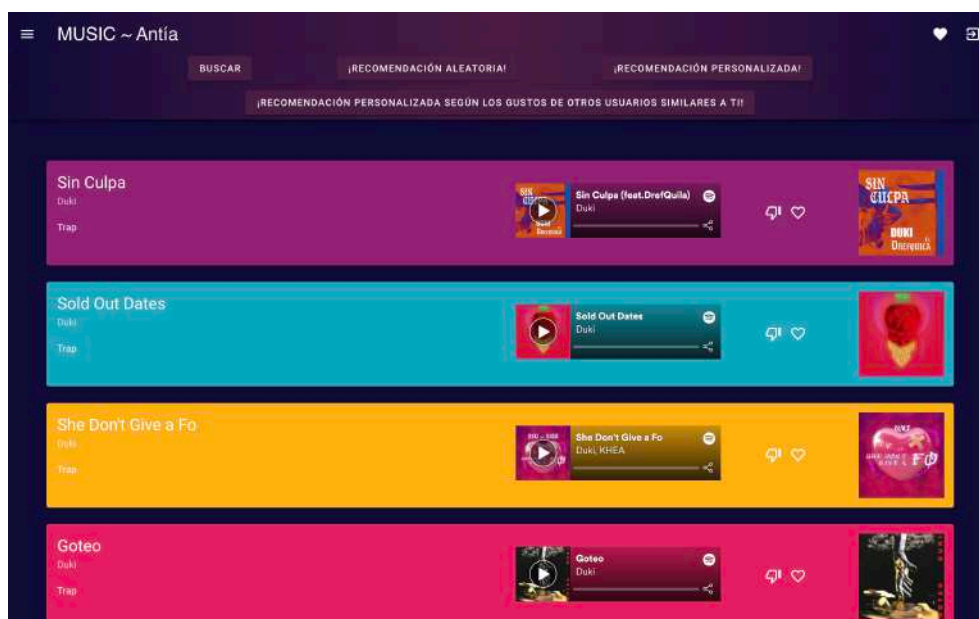
Comenzamos buscando una canción de género “Trap”, y podemos ver que en artistas solo aparecen artistas de este género.



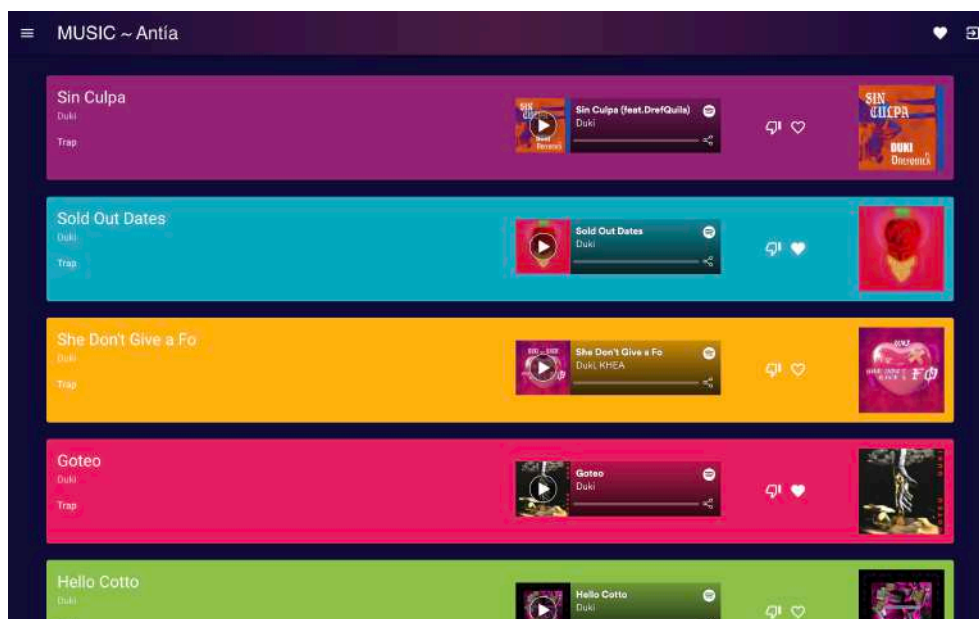
Seleccionamos a “Duki”, y ordenamos los resultados por mas enérgicas y mas bailables primero. Estos serían los parámetros:



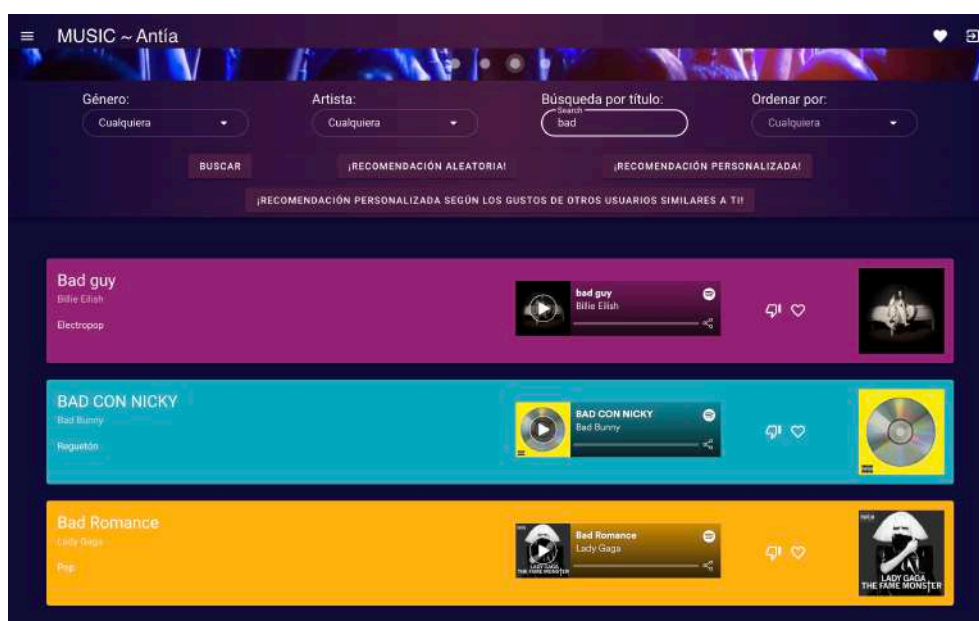
Pulsamos sobre el botón “Buscar”, y obtenemos los siguientes resultados:



Marcamos un par de canciones como favoritas:



Ahora, busquemos por nombre, ponemos “Bad” en el buscador y pulsamos enter, obtenemos los siguientes resultados:



Marcamos la segunda como favorita.

Ahora, buscaremos unas cuantas recomendaciones aleatorias hasta que nos gusten 3 de ellas:

MUSIC ~ Antía

Descubre nuevos artistas

Género: Cualquiera

Artista: Cualquiera

Búsqueda por título: Search

Ordenar por: Cualquiera

BUSCAR

¡RECOMENDACIÓN ALEATORIA!

¡RECOMENDACIÓN PERSONALIZADA!

¡RECOMENDACIÓN PERSONALIZADA SEGÚN LOS GUSTOS DE OTROS USUARIOS SIMILARES A TÍ!

One Thing Right
Marshmello
Electronic

A Un Paso De La Luna
Ana Mena
Pop

FRIENDS
Marshmello, Anne-Marie
Electronic

Ayer Me Llamó Mi Ex
Khea
Trap

Call You Mine
The Chainsmokers
EDM

Cae la Noche
Hard G2
Rap

Como ya tenemos unas cuantas canciones añadidas a favoritos, podemos solicitar una recomendación personalizada, así que en primer lugar, pulsamos el botón de recomendación personalizada en base a mis gustos. (Sin filtrado colaborativo)

Esta, al igual que la recomendación aleatoria, muestra los resultados de uno en uno, por lo que realizaremos esta petición 5 veces seguidas, y de esas, las que nos gusten las añadiremos a favoritos.

MUSIC ~ Antía

Encuentra música adaptada a tus gustos

Género: Cualquiera Artista: Cualquiera Búsqueda por título: Search Ordenar por: Cualquiera

BUSCAR ¡RECOMENDACIÓN ALEATORIA! ¡RECOMENDACIÓN PERSONALIZADA!

¡RECOMENDACIÓN PERSONALIZADA SEGÚN LOS GUSTOS DE OTROS USUARIOS SIMILARES A TI!

Changes
XXX Tentacion
Trap

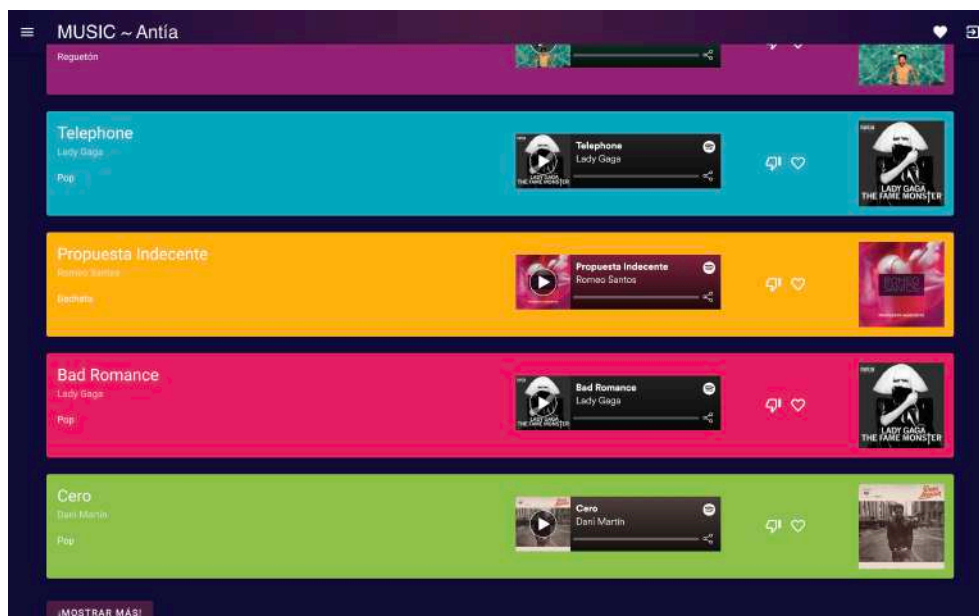
Hitboy
Duki
Trap

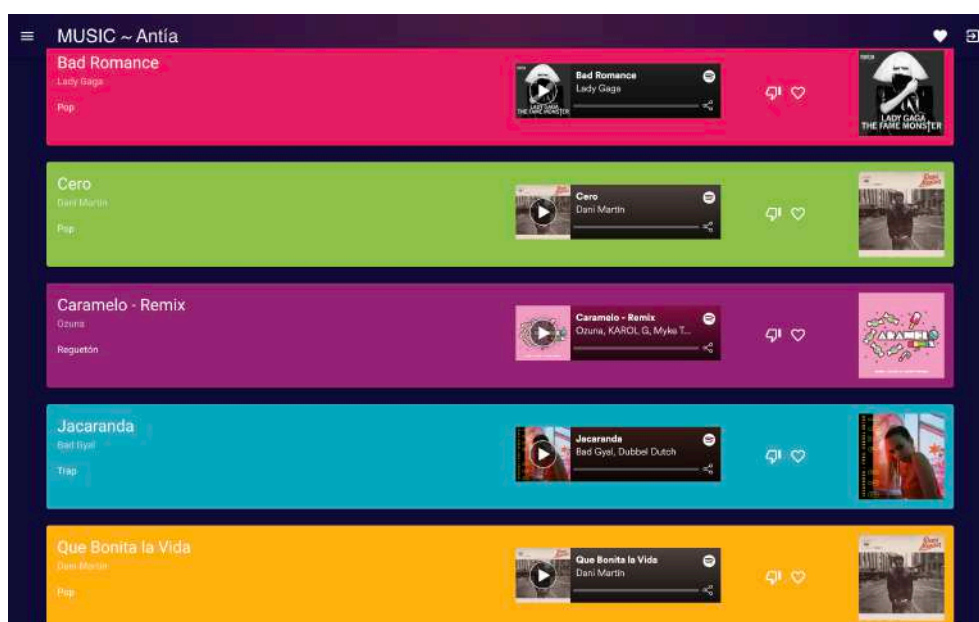
Jacaranda
Bad Gyal
Trap

BENDICIONES
Bad Bunny
Trap

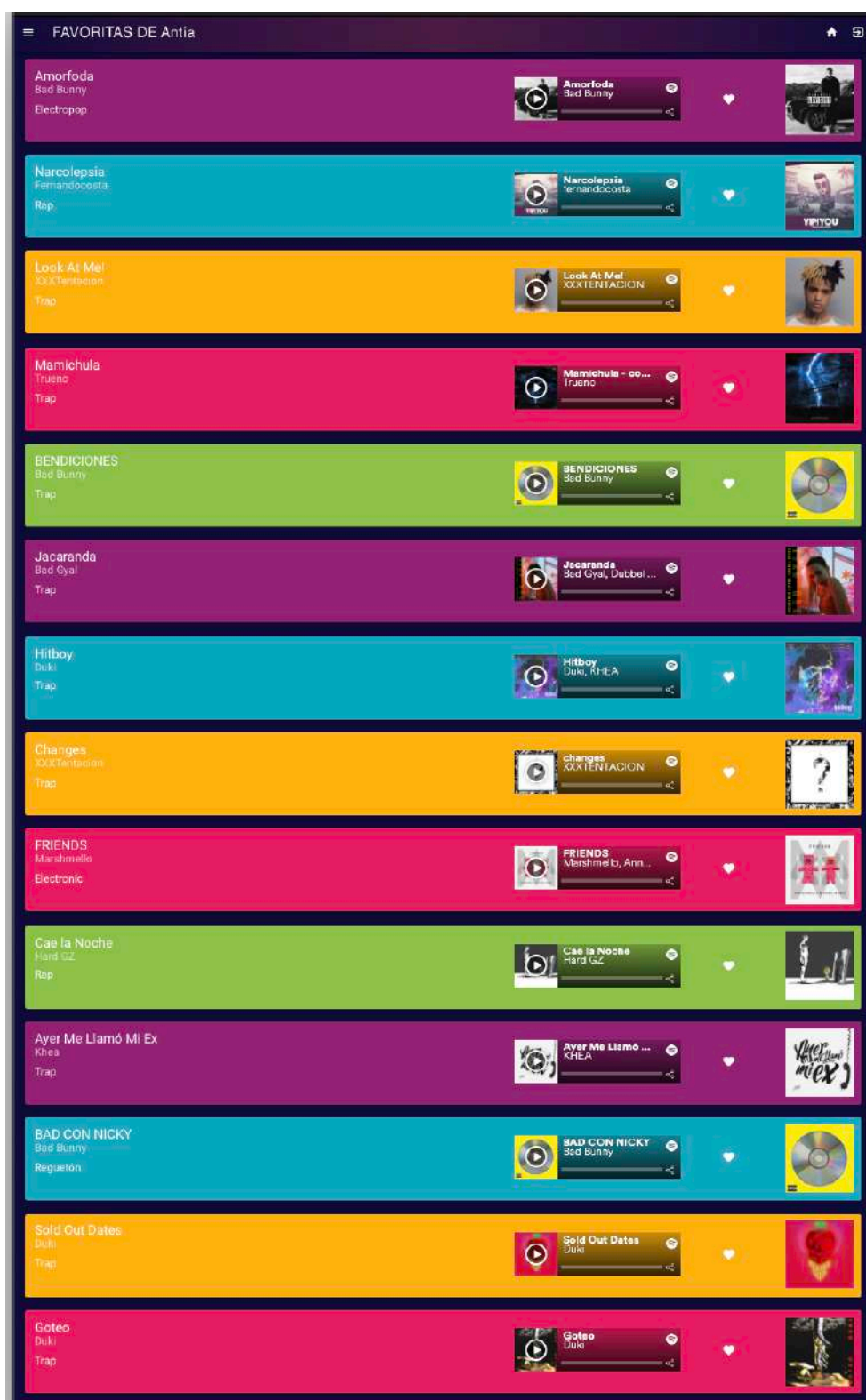
El No Te Da
DASOUL
Trap

Ahora, solicitamos la recomendación personalizada por filtrado colaborativo, que nos muestran 5 resultados, y añadimos a favoritos los que nos han gustado, si hay más de 5 resultados, nos aparecerá el botón de “Mostrar Más”, y si pulsamos sobre el podremos ver ordenados de más a menos recomendadas todas las canciones que :

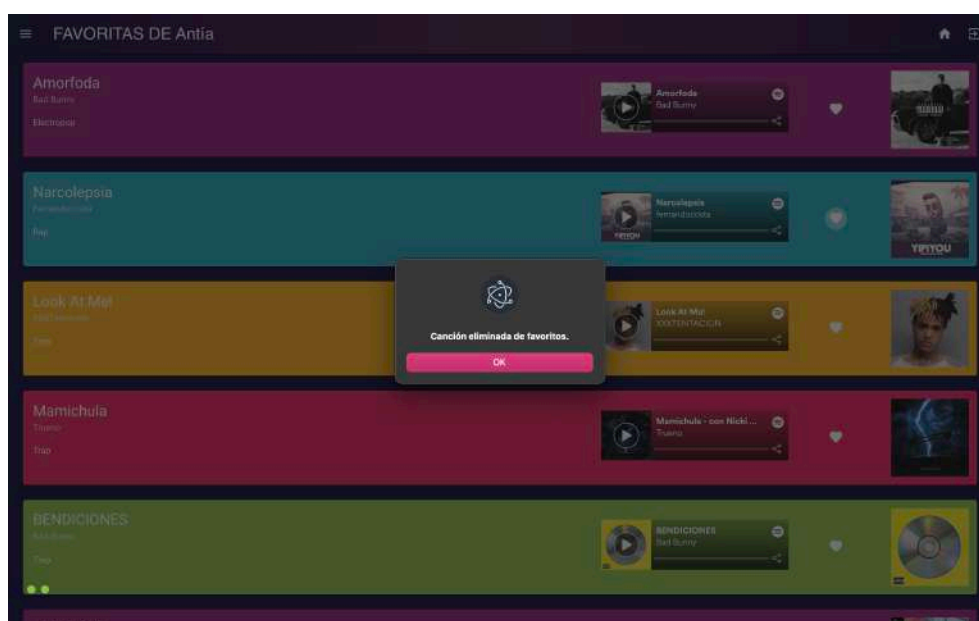




Podemos visualizar nuestra página de favoritos:



Vamos a eliminar una de las canciones de favoritos:



6.2. EJEMPLO 2.

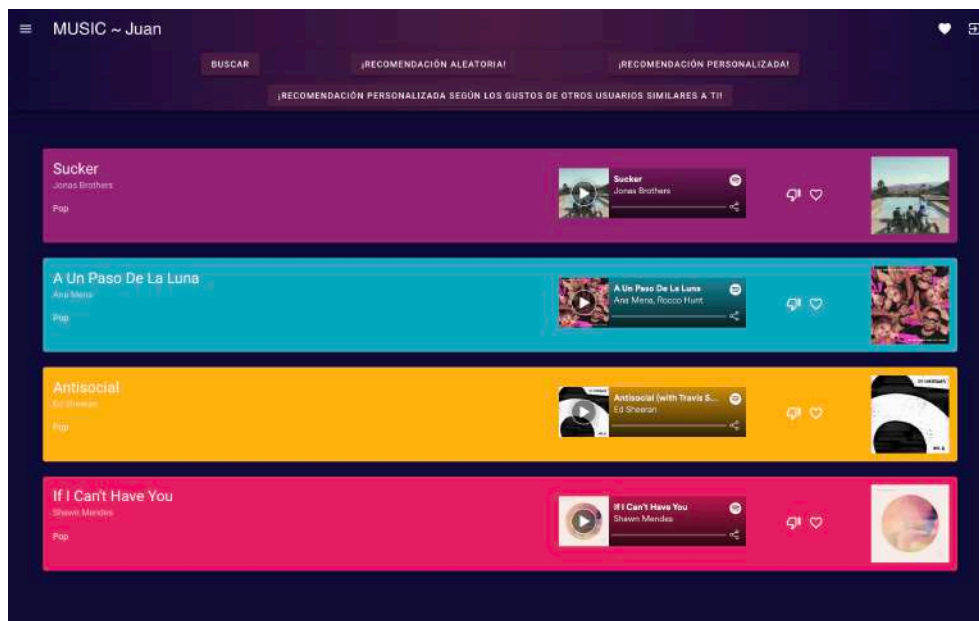
Accedemos a la aplicación, pulsamos sobre registrarse, y creamos un usuario:

The screenshot shows a registration form titled "REGISTRO" in a dark-themed application. The form is titled "Registro de usuario" and contains the following fields: "Nombre" (Name) with the value "Juan", "DNI" (ID Number) with the value "43214322R", "email" with the value "juannc@email.com", and "Contraseña" (Password) with a masked input. A "REGISTRAR" button is located at the bottom right of the form.

Buscamos música de género Pop, sin importar el artista, y las ordenamos por más animadas y más populares primero.

The screenshot shows the music application interface for a user named "Juan". The header displays "MUSIC ~ Juan". The main content area features a banner with the text "Descubre nuevos artistas" and a background image of a band performing. Below the banner, there are search filters: "Género:" (Genre) set to "Pop", "Artista:" (Artist) set to "Cualquiera" (Any), and "Búsqueda por título:" (Search by title) with a "Search" button. A "BUSCAR" button is also present. Below the search filters, there are two buttons: "¡RECOMENDACIÓN ALEATORIA!" and "¡RECOMENDACIÓN PERSONALIZADA SEGÚN LOS GUSTOS DE OTROS!". On the right side, there is a "Ordenar por:" (Sort by) dropdown menu with the following options: "Más Animadas primero, Más Populares primero" (selected), "Más Bailables primero", "Menos Bailables primero", "Más Enérgicas primero", "Más Tranquilas primero", "Más Populares primero" (checked), and "Menos Populares primero".

Obtenemos esto, y vamos añadiendo canciones a favoritos:

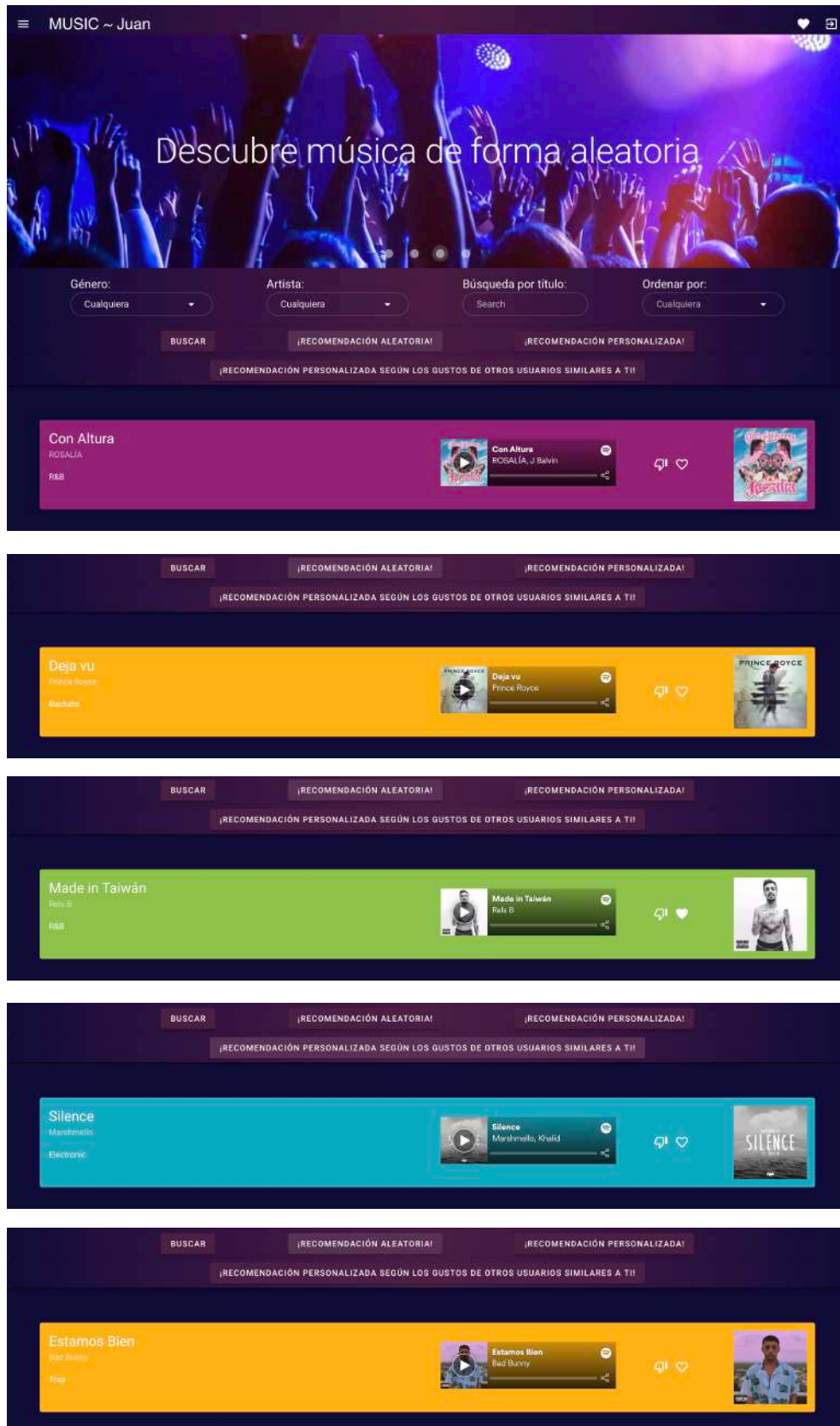


En este caso, al no haber seleccionado a ningún artista, nos salen mas canciones, por lo que fuimos seleccionando algunas de ellas, así que para mostrarlas, las visualizamos desde la página de favoritas:



Como ya tenemos bastantes canciones en la lista de favoritos, vamos a obtener algunas recomendaciones.

En primer lugar, recomendaciones aleatorias, vamos a solicitar 5, y marcamos las que nos gustan como favoritas:



Ahora solicitamos 5 recomendaciones personalizadas, y marcamos las que nos gusten como favoritas:



Finalmente, solicitamos la recomendación personalizada por filtrado colaborativo, obtenemos, en principio, las 5 más recomendadas, que se corresponden con los siguientes resultados, y marcamos las que nos han gustado:



7. DAFO.

Análisis DAFO: Debilidades, Amenazas, Fortalezas y Oportunidades.

<p>Debilidades</p> <ul style="list-style-type: none"> - La base de datos no es lo suficientemente grande para poder obtener una recomendación precisa. - Algoritmos con una gran complejidad. 	<p>Amenazas</p> <ul style="list-style-type: none"> - Cualquier persona podría implementar una aplicación similar.
<p>Fortalezas</p> <ul style="list-style-type: none"> - Interfaz bonita e intuitiva. - Varios tipos de recomendación y filtrado con respuestas muy rápidas. 	<p>Oportunidades</p> <ul style="list-style-type: none"> - Base de datos fácilmente ampliable.

- **Debilidades:**
 - Puede que la base de datos utilizada no sea lo suficientemente grande como para poder realizar una recomendación muy precisa. Al no tener un gran número de usuarios con datos diferentes, puede que las predicciones no siempre sean exactas, pero sin duda alguna, nos sirve para una primera toma de contacto con este tipo de algoritmos, y con el mundo de los datos.
 - Los algoritmos de filtrado colaborativo tienen una gran complejidad ya que procesan una gran cantidad de datos, por tanto, pueden provocar que el procesamiento de la información, y por tanto la respuesta, sea algo lento.
- **Amenazas:**
 - Cualquier persona podría implementar una aplicación similar ya que los datos los podemos obtener de internet.
- **Fortalezas:**
 - La interfaz es bonita e intuitiva, lo que hace que la aplicación sea muy sencilla de usar.
 - Ofrece recomendaciones personalizadas y completamente gratuitas.
 - La aplicación ofrece varios tipos de recomendación y de búsqueda diferentes, que obtienen las respuestas rápidamente.
- **Oportunidades:**
 - La base de datos se puede ampliar de forma muy sencilla gracias al programa creado en python. Simplemente necesitamos crear una playlist de Spotify y por medio de su identificador, podremos obtener todos los datos, exceptuando el género musical al que pertenece. (Por ello, una de las ideas era asignar el género musical al artista, pero un artista puede tener canciones que pertenezcan a distintos géneros).

8. LÍNEAS DE FUTURO.

El sector de la música es un sector que está en continuo crecimiento, y sí que es cierto, que hay aplicaciones como Spotify que además de permitirte reproducir la música, también se encargan de recomendarte canciones que podrían gustarte.

Al ser un tema que me gusta, no me importaría seguir invirtiendo tiempo en ampliar y mejorar las funcionalidades de esta aplicación.

Lo primero que me gustaría implementar es una mejora de los algoritmos de recomendación, ya que a pesar de que están bastante logrados, y teniendo en cuenta que es la parte que más trabajo me llevó a la hora de implementar la aplicación, sé que se pueden mejorar para proporcionar así una experiencia única al usuario, junto con algún método de codificación de datos.

Otro de los puntos que me gustaría incluir es el hecho de poder almacenar el historial de búsquedas, y poder mostrarle al usuario que es lo que buscó recientemente. Aunque la interfaz está creada, y mi idea principal era poder hacer alguna recomendación en función de las búsquedas del usuario, finalmente tuve que descartar esta idea ya que todos los datos que uso los voy solicitando según los necesito a la base de datos, y no tenía claro como poder almacenar este historial en ella.

No me importaría añadir el apartado de editar perfil para poder cambiar los datos, incluso llegando a insertar una foto de perfil, añadir una descripción, gustos musicales, y demás datos, y dar también la opción de poder buscar y seguir a otros usuarios.

Sin duda alguna, otra de las ideas que me gustaría incluir en esta aplicación sería el crear un reproductor de música mediante el cual pudiésemos reproducir las canciones completas, y en el que pudiésemos crear también listas de reproducción para así poder usar la aplicación como si fuese Spotify.

9. LECCIONES APRENDIDAS.

Esta asignatura me ayudó a demostrarme a mi misma que, ahora mas que nunca, si quiero hacer algo lo voy a conseguir, por muy perdida que esté inicialmente, soy capaz de hacer mucho más de lo que creo, y que si me propongo algo lo voy a conseguir. Al final es como todo, y siempre depende del tiempo y las ganas que inviertas en ello, pero si me llegan a decir en septiembre que en noviembre tendría implementada esta aplicación y con un sistema de recomendación por filtrado colaborativo, estoy segura de que no me lo creería.

En cuanto a aprender cosas nuevas por mi cuenta, siempre fui una persona bastante autodidacta, por lo que en cuanto a este tema sabía que no me toparía con muchas dificultades. A pesar de esto, he de reconocer que el mundo de las bases de datos no era mi mayor fuerte ya que no me apasionan, pero este proyecto me ha dado la oportunidad de conocer una nueva forma de gestionar los datos y de crear bases de datos. Todo esto, aprendiendo a trabajar con un tipo de bases de datos completamente nuevo y que tiene un lenguaje propio.

También, me ayudó a proponerme nuevos retos, y a intentar llegar un poco mas lejos de lo que inicialmente pensaba que podría, a atreverme a manejar tecnologías y lenguajes de los que igual no tenía la mayor práctica del mundo, pero con los que ahora mismo puedo asegurar que me defiendo sin mayor problema. Este proyecto exige mucho trabajo, ya que desde el minuto uno debes trabajar por tu cuenta, investigando las herramientas, los lenguajes, el funcionamiento, etc.

10. GUÍA DE INSTALACIÓN DE LA APLICACIÓN.

Para poder instalarnos la aplicación, lo primero que debemos hacer es tener instalado NEO4J para poder ejecutar la base de datos. Si no lo tenemos instalado, lo podemos descargar desde la página oficial de NEO4J que os dejo en el siguiente [enlace](#). Una vez instalado, debemos crear un nuevo proyecto en Neo4J pulsando sobre “+ New”. A continuación, añadimos una Base de datos de tipo grafo pulsando sobre “+ Add Database” y a continuación sobre “Create a Local Database”. Le ponemos nombre y contraseña, y pulsamos en los puntos de la derecha y le damos a “Manage”. Pulsamos sobre la flecha de “Open Folder” y pulsamos en “Import”.

Una vez tenemos abierta la carpeta “Import”, debemos descargarnos las bases de datos, que se encuentran en mi GitHub (baseDatosMusica.csv y datosUsuarios.csv), y las arrastramos hasta esa carpeta nuestros archivos .csv, e iniciamos la base de datos pulsando sobre “Start” y finalmente, abrimos el browser de NEO4J y ejecutamos la query dada.

Ejecutamos en primer lugar la de baseDatosMusica.csv, y a continuación la de datosUsuarios.csv.

```
LOAD CSV WITH HEADERS FROM 'file:///baseDatosMusica.csv' AS line WITH
line LIMIT 1000

MERGE (s:Style {name: "Estilos Musicales"})

MERGE (c:Canciones { name: line.TrackName, artist: line.ArtistName,
genre: line.Genre, energy: toInteger(line.Energy), danceability:
toInteger(line.Danceability), valence: toInteger(line.Valence),
popularity: toInteger(line.Popularity), cover: line.TrackCover,
preview: line.TrackPreview})

FOREACH (n IN (CASE WHEN line.ArtistName IS NULL THEN [] ELSE [1] END)
| MERGE (a:ArtistName {name: line.ArtistName})

    MERGE (a)-[:MADE]->(c)

    FOREACH (m IN (CASE WHEN line.Genre IS NULL THEN [] ELSE [1] END)
    | MERGE (g:Genre {name: line.Genre})

        MERGE (g)-[:GENRE_SINGED]->(a)

        MERGE (s)-[:IS_STYLE]->(g)

    )

)
```

Sentencia para la creación de la base de datos de música

```

LOAD CSV WITH HEADERS FROM 'file:///datosUsuarios.csv' AS line WITH
line LIMIT 1000

MERGE (r:Users {name: "Usuarios"})

FOREACH (n IN (CASE WHEN line.UserName IS NULL THEN [] ELSE [1] END) |
MERGE (u:UserName {name: line.UserName, dni: line.Dni, passw:
line.Passw, email: line.Email})

    MERGE(t:Canciones {name:line.TrackName})

    MERGE (u)-[:LIKES]->(t)

    MERGE (u)-[:IS_USER]->(r)

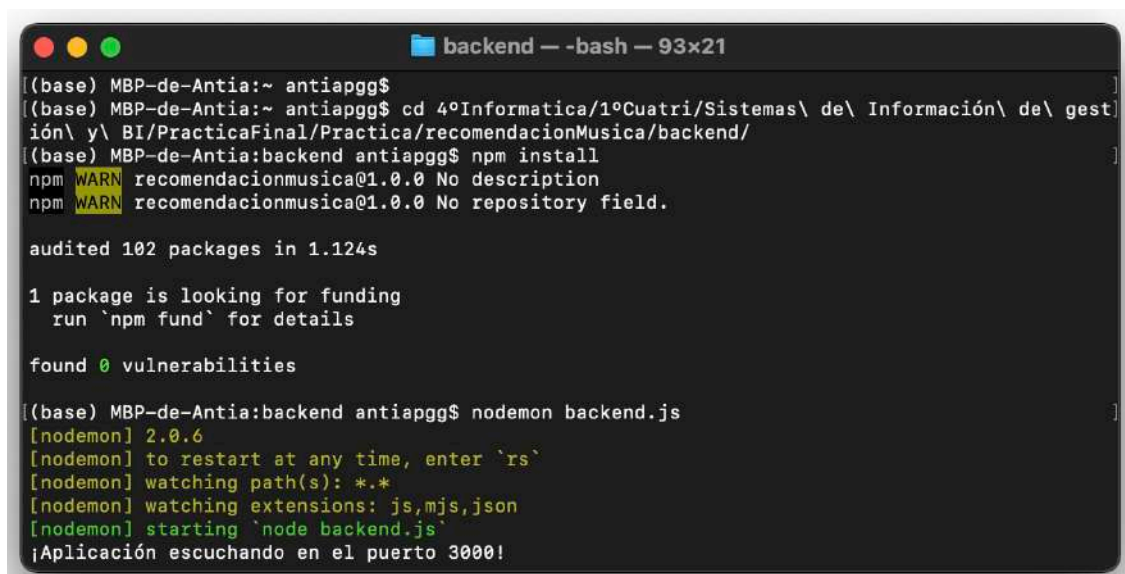
)

```

Sentencia para la creación de la base de datos de usuarios

Ya tenemos la base de datos instalada y funcionando, así que el siguiente paso, es descargar las carpetas de backend y frontend del enlace del [GitHub](#). Debemos saber que tanto el backend como el frontend se ejecutarán en dos terminales diferentes.

Comenzamos con el backend. Abrimos la primera terminal, y con ayuda del comando “cd” accedemos al directorio donde se encuentra la carpeta backend. A continuación, ejecutamos el comando “*npm install*” para instalar todas las dependencias necesarias para poder ejecutar nuestra aplicación. Finalmente, ejecutamos el comando “*nodemon backend.js*” para iniciar nuestro backend.



```

backend — -bash — 93x21
[(base) MBP-de-Antia:~ antiapgg$
[(base) MBP-de-Antia:~ antiapgg$ cd 4ºInformatica/1ºCuatri/Sistemas\ de\ Información\ de\ gest
ión\ y\ BI/PracticaFinal/Practica/recomendacionMusica/backend/
[(base) MBP-de-Antia:backend antiapgg$ npm install
npm WARN recomendacionmusica@1.0.0 No description
npm WARN recomendacionmusica@1.0.0 No repository field.

audited 102 packages in 1.124s

1 package is looking for funding
  run `npm fund` for details

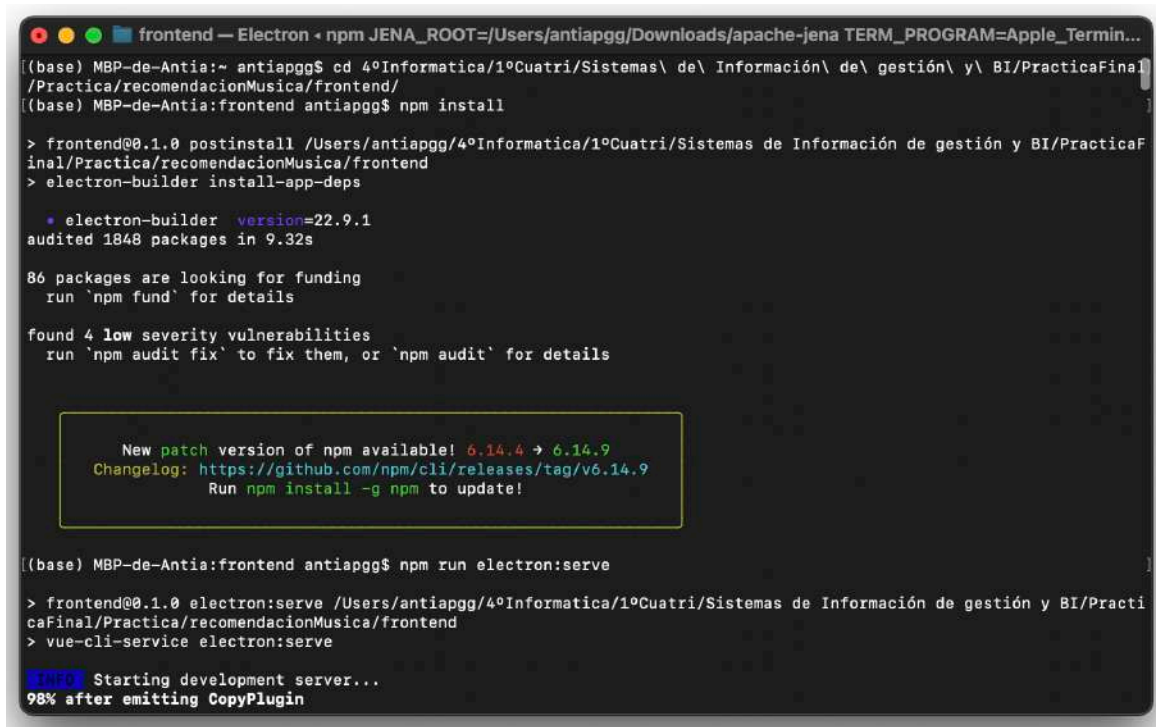
found 0 vulnerabilities

[(base) MBP-de-Antia:backend antiapgg$ nodemon backend.js
[nodemon] 2.0.6
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node backend.js`
¡Aplicación escuchando en el puerto 3000!

```

Activación del Backend

A continuación, accedemos de igual forma al directorio donde se encuentra el frontend. Una vez allí, instalamos todas las dependencias con el comando “*npm install*” y en mi caso, instalé electron para transformar la aplicación web en una aplicación instalable, por tanto, para poder ejecutarlo de esta forma, ejecutamos el comando “*npm run electron:serve*” como se muestra en la imagen. Si no tuviésemos este plugin, la aplicación se ejecutaría con el comando “*npm run serve*”.

A terminal window titled "frontend — Electron · npm JENA_ROOT=/Users/antiapgg/Downloads/apache-jena TERM_PROGRAM=Apple_Termin..." showing the execution of npm commands. The user navigates to the frontend directory and runs 'npm install', which installs electron-builder and updates dependencies. A message indicates a new patch version of npm is available (6.14.4 to 6.14.9). Then, the user runs 'npm run electron:serve', which starts the development server for the electron application.

```
[(base) MBP-de-Antia:~ antiapgg$ cd 4ºInformativa/1ºCuatri/Sistemas\ de\ Información\ de\ gestión\ y\ BI/PracticaFinal/Practica/recomendacionMusica/frontend/
[(base) MBP-de-Antia:frontend antiapgg$ npm install

> frontend@0.1.0 postinstall /Users/antiapgg/4ºInformativa/1ºCuatri/Sistemas de Información de gestión y BI/PracticaFinal/Practica/recomendacionMusica/frontend
> electron-builder install-app-deps

  * electron-builder version=22.9.1
  audited 1848 packages in 9.32s

  86 packages are looking for funding
    run `npm fund` for details

  found 4 low severity vulnerabilities
    run `npm audit fix` to fix them, or `npm audit` for details

  New patch version of npm available! 6.14.4 → 6.14.9
  Changelog: https://github.com/npm/cli/releases/tag/v6.14.9
  Run npm install -g npm to update!

[(base) MBP-de-Antia:frontend antiapgg$ npm run electron:serve

> frontend@0.1.0 electron:serve /Users/antiapgg/4ºInformativa/1ºCuatri/Sistemas de Información de gestión y BI/PracticaFinal/Practica/recomendacionMusica/frontend
> vue-cli-service electron:serve

[INFO] Starting development server...
98% after emitting CopyPlugin
```

Activación del Frontend

Ya tenemos nuestra aplicación lista para usar.

11. REFERENCIAS.

1. Kaggle: <https://www.kaggle.com/datasets>
2. Spotipy: <https://github.com/plamere/spotipy>
3. Spotify: <https://www.spotify.com>
4. Desarrollador Spotify: <https://developer.spotify.com>
5. Pandas: <https://pandas.pydata.org>
6. Curso Básico Neo4J: <https://neo4j.com/graphacademy/>
7. Guía Cypher Andrea: <https://github.com/aferns16/Game4U/GuiaCypher>
8. Node.js : <httpS://nodejs.org/es>
9. Express: <https://expressjs.com/es>
10. Driver NEO4J: <https://neo4j.com/javascript-driver/>
11. Vue.js : <https://vuejs.org>
12. Vuetify: <https://vuetifyjs.com/en>
13. Axios: <https://github.com/axios/axios>
14. Electron: <https://electronjs.org>
15. NEO4J: <https://neo4j.com/download>
16. GitHub: <https://github.com/apereg20/SIBI>

12. BIBLIOGRAFÍA Y WEBGRAFÍA.

GitHub del Proyecto:

<https://github.com/apereg20/SIBI>

Aprendiendo a usar Neo4J:

<http://blog.jortilles.com/neo4j/>

<https://sandbox.neo4j.com>

<https://www.diegocalvo.es/tutorial-neo4j-espanol/>

Neo4J y Javascript:

<https://neo4j.com/developer/javascript/>

Información sobre librería Spotipy:

<https://spotipy.readthedocs.io/en/2.16.1/>

Idea de filtrado colaborativo:

https://e-archivo.uc3m.es/bitstream/handle/10016/22845/PFC_alma_collado_sanchez_2014.pdf?sequence=1&isAllowed=y