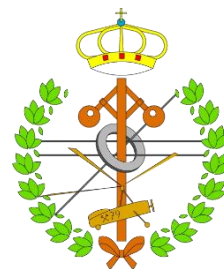




universidad
de león



Escuela de Ingenierías

Industrial, Informática y Aeroespacial

GRADO EN INGENIERÍA INFORMÁTICA

Sistemas de Información de Gestión y Business Intelligence (SIBI)

Aplicación MusicFactory

0. Índice

1. INTRODUCCIÓN	3
2. DESCRIPCIÓN DEL PROBLEMA.....	5
2.1.MI SOLUCIÓN AL PROBLEMA	6
3. HERRAMIENTAS UTILIZADAS.....	7
3.1.HERRAMIENTAS PARA EL DESARROLLO DE LA APLICACIÓN	7
FRONTEND:.....	7
BACKEND:.....	8
3.2.HERRAMIENTAS UTILIZADAS PARA EL DESARROLLO DE LA BASE DE DATOS	8
BASE DE DATOS:.....	¡ERROR! MARCADOR NO DEFINIDO.
3.3.OTRAS HERRAMIENTAS.....	9
4. LA APLICACIÓN MÁS A FONDO	11
4.1. DATASET	11
4.2. BASE DE DATOS	12
4.3. SISTEMA DE INICIO DE SESIÓN Y USUARIOS.....	14
4.4. VISTA HOME PRINCIPAL.....	16

1. Introducción

La aplicación MusicFactory nace por el deseo de descubrir música nueva que nos invade a todos cuando encontramos un nuevo artista, género o álbum y nos enganchamos tanto a ello que queremos más y más hasta que está tan explotado que pasamos a algo nuevo y diferente, cerrando un nuevo ciclo más.

Queremos actuar como ese amigo fanático de la música que todos los días descubre canciones y artistas nuevos y su criterio nos gusta. La idea es aprender del usuario y generar un algoritmo capaz de recomendar algo que con una gran probabilidad y seguridad te guste.

En primer lugar, expondremos el problema que intentamos resolver, las tecnologías utilizadas para la realización de la práctica, la composición, como hemos realizado la base de datos, como la hemos implementado, explicaremos el propio funcionamiento de la aplicación, como utilizarla, la parte del backend, el frontend, base de datos, la interfaz, un ejemplo de usabilidad y una conclusión.

También realizaremos un análisis de los algoritmos de recomendación usados, y analizaremos los resultados obtenidos por nuestro sistema.

2. ¿Quiénes Somos?

Nuestra aplicación de recomendación de música y artistas es una plataforma en línea diseñada para brindar a nuestros usuarios una experiencia de descubrimiento y reproducción de música personalizada y relevante. Utilizamos algoritmos de recomendación avanzados para analizar las preferencias y el historial de reproducción de cada usuario y sugerir canciones y artistas que creemos que disfrutarán.

Nuestro objetivo es ser una herramienta esencial para nuestros usuarios en su búsqueda de nueva música y artistas. Nos esforzamos por ofrecer una amplia selección de géneros y estilos para satisfacer las necesidades de todos nuestros usuarios. Además, trabajamos constantemente para mejorar y actualizar nuestros algoritmos de recomendación para garantizar que siempre estén brindando sugerencias precisas y relevantes para cada usuario.

Nos sentimos muy comprometidos con la comunidad de música y estamos constantemente buscando maneras de apoyar a los artistas emergentes y promover el descubrimiento de nuevo talento. Nos enorgullece poder ser una plataforma que ayude a los artistas a conectarse con nuevos fans y a expandir su base de seguidores.

En resumen, somos una aplicación de recomendación de música y artistas comprometida con ofrecer a nuestros usuarios una experiencia de descubrimiento y reproducción de música personalizada y relevante, mientras apoyamos a los artistas y promovemos el descubrimiento de nuevo talento.

3. Descripción del problema

Los algoritmos de recomendación están ahora mismo ocupando una gran parte del mercado y un buen método para recomendar al usuario el contenido que desea consumir se traduce en un usuario más contento y enganchado al servicio que estamos sirviendo. Las grandes empresas de multimedia compiten día tras día para ofrecer el mejor servicio de recomendación y usabilidad para el usuario. Cada vez nuestra vida es más frenética y la tecnología está ahí para hacernos todo mas fácil y poder realizar tareas en mucho menos tiempo que en el pasado. Por eso cuando nos sentamos en el sofá a disfrutar de Netflix, no queremos pasarnos más de 20min decidiendo qué ver de entre todo su extenso catálogo para que al final cuando nos decidamos por algo, ya no tengamos tiempo para verlo. Con Spotify y la música pasa de la misma forma, todo el mundo escucha música para hacer recados, tareas, trabajos, en el gimnasio, viajes, de camino a sitios etc. Y nadie quiere posponer esas tareas o tardar más en realizarlas por el simple hecho de que no saben qué escuchar, porque no tienen tiempo para dedicar a que artistas han sacado nueva música o que álbumes de música de un género son los mejores y faltan por descubrir.

Está claro que la idea es imitar un poco el comportamiento que tiene Spotify en torno a sus recomendaciones y de hecho utilizaré la API de Spotify para servirme de datos que proporciona para el desarrollo de mi aplicación.

Los algoritmos de recomendación son fundamentales para la plataforma de música en streaming Spotify debido a varias razones. En primer lugar, proporcionan una experiencia de usuario más personalizada y relevante. A través de la recopilación de datos sobre las preferencias y el historial de reproducción de cada usuario, los algoritmos de recomendación pueden sugerir canciones y artistas que sean más propensos a ser disfrutados por cada usuario en particular. Esto permite a Spotify mantener a sus usuarios comprometidos y satisfechos con la plataforma.

Además, los algoritmos de recomendación son cruciales para la monetización de la plataforma. A través de la recomendación de contenido relevante para cada usuario, Spotify puede aumentar la tasa de reproducción de las canciones y, por lo tanto, aumentar la cantidad de dinero que se genera a través de publicidad y suscripciones.

Otra ventaja de los algoritmos de recomendación es su capacidad para descubrir y promover nuevo contenido a los usuarios. Al recomendar canciones y artistas similares a los que ya están siendo reproducidos por un usuario, los algoritmos pueden ayudar a los artistas emergentes a llegar a un público más amplio y a construir su base de seguidores.

En resumen, los algoritmos de recomendación son esenciales para Spotify debido a su capacidad para mejorar la experiencia de usuario, monetizar la plataforma y promover el descubrimiento de nuevo contenido.

3.1. Mi solución al problema

Mi idea es una aplicación en la que un usuario con tan solo dedicarle unos minutos a navegar entre las secciones de la aplicación acabe encontrando un artista nuevo que no conocía del que sacar horas de entretenimiento, o quizás descubrir que le gusta un género que hasta entonces era desconocido.

He desarrollado una aplicación web cliente servidor que es lo que más se ajustaba a lo que pedía la práctica de la asignatura. Tiene un sistema de usuarios con nombre de usuario y contraseña, donde guardar las canciones que eliges favoritas y odiadas. La aplicación se maneja de una forma muy sencilla e intuitiva, solo necesita de unos botones para comenzar a recomendar música y unos filtros para realizar esa búsqueda de una manera mucho más acertada, además de la posibilidad de ordenarlos por determinadas características.

4. Herramientas utilizadas

La solución elegida que mejor se adapta al propósito descrito es una aplicación web. Las aplicaciones web operan bajo el modelo Cliente-Servidor, el cliente ejecuta el código frontend y el servidor el código backend. Para la persistencia de la información se utiliza en este caso una base de datos. Así quedan definidas las tres partes que la componen.

4.1. Herramientas para el desarrollo de la aplicación

En primer lugar, vamos a separar la aplicación en **Frontend** y **Backend**.

4.2. Frontend:

Para el desarrollo del Frontend he utilizado el framework de desarrollo de aplicaciones llamado **Vue**. Vue.js (también llamado Vue o VueJS) es un framework progresivo Javascript para crear interfaces de usuario, o en otras palabras, se trata de una «capa» añadida a Javascript formada por herramientas, convenciones de trabajo y un lenguaje particular que nos permite crear aplicaciones de forma rápida, agradable, sencilla y muy práctica.



Para la parte más estética y visual he utilizado **Vuetify** un framework que combina la potencia del popular VueJS con la estética de Material Design. Permite acelerar el desarrollo de aplicaciones web complejas, incorporando una gran cantidad de componentes “listos para usar”.

Para poder realizar las peticiones HTTP desde el cliente, el cual interactúa con el frontend, al backend, utilicé **Axios**, una librería de JavaScript que nos permite eso.



4.3. Backend:

El backend tiene que ver con la lógica de la aplicación, es decir, contiene todo el código de los algoritmos de recomendación, la lógica correspondiente al tratamiento de los datos y por último el acceso a la persistencia, es decir, el código necesario para la comunicación con la base de datos. Consiste en el conjunto de acciones que pasan dentro de una web, pero que no podemos ver.

Para crear el backend de mi aplicación, usé 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.



4.4. Herramientas utilizadas 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.

4.5. Otras herramientas

Kaggle: Kaggle es una plataforma web que reúne la comunidad Data Science más grande del mundo. Permite acceder de manera gratuita a GPUs y a una gran cantidad de datos y códigos publicados por la comunidad. Los temas que se tratan en Kaggle son muy variados, desde los intentos de predecir la aparición del cáncer con la examinación de fichas de pacientes, hasta el análisis de los sentimientos generados por las críticas de una película.

API de Spotify: Las API son herramientas – librerías – que los desarrolladores de un producto ofrecen a otras compañías para que puedan utilizar sus servicios. Estas compañías no tienen acceso al código fuente, sino que se les pasa documentación de cómo utilizar estas librerías para acceder a sus productos.

Los desarrolladores de estas terceras compañías sólo pueden enviar datos a estos servicios, y esperar que los mismos le devuelvan el resultado de ese proceso. En el caso de Spotify, se ha facilitado a los desarrolladores las herramientas para que puedan construir un reproductor musical (total o parcial) integrado con su aplicación.

OSF: Cuaderno de trabajo en el que llevar el progreso actualizado y un seguimiento que facilita la redacción de la memoria y el almacenamiento del proyecto en su totalidad, así como su posterior publicación.

GitHub: Sistema de control de versiones empleado para subir los resultados finales del proyecto con objeto de que queden visibles para quien puedan ser de ayuda.

Neo4j Desktop: Esencial para la creación de la base de datos, instalación de librerías y lectura de archivos csv junto a su neo4j browser que he usado muchísimo para conformar consultas depurar errores, consultar resultados y hacer pruebas con la base de datos.

5. La aplicación más a fondo

Comenzaré a detallar paso por paso como desarrollé la aplicación, las tecnologías utilizadas en cada momento y las decisiones que tomé, las cuales me llevaron al producto final de MusicFactory.

5.1. Dataset

El primer paso fue elegir el dataset utilizado para completar la base de datos. Este dataset lo encontré en Kaggle. Kaggle es una plataforma web que reúne la comunidad Data Science más grande del mundo. Permite acceder de manera gratuita a GPUs y a una gran cantidad de datos y códigos publicados por la comunidad. Los temas que se tratan en Kaggle son muy variados, desde los intentos de predecir la aparición del cáncer con la examinación de fichas de pacientes, hasta el análisis de los sentimientos generados por las críticas de una película. Mi forma de encontrar el dataset adecuado fue buscar las palabras Spotify base de datos en el buscador y me aparecieron unas cuantas opciones, ahora quedaba elegir la que mejor se ajustara a lo que necesitaba. Aparecieron muchas opciones las cuales a grandes rasgos se diferenciaban en que unas eran datos de canciones en el TOP 200 de Spotify de diferentes y otras simplemente canciones de un género o aleatorias. Lo que no quería era canciones de solo un género o grupo para que los datos fueran lo mas generales posibles y poder llegar al mayor número de usuarios en mi plataforma, con lo cual mi decisión fue optar por bases de datos del TOP 200 de cierto año. Ahora quedaba concretar más mi base de datos, es decir, de las elegidas, me quedé con la que recopilaba los datos de forma mas completa, ya que mi dataset tiene de cada canción lo siguiente:

Node properties	
Canciones	
<id>	44
artist	Bad Bunny
cover	https://i.scdn.co/image/ab67616d0000b27334c8199b0b3b3fb42b8a98a8
danceability	73
energy	57
genre	Reguetón
name	Dakiti
popularity	78
preview	https://open.spotify.com/track/47EiUVwUp4C9fGccaPuUCS
valence	54

ID: Identificador propio del nodo.

Artist: Cantante de la pista.

Cover: Dirección HTTP que te lleva a la portada de la pista.

Danceability: Danzabilidad.

Energy: Energía.

Genre: Género de la pista.

Name: Nombre de la canción.

Popularity: Popularidad.

Preview: Dirección HTTP que te lleva a un reproductor para escuchar la pista.

Valence: Positividad de la pista.

Una vez elegido el dataset, había que transportar esos datos a la base de datos orientada a nodos Neo4J. El primer paso fue abrir la hoja de calculo .CSV en Excel y ver si todos los datos estaban correctos y no hacia falta deshacerse de ninguno o de algún formato que estuviera erróneo, una vez comprobado esto se podía ya por fin usar ese archivo .CSV para realizar la base de datos de Neo4J.

5.2. 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.

La sentencia a correr para la creación de la base de datos es la siguiente:

```
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)
)
```

Lenguaje de consulta Cypher



Una vez realizada, la base de datos debería estar alimentada, al menos de música, pero para completarla necesitamos meter usuarios que interactúen con esa música, usuarios que tengan relaciones del tipo Likes o Hates, para poderles recomendar música de forma personalizada y poder hacer funcionar a la aplicación. Para ello se correrá la siguiente sentencia:

```
LOAD CSV WITH HEADERS FROM 'file:///datosUsuarios.csv' AS line
MERGE (r:Users {name: "Usuarios"})
FOREACH (n IN (CASE WHEN line.Usuario IS NULL THEN [] ELSE [1] END) |
MERGE (u:Usuario {name: line.Usuario, nombre: line.Nombre, email: line.Email, pass
w: line.Passw})
MERGE (t:Canciones {name:line.TrackName})
MERGE (u)-[:LIKES]->(t)
MERGE (u)-[:IS_USER]->(r)
)
```

Para explicar la base de datos de una forma más detallada voy a ir explicando los nodos y sus relaciones.

Nodos:

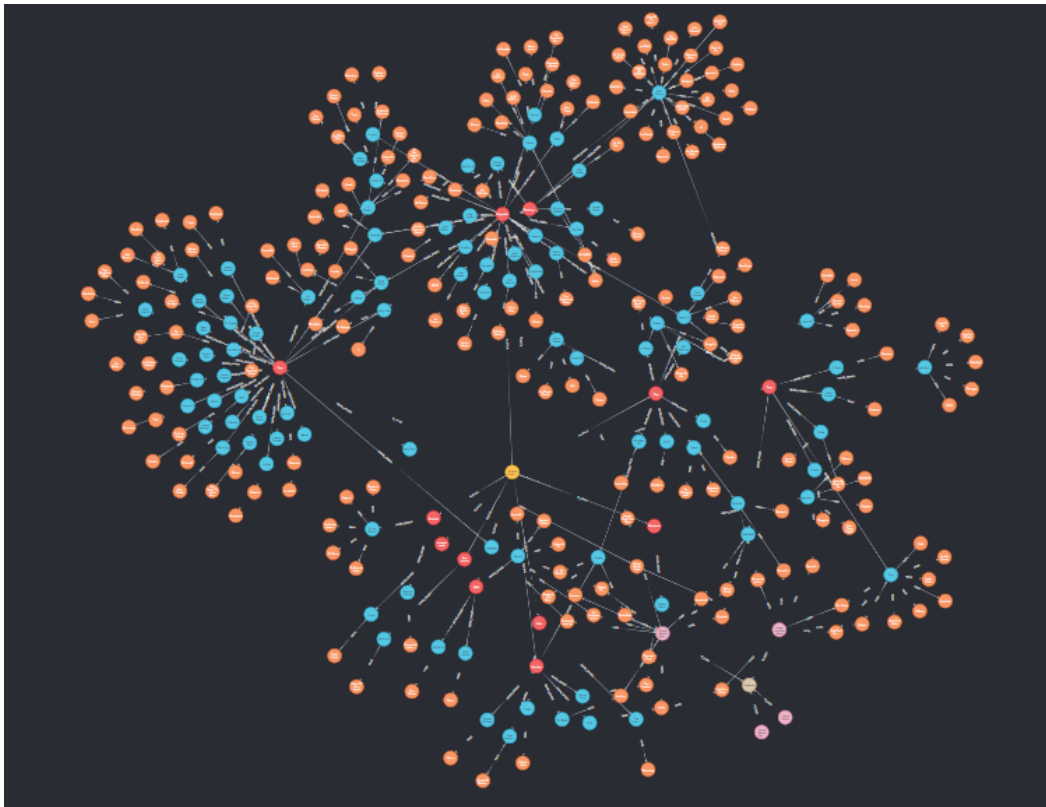
- Usuario: Cada uno de los usuarios que tiene la aplicación conforma un nodo diferente del cual se almacenan: Email, Name, Password, Usuario.
- Canciones: Cada pista musical en la base de datos conforma un nodo del cual se almacenan: Artist, Cover, Danceability, Energy, Genre, Name, Popularity, Preview, Valence.
- ArtistName: Cada artista se almacena como un nodo registrando su Name.
- Genre: Cada género se almacena como nodo registrando su Name.
- Style: Los estilos musicales conforman un único nodo al que van a dar a parar todos los géneros.
- Users: Los usuarios van a dar a único nodo.

Relaciones:

- Likes: Relaciona un usuario al que le gusta una pista musical.
- Hates: Relaciona un usuario que odia una pista musical.
- Is_User: Relaciona un nodo usuario con el nodo entero Users.
- Made: Relaciona un artista con una pista musical que haya compuesto.
- Genre_Singed: Relaciona un género con el artista al que pertenece.
- Is_Style: Relaciona todos los géneros al nodo de Estilos Musicales.

La base de datos hasta este punto ya estaría completa.

Imagen adjunta de la base de datos:

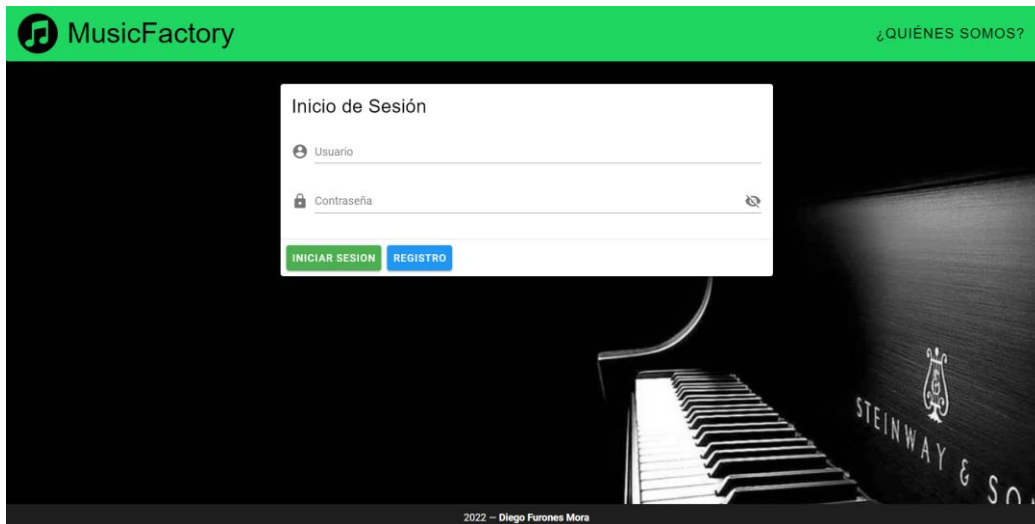


5.3. Sistema de Inicio de Sesión y usuarios

El próximo paso era realizar un sistema de registro de usuarios e inicio de sesión. Lo hice de una manera simple, solo quería que los usuarios sirvieran como almacén de canciones favoritas, no quería meterme en temas de perfiles llenos de datos innecesarios. Lo que hice fue crear una pagina de registro con un NavBar en la parte superior con el Logo y nombre de la aplicación, una pestaña a modo de introducción que llame ¿Quiénes Somos? y lo más importante, una Card con un formulario a rellenar con Nombre, email, Usuario y Contraseña. Seguí el mismo procedimiento para la página de Inicio de sesión, pero decidí que lo único necesario para iniciar sesión sería el nombre de usuario (campo que debería ser único para cada usuario registrado en la base de datos) y la contraseña. No implemente una opción de olvido de

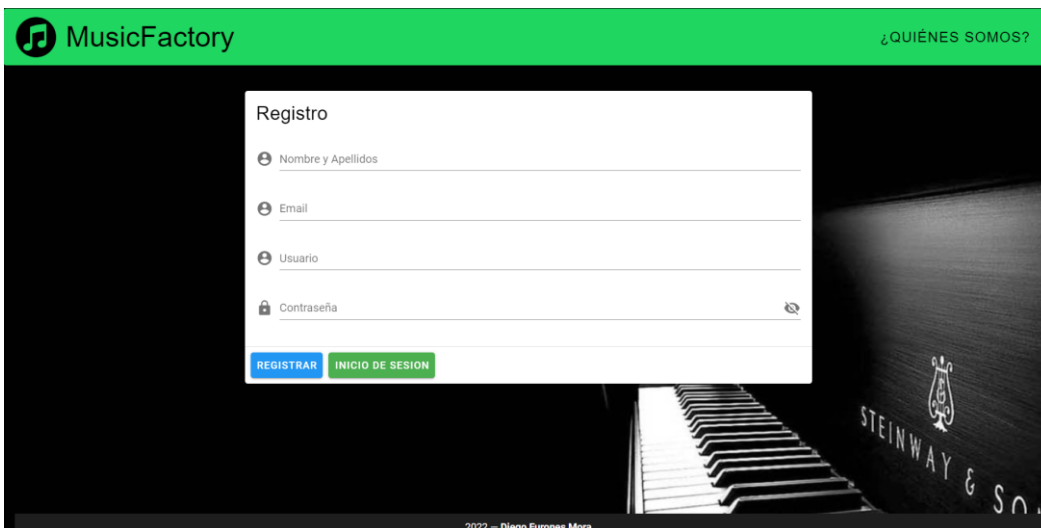
contraseña ni checkboxes aceptando los términos de uso como si hice en otras asignaturas, pero estas se centraban más en la funcionalidad pura de programar una aplicación web concreta y no en el algoritmo de recomendación como es necesario en esta asignatura. La página de ¿Quiénes Somos? se compone del mismo diseño que las otras dos mencionadas anteriormente, pero tiene un texto que funciona a modo de ayuda para entender la funcionalidad de la aplicación.

Pantalla de Inicio de Sesión:



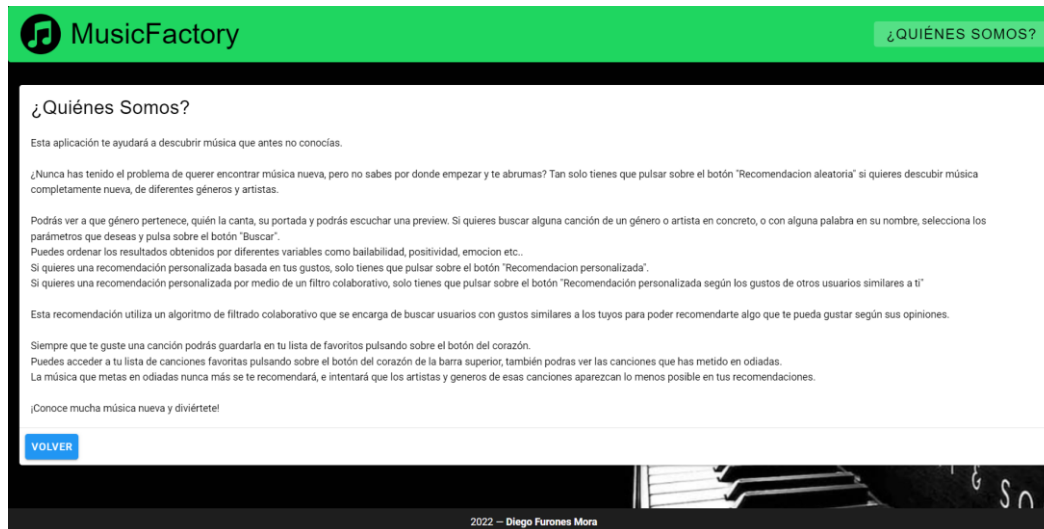
The screenshot shows the login page of the MusicFactory application. It features a green header with the MusicFactory logo and a link to '¿QUIÉNES SOMOS?'. The main content area has a dark background with a Steinway & Sons piano keyboard. A white login form is centered, titled 'Inicio de Sesión'. It contains two input fields: 'Usuario' and 'Contraseña' (password), with a toggle icon for password visibility. Below the fields are two buttons: 'INICIAR SESION' (green) and 'REGISTRO' (blue). At the bottom, there is a small copyright notice: '2022 - Diego Furonés Mora'.

Pantalla de Registro:



The screenshot shows the registration page of the MusicFactory application. It features a green header with the MusicFactory logo and a link to '¿QUIÉNES SOMOS?'. The main content area has a dark background with a Steinway & Sons piano keyboard. A white registration form is centered, titled 'Registro'. It contains four input fields: 'Nombre y Apellidos', 'Email', 'Usuario', and 'Contraseña' (password), with a toggle icon for password visibility. Below the fields are two buttons: 'REGISTRAR' (blue) and 'INICIO DE SESION' (green). At the bottom, there is a small copyright notice: '2022 - Diego Furonés Mora'.

Pantalla ¿Quiénes Somos?



5.4. Vista Home principal

Una vez la pasarela de autenticación de usuarios estaba implementada, era el turno de programar la vista principal, lo que cada usuario vería de su aplicación web una vez iniciara sesión.

La vista se compone de una barra (**NavBar**) de navegación con otra vez el logo y el nombre de la aplicación, además de un botón Recomendaciones que realizará un “scroll down” de la página hacia abajo, para mostrar los botones de recomendaciones de la aplicación. Además, para finalizar con la barra de navegación aparecen un símbolo de un corazón que te lleva a la página de favoritas, y un botón que al presionarlo cierra la sesión de la aplicación y te vuelve a llevar al inicio de sesión.

Debajo de la barra de navegación aparece un **Carrusel** de imágenes a modo de presentación de la aplicación de forma visual, con eslóganes y pocas palabras.

La página sigue de un apartado **Container** que muestra la portada de algunos de los álbumes más escuchados en 2022, además de un link que lleva a Spotify para escucharlos.

Seguido de esto por fin llega el **buscador** de canciones, tienes 3 opciones de Búsqueda y un filtrado de orden para los resultados. La primera opción es buscar las canciones por Género, la siguiente por Artista y por último tienes la opción de buscar la canción por su nombre. Los filtros de ordenado son los siguientes: Más o Menos animadas primero, Más o Menos bailables primero, Más enérgicas o tranquilas primero, Más

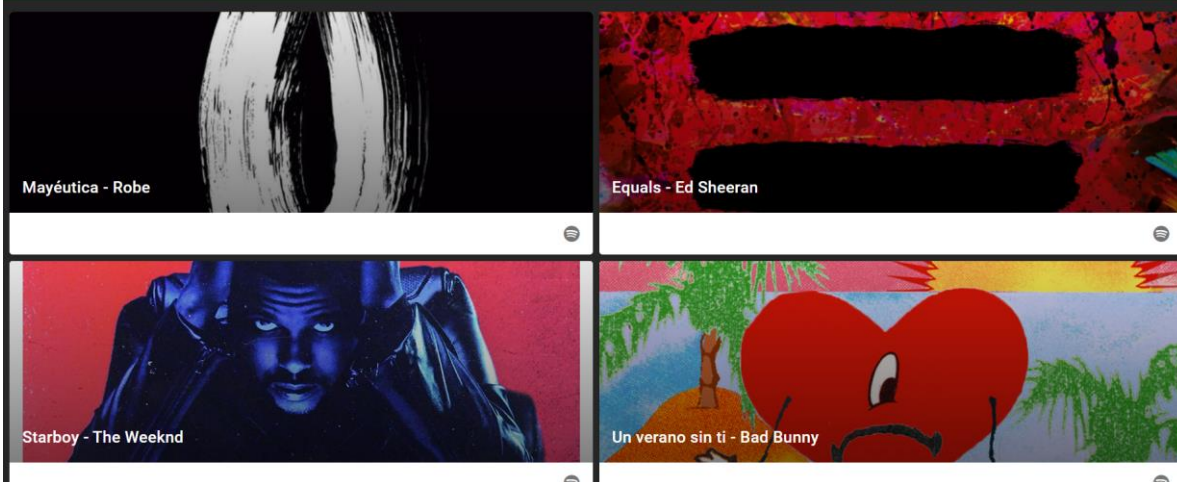
populares o menos primero. Debajo se encuentran los botones más importantes de la aplicación. El primero, “Recomendación Aleatoria”, hace que cuando sea presionado aparezca en la lista de canciones, una pista musical aleatoria de entre todo el dataset. El botón de su derecha, “Recomendación Personalizada”, hace que cuando lo presiones aparezca en la lista una canción que, según el algoritmo de recomendación que posteriormente explicaré, sea acorde a tus gustos, una canción perteneciente a tu género mas frecuente y de algún artista que aparezca más de una vez entre tu lista de favoritos. El último botón, “Recomendación personalizada según otros usuarios similares a ti”, hace que te aparezcan las 5 canciones que el algoritmo de filtro colaborativo decide que te pueden gustar en función de usuarios con mismo géneros o artistas frecuentes. También explicaré este algoritmo posteriormente.

Página principal, NavBar, Carrusel:



Álbumes más escuchados 2022:

Álbumes más escuchados 2022:

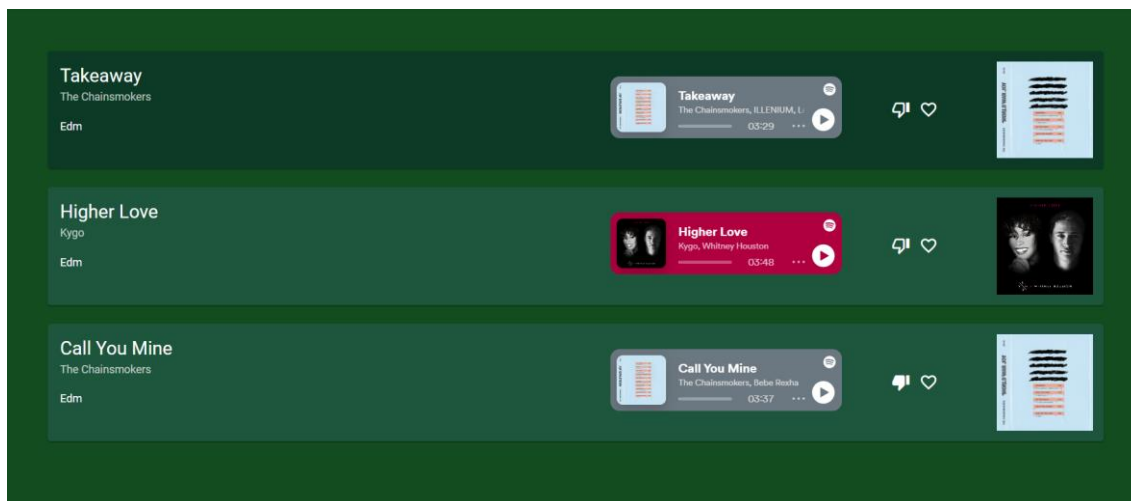


Recomendaciones:

Recomendaciones:

Género: Artista: Búsqueda por título: Ordenar por:

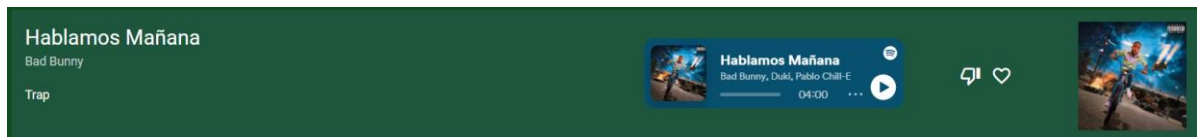
Lista de Canciones:



5.5. Cards de cada canción

Cada componente de la lista de canciones se forma de un tipo Card que presenta las siguientes partes:

- Título de la canción: Hablamos Mañana.
- Artista: Bad Bunny
- Género: Trap
- Preview de Spotify: Permite escuchar la canción presionando el Play e incluso acceder a la página de Spotify clickando en su icono.
- Botón no me gusta: Si lo presionas meterás la canción en tu lista de odiadas.
- Botón me gusta: Si lo presionas meterás la canción en tu lista de favoritas.
- Cover: Portada del álbum de la canción.







5.6. Vista Favoritos y Odiados

Al clicar en el corazón del NavBar de la vista principal de la aplicación nos llevará a esta página donde se muestra una pequeña información del usuario y su lista de canciones favoritas y odiadas.

Lo primero que encontramos es un banner con el fondo de una guitarra, en el frente encontramos una foto de perfil predefinida con un PNG de un avatar general, junto a él se encuentra el nombre completo asociado a la cuenta de usuario y su email. Debajo de esto se encuentra la funcionalidad de la propia pestaña: la lista de favoritos y odiados del usuario. Existe la opción de hacer click en un Switch para intercambiar la lista entre favoritos y odiados. En estas listas puedes seguir operando de la misma manera que en la vista principal, es decir, puedes desmarcar como me gusta una canción que desees eliminar de tu lista de favoritos, y, de la misma forma, eliminar una canción de tu lista de odiados.


Vista de Favoritos:


 Favoritas de dfurom00  

 **Diego Furones Mora**
diegosenoruf@gmail.com


Lista de Favoritas: ☒ Favoritas


Goteo
Duki
Trap

 **Goteo**
Duki
02:44







She Dont Give a Fo
Duki
Trap

 **She Don't Give a Fo**
Duki, KHEA
03:29




Vista de Odiados:


 Favoritas de dfurom00  

 **Diego Furones Mora**
diegosenoruf@gmail.com


Lista de Odiadas: ☐ Odiadas


Call You Mine
The Chainsmokers
Edm

 **Call You Mine**
The Chainsmokers
03:37



Never Really Over
Katy Perry
Pop

 **Never Really Over**
Katy Perry
03:43





6. La aplicación en algoritmos

Ahora pasaré a detallar como desarrollé el código necesario y las consultas Cypher a la base de datos para que la aplicación mostrara las canciones que se le piden.

6.1. Algoritmo de Búsqueda por Género, Artista y Nombre.

1.

```
app.get("/getSongs",(req,res)=>{
  const session = driver.session();
  var genre = req.query.genre;
  var busqueda = req.query.busqueda;
  var artist = req.query.artist;
  var tipo = [];
  tipo = req.query.type;
  console.log(req.query.type);
  var lista = [];
  var query = "MATCH (c: Canciones) ";
  //Género
  if(genre != "Cualquiera" && genre != ""){
    query += "where c.genre='" + genre + "' ";
  }
  //Artista
  if(artist != "Cualquiera" && artist != ""){
    if(query.includes("where")){
      query+="AND c.artist='"+artist+"' ";
    }else{
      query+="where c.artist='"+artist+"' ";
    }
  }
  //Nombre
  if (busqueda != "") {
    if(query.includes("where")){
      query += " AND (c.name =~ '(?i).*" + busqueda + ".*') ";
    }
    else{
      query += " where (c.name =~ '(?i).*" + busqueda + ".*') ";
    }
  }
  query += "return c";
  //Orden
```



2.

```
//Orden
if(tipo[0] != "No"){
  for(var i = 0; i < tipo.length; i++){
    if(tipo[i] == "+animada"){
      if(i == 0){
        query += " order by c.valence desc";
      }
      else{
        query += ", c.valence desc";
      }
    }
    else if(tipo[i] == "-animada"){
      if(i == 0){
        query += " order by c.valence";
      }
      else{
        query += ", c.valence";
      }
    }
    else if(tipo[i] == "+bailable"){
      if(i == 0){
        query += " order by c.danceability desc";
      }
      else{
        query += ", c.danceability desc";
      }
    }
    else if(tipo[i] == "-bailable"){
      if(i == 0){
        query += " order by c.danceability";
      }
      else{
        query += ", c.danceability";
      }
    }
  }
}
```

3.

```
else if(tipo[i] == "+energica"){
    if(i == 0){
        query += " order by c.energy desc";
    }
    else{
        query += ", c.energy desc";
    }
}
else if(tipo[i] == "-energica"){
    if(i == 0){
        query += " order by c.energy";
    }
    else{
        query += ", c.energy";
    }
}
else if(tipo[i] == "+popular"){
    console.log("++POPULARR");
    if(i == 0){
        query += " order by c.popularity desc";
    }
    else{
        query += ", c.popularity desc";
    }
}
else if(tipo[i] == "-popular"){
    if(i == 0){
        query += " order by c.popularity";
    }
    else{
        query += ", c.popularity";
    }
}
```

4.



```
const result = 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 se encuentran canciones.");
  }
});
```

1. Primero comprobamos qué es lo que nos está pidiendo el usuario que busquemos, por artista, género o por nombre, y vamos formando la query de Cypher en función a esto (se pueden encadenar por ejemplo artista y género).
2. Ahora comprobamos qué orden está pidiendo el usuario para mostrar las canciones.
3. Lo mismo que en el 2.
4. Tratamos el resultado de la query y devolvemos una lista con las canciones que hemos recibido de retorno.

6.2. Algoritmo de recomendación aleatoria

- 1.



```
randomSong() {  
  this.songs = [];  
  setTimeout(() =>{  
    var random = Math.floor(Math.random() * this.generos.length);  
    var g = this.generos[random];  
    axios.get(direccionIp + "/getRandomSongs",{  
      params:{  
        genre: g,  
        usuario: this.usuario,  
      },  
    }).then(response => {  
      if(response.data[0] == " ERROR"){  
        alert(response.data[0]);  
      }  
      else {  
        if(response.data[0] == undefined){  
          this.randomSong();  
        }  
        else {  
          this.songs.push(response.data[random]);  
          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");  
},
```

2.



```
app.get("/getRandomSongs",(req,res)=>{
  const session = driver.session();
  var genre = req.query.genre;
  var usuario = req.query.usuario;
  var query = "Match(c:Canciones), (a:Usuario) where c.genre='" + genre + "' and a.usuario='"
    + usuario + "' AND NOT (a)-[:LIKES]->(c) AND NOT (a)-[:HATES]->(c) ";
  var lista = [];
  query += "return c";
  const result = 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");
    }
  })
});
```

1. Esta pieza de código corresponde al frontend y lo que hace es que al activarse por medio del botón Recomendación Aleatoria calcula un número aleatorio para decidir el género de la canción y realiza una petición con ese género, cuando el backend le devuelve una lista con todas las canciones de ese género, este calcula un número aleatorio para decidir que canción elegir de entre todas como recomendación aleatoria.
2. En el backend recibes un género y realizas una query donde buscas todas las canciones de ese género las cuales el usuario no tenga ni en relación de Hate ni de Likes y devuelves una lista con todas.



6.3. Algoritmo de recomendación personalizada

```
personalizedSong(){
    var freqGenre = "";
    var freqArtist = "";
    var mostFrecGenre = 0;
    var mostFrecArtist = 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 (mostFrecGenre < countGenre) {
                    mostFrecGenre = countGenre;
                    freqGenre = this.favs[i].genre;
                }
                if (this.favs[i].artist == this.favs[j].artist) {
                    countArtist++;
                }
                if (mostFrecArtist < countArtist) {
                    mostFrecArtist = countArtist;
                    freqArtist = this.favs[i].artist;
                }
            }
        }
        this.rellenaArtistaG(freqGenre);
        var generoArtista = false;
        setTimeout(()=> {
```



```
setTimeout(=> {
  for(var w = 0; w < this.artistas.length; w++){
    if(this.artistas[w] == freqArtist){
      generoArtista = true;
      break;
    }
  }
  if(generoArtista == false){
    freqArtist = "";
  }
  if (freqGenre == "" && freqArtist == "") {
    alert("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;
        }
      }
    })
  }
}
```



```
}).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 + "/getGeneros", {
    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;
}
```



```
this.art = "";
}
this.genre = freqGenre;
axios.get(direccionIp + "/getPersonalizedSongs", {
  params: {
    genre: this.genre,
    artist: this.artist,
    usuario: this.usuario,
  },
}).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.búsqueda = "";  
    this.genre = "Cualquiera";  
    }, 100);  
    }, 200);  
    console.log("Salimos de la función PERSONALIZED SONG");  
    },
```

Esta función tiene como objetivo recomendar una canción personalizada para un usuario. La función comienza obteniendo las canciones favoritas y las que no le gustan al usuario (mediante las funciones `getFavs()` y `getHateds()`). Luego, se calculan dos estadísticas para las canciones favoritas del usuario:

1. El género de música más frecuente entre las canciones favoritas del usuario.
2. El artista más frecuente entre las canciones favoritas del usuario.

Para calcular estos valores, se itera sobre cada una de las canciones favoritas y se comparan con las demás. Si se encuentra que un género o artista es igual a uno ya contabilizado, se aumenta un contador. Al final del proceso, se guarda el género y artista más frecuentes en las variables `freqGenre` y `freqArtist` respectivamente.

Si ninguno de estos valores es encontrado (es decir, si `freqGenre` y `freqArtist` son ambos iguales a ""), entonces se muestra un mensaje de alerta y se procede a recomendar una canción aleatoria. Si sólo falta uno de estos valores, entonces se hace una llamada al backend para obtener un género o artista posible y se elige uno al azar de entre estas opciones que devuelve.

Una vez que se ha determinado un género y artista para la recomendación personalizada, se procede a buscar canciones que coincidan con estos criterios mediante la función `rellenaCanciones()`.



```
app.get("/getPersonalizedSongs",(req,res)=>{
  console.log("Estoy en funcion /getPersonalizedSongs");
  const session = driver.session();
  var genre = req.query.genre;
  var usuario = req.query.usuario;
  var artist = req.query.artist;
  var query = "Match(c:Canciones), (a:Usuario) where c.genre='" + genre + "' and a.usuario='"
+ usuario + "' 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");
});
```

La función `app.get("/getPersonalizedSongs",(req,res)=>{...})` es una ruta HTTP GET que se ejecuta cuando el cliente envía una solicitud GET a la ruta especificada. Esta ruta se utiliza para obtener una lista de canciones personalizadas para un usuario específico.

La ruta acepta tres parámetros en la solicitud:

1. `genre`: el género de música que se quiere obtener.
2. `usuario`: el nombre de usuario del usuario al que se le están recomendando las canciones.
3. `artist`: el artista del que se quieren obtener las canciones.

La consulta busca canciones que tengan el género especificado y que no hayan sido marcadas como favoritas ni como odiadas por el usuario especificado. Los resultados de la consulta se guardan en una lista y se envían al cliente en la respuesta HTTP.

La función también incluye manejo de errores para casos en los que no se encuentren canciones que cumplan los criterios de búsqueda.

6.4. Algoritmo de recomendación de filtro colaborativo

La función tiene como objetivo recomendar canciones a un usuario principal en base a sus gustos musicales y los de otros usuarios que tienen gustos similares.

La función comienza imprimiendo un mensaje en la consola y guardando el número de canciones favoritas del usuario principal en una variable llamada "oldFavs". Luego, llama a otra función llamada "getFavs", que parece obtener las canciones favoritas del usuario principal. Después de un retraso, la función imprime las canciones favoritas del usuario principal en la consola.

Si el usuario principal tiene 5 o más canciones favoritas, la función realiza una serie de pasos para recomendar canciones al usuario. Primero, busca otros usuarios que tienen gustos similares al usuario principal (es decir, que tienen algunas de las mismas canciones favoritas). Luego, se queda con aquellos usuarios que tienen el mayor número de canciones en común con el usuario principal. Después, busca entre esos usuarios aquellos que tienen un número similar de canciones favoritas (que no sean canciones que el usuario principal ya haya marcado como favoritas o como no favoritas). Finalmente, busca entre esos usuarios las canciones más coincidentes y las muestra al usuario principal, ordenadas de más a menos coincidentes.

Si el usuario principal tiene menos de 5 canciones favoritas, la función muestra un mensaje indicando que no hay suficientes canciones para recomendar al usuario.

La función también incluye una serie de alertas para informar al usuario de lo que está sucediendo a medida que se realizan estos pasos. Además, incluye algunas variables y funciones adicionales que no se mencionan en el código proporcionado, como "songs", "songsDepuradas", "usuariosVecinos", "vecinosDepurados" y "reordenarRecomendación". Es posible que estas variables y funciones se utilicen en alguno de los pasos mencionados anteriormente para llevar a cabo la recomendación de canciones.



```

/***** COLABORATIVE FILTER *****/
colaborativeFilter: function(){
  this.songs = [];
  console.log("Estamos en la función COLABORATIVE FILTER Máximo de 5 canciones recomendadas");
  // 1. Guardamos las canciones favoritas del usuario Principal
  var oldFavs = this.favs.length;
  this.rellenar();
  this.getFavs();
  setTimeout(() =>{
    console.log(this.favs);
    if(this.favs.length >= 5){
      if(this.favs.length == oldFavs){
        this.num = 0;
        this.songs = [];
        console.log("Mora")
        console.log(this.songsDepuradas);
        if(this.songsDepuradas.length > 5){
          this.visible = true;
          alert("Mostrando las 5 canciones más recomendadas ordenadas de más a menos recomendadas")
        }
        else{
          if(this.songsDepuradas.length >= 1){
            alert("Mostrando las canciones más recomendadas ordenadas de más a menos recomendadas")
          }
          else{
            alert("No hay canciones recomendadas para ti. jincho");
          }
        }
      }
      //Recomendamos las 5 canciones con más coincidencias
      for(var i = 0; i < this.vecinosSongs.length; i++){
        this.songs.push(this.vecinosSongs[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;
      }
    }
  }, 1000);
}

```



```
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 < 1){
                alert("No hay suficientes usuarios con gustos similares a los tuyos como para hacer una recomendación personalizada.");
            }
            else{
                //4. Buscamos los que tengan un num de canciones favoritas (distintas de las que ama y odia el usuario)
                this.depurarUsuariosPorCanciones();
                setTimeout(() => {
                    //5. Buscamos las canciones mas coincidentes entre los usuarios
                    this.depurarCancionesRecomendadas();
                    setTimeout(() => {
                        this.songs = [];
                        //Reordenamos las canciones
                        this.reordenarRecomendacion();
                        //Ponemos color a las canciones
                        this.num = 0;

                        //Recomendamos las 5 canciones con más coincidencias
                        console.log("final");
                        alert("Mostrando las 5 canciones más recomendadas ordenadas de más a menos recomendadas. Pulsa aquí para verlas");
                        console.log(this.vecinosSongs);
                    });
                });
            }
        });
    });
}
```

```
for(var i = 0; i < this.vecinosSongs.length; i++){
    this.songs.push(this.vecinosSongs[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){
        break;
    }
}
}, 200);
}, 200);
}
}, 300);
}, 300);
}
}
else{
    alert("No tiene suficientes canciones favoritas como para realizar una recomendación personalizada.");
}
}, 500);
console.log("Salimos de la función COLABORATIVE FILTER");
},
```

7. Simulaciones

Las simulaciones de la aplicación se realizarán en el vídeo de la entrega.

8. Análisis DAFO

El análisis consiste en la identificación y evaluación de las Fortalezas (Strengths), Debilidades (Weaknesses), Oportunidades (Opportunities) y Amenazas (Threats) de la organización o proyecto.

El objetivo del análisis DAFO es proporcionar una visión global de la situación actual de la organización y ayudar a identificar las estrategias más adecuadas para aprovechar las oportunidades y fortalezas, y minimizar las debilidades y amenazas.

Fortalezas:

- Algoritmos de recomendación avanzados que proporcionan sugerencias personalizadas y relevantes para cada usuario.
- Interfaz de usuario amigable y fácil de usar.
- Integración con Spotify para facilitar la reproducción.

Debilidades:

- Dependencia de la calidad y disponibilidad de los datos de usuario para proporcionar recomendaciones precisas.
- La competencia con otras aplicaciones de recomendación de música puede limitar la adquisición y retención de usuarios.
- Puede ser costoso mantener y actualizar constantemente la base de datos y los algoritmos de recomendación.

Oportunidades:

- Una amplia base de usuarios puede proporcionar una fuente importante de datos para mejorar la precisión de las recomendaciones.
- La aplicación puede ser utilizada como una herramienta de marketing para promocionar a artistas emergentes y ayudarles a conectarse con nuevos fans.
- Puede ser una fuente de ingresos a través de publicidad y suscripciones.

Amenazas:

- La competencia con otras aplicaciones de recomendación de música puede limitar el crecimiento y la rentabilidad.

- La dependencia de terceros (por ejemplo, la API de Spotify) para proporcionar contenido puede aumentar el riesgo de interrupciones o problemas de acceso.
- Cambios en las preferencias de los usuarios o en las tendencias de la industria de la música pueden afectar negativamente el rendimiento de la aplicación.

9. Líneas de Futuro

Se me ocurren muchas posibilidades con el rumbo que puede tomar esta aplicación web de recomendación de música.

Ahora mismo a finales de año como estamos, están apareciendo muchísimas plataformas que realizan una especie de recopilación de la música que más escucha cada usuario y se está poniendo muy de moda. A los usuarios les encanta que las aplicaciones definan su gusto musical para compartirlo en otras redes como Instagram, y darse cuenta de que compartían mas gustos de los que creían con otras personas. Es por eso que implementar un servicio de estadísticas en la propia aplicación sería increíble y una apuesta por la que tiraría porque veo una gran oportunidad a explotar.

Lo implementaría de forma de que cada usuario pudiera acceder en cualquier momento a una especie de top con los 5 artistas más escuchados en su perfil, además de un gráfico circular con los géneros más frecuentes para él y las canciones más escuchadas. Además, podría implementar lo mismo, pero a nivel mundial y no personalizado, es decir, los artistas y géneros más escuchadas por todos los usuarios de la web, para ver cuanto distan de estos. Todo esto necesita una especie de link para compartir directamente una captura a modo de resumen para poder subirlo a Instagram.

10. Lecciones Aprendidas

A base de la repetición y los errores se aprende mucho más que si terminas sacando todo a la primera, para mejorar hay que llegar a un estado en el que parece que estas dando palos de ciego hasta que empiezas a vislumbrar un camino y tirar por ahí. La verdad que en cuanto a la idea del proyecto siempre tuve claro que quería hacer un sistema de recomendación de música. Al principio no sabía ni por donde empezar, ni como utilizar la API de Spotify, y mucho más importante, nunca había manejado una base de datos basada en grafos como es Neo4J. Las primeras semanas parecía que no avanzaba porque realmente no programaba nada, pero las dediqué a aprender como hacer las cosas para luego ponerlas en práctica. Ahora, una vez finalizada la práctica, me doy cuenta de que sé manejar el framework de desarrollo de frontend, Vue, que es otra cosa que antes no había manejado y decidí aprenderlo, además de Neo4J, he aprendido algoritmos de programación para recomendar, aprovechar la API de Spotify y hasta en mejorado en la forma de pensar el cómo abarcar este tipo de trabajos.

11. Conclusión

Creo que he realizado un gran trabajo del que estoy satisfecho y espero aprobar. He aprendido bastante en su realización y tengo pensado realizar el TFG a partir de este trabajo. De cara al futuro y a la realización de este tengo pensado lo primero mejorar la base de datos, ya que a la hora de realizar el trabajo estaba un poco limitado por el volumen de esta, y al final cuantas mas canciones, mas acertado es el algoritmo de recomendación. Otro aspecto a mejorar es el diseño del frontend para que sea mas agradable visualmente y más fácil de utilizar para el usuario.

Me he dado cuenta de que el tema de este trabajo es de uno de los aspectos que más me han gustado en la carrera, puesto que me encanta la música y programar frontends y esta práctica lo fusiona.

12. Bibliografía y referencias

Web api. (n.d.). Retrieved December 20, 2022, from
<https://developer.spotify.com/documentation/web-api/>