UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

**TFG del Grado en Ingeniería Informática**

# SpotMyFM

Sistema de Gestión, Análisis y Recomendación de Biblioteca Musical de Spotify apoyado por Inteligencia Artificial

spotmyfm.jorgeruizdev.com

Presentado por Jorge Ruiz Gómez
en Universidad de Burgos — 7 de julio de 2022
Tutor: Bruno Baruque Zanón

UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

D. Bruno Baruque Zanón, profesor del departamento de Ingeniería Informática, área de Ciencia de la Computación e Ingeligencia Artificial.

Expone:

Que el alumno D. Jorge Ruiz Gómez, con DNI —, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado SpotMyFM.

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección del que suscribe, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 7 de julio de 2022

Vº. Bº. del Tutor:

D. Bruno Baruque Zanón

## Resumen

Spotify es una de las plataformas de música en streaming más importantes del mundo. Se propone aumentar las funcionalidades de Spotify a partir de su API pública y otros servicios.

Se plantea el desarrollo y despliegue de una plataforma web que permita a los usuarios de Spotify explorar y gestionar su biblioteca de una manera más avanzada y transparente a la que ofrece Spotify por defecto.

La plataforma estará acompañada por una base de datos y un servicio de recuperación de información musical[1], que extraerá características de la señal de sonido de la canción y permitirá asociar géneros, subgéneros y estados de ánimo musicales a las distintas pistas; se utilizarán tres fuentes de datos, la API pública de Spotify, la API pública de LastFM y la API de SpotMyFM.

## Descriptores

Recuperación de información musical, CNN, Spotify, plataforma web, sistema de recomendación musical[2].

[1] Campo de investigación dedicado al análisis, modificación y creación de música a partir de diversas técnicas.

[2] Sistema que permite predecir la preferencia de usuario ante una serie de canciones.

## Abstract

Globally, Spotify is one of the most important music streaming platforms. This project aims to improve Spotify's functionalities based on its public API and other services.

This project explores the development and deployment of a web platform that allows Spotify users to explore and manage their library in a more advanced and transparent way than the one offered by Spotify.

The platform will be supported by a database and a music information retrieval service (MIR), which will extract features from the sound signal of the song and associate genres, subgenres, and musical moods to the different tracks; three data sources will be used, Spotify's public API, LastFM's public API, and SpotMyFM API.



## Keywords

Music information retrieval, CNN, Spotify, web platform, recommender system.

# Índice general

# Índice de figuras

# Índice de tablas

# Introducción

SpotMyFM es una plataforma diseñada para gestionar la biblioteca y playlists personales de un usuario de Spotify.

Este proyecto busca estudiar el diseño de Spotify y sus distintos sistemas, como sistemas de recomendación o generación de playlists automáticas. Para llevar a cabo este objetivo se ha decidido implementar en un servicio a parte una expansión de la funcionalidad de Spotify que complemente a los servicios ya existentes, como consultar estadísticas, canciones favoritas o generar playlists específicas a partir de filtros y/o recomendaciones.

La plataforma web incluye 7 servicios diferenciados:

1. **Gestión de Favoritos:** Es la página principal, muestra las canciones y artistas más escuchados en varios intervalos.
2. **Gestión de Biblioteca:** Permite explorar todas las características de la biblioteca personal del usuario[3]
3. **Gestión de Álbumes:** Permite gestionar y etiquetar los álbumes favoritos del usuario. El sistema de etiquetar álbumes se asemeja a la creación de listas de álbumes.
4. **Gestión de Playlists:** Permite explorar las playlists creadas o marcadas como me gusta por el usuario. Se pueden detallar cada una de las playlists.
5. **Análisis de Canción:** Permite subir y analizar una canción cualquiera. El analizador devuelve los géneros, subgéneros y estados de ánimo de la canción, así como 5 canciones similares.

[3]Las canciones marcadas como "Me Gusta

6. **Búsqueda:** Permite buscar a cualquier artista, canción, álbum o playlist y disfrutar de las características que ofrece la plataforma.

7. **Reproductor:** Permite reproducir álbumes o canciones específicas, así como añadir a la cola, saltar o retroceder una canción.

Para implementar todas estas características se ha usado la API pública de LastFM, y se ha desarrollado una arquitectura de contenedores escalables interconectados 1.1.

Ludwig es un servicio que permite, a partir del un fragmento de una canción, obtener canciones similares en Spotify, géneros, subgéneros y los estado de ánimo de una canción. Este servicio utiliza redes neuronales convolucionales (CNN) para clasificar las canciones a partir de su representación mediante sus coeficientes cepstrales en la frecuencia de mel.

NextJS actúa como intermediario entre el cliente, Ludwig y la base de datos. Se encarga de la generación de las páginas web, la autenticación de usuario, comunicación con Ludwig y la gestión de las transacciones de la base de datos.

Se ha desplegado el proyecto en spotmyfm.jorgeruizdev.com con una pequeña demo interactiva sin necesidad de registro con algunas de las funcionalidades.

## Estructura de la Memoria

1. **Introducción**: Descripción del proyecto y estructura de la documentación.

2. **Objetivos del Proyecto**: Objetivos generales, personales y técnicos que se esperan cumplir con este proyecto.

3. **Conceptos Teóricos**: Explicación detallada sobre los conceptos más importantes del proyecto.

4. **Técnicas y Herramientas**: Listado y descripción de bibliotecas, servicios y técnicas utilizados durante el proyecto, así como una breve justificación y explicación de su uso.

5. **Aspectos Relevantes**: Listado de los puntos más interesantes e importantes que han surgido durante el desarrollo del proyecto.

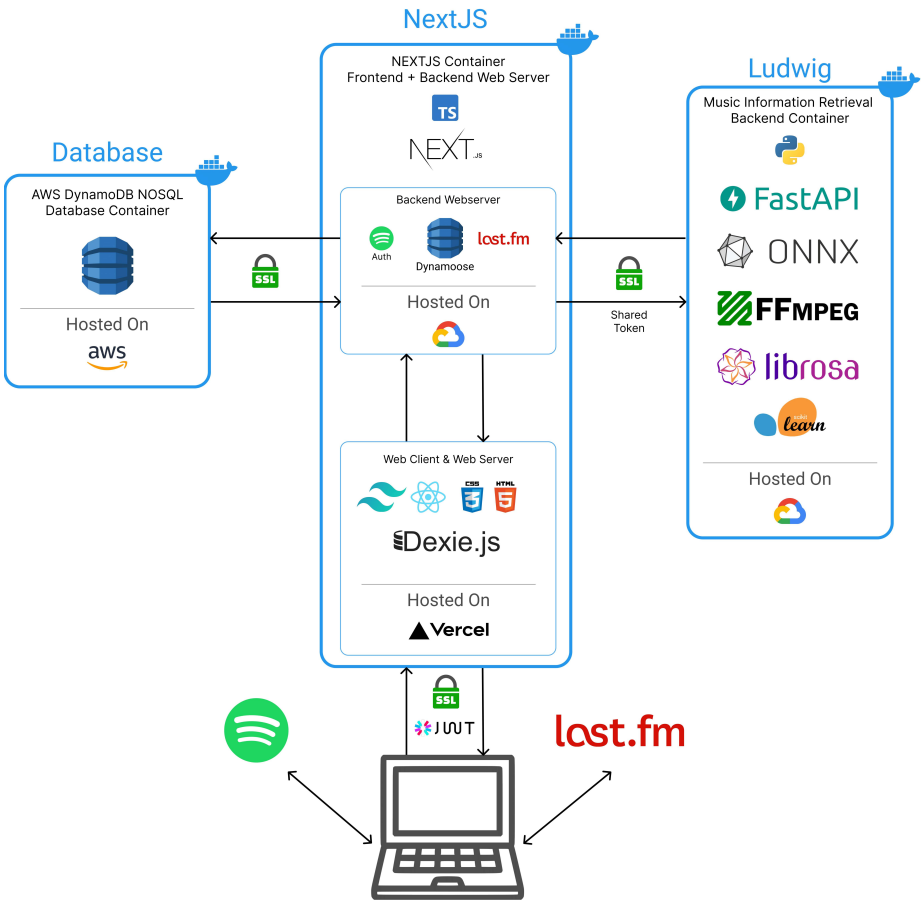



Figura 1.1: Arquitectura del Proyecto

6. **Trabajos Relacionados**: Exposición de trabajos y plataformas relacionadas con SpotMyFM.

7. **Conclusiones y Lineas de Trabajo Futuras**: Conclusiones sobre los distintos aparatados del proyecto y un planteamiento de las mejoras que puede recibir el proyecto a lo largo del tiempo.

## Estructura de los Anexos

A. **Plan del proyecto**: Plan de proyecto software que estudia la planificación y viabilidad del proyecto.

B. **Especificación de Requisitos**: Catálogo de los requisitos funcionales y no funcionales del proyecto.

C. **Especificación de Diseño**: Diseño general de los distintos apartado del sistema así como su justificación.

D. **Manual de Programador**: Manual con toda la información importante que permita a otro programado retomar el proyecto rápidamente.

E. **Manual de Usuario**: Manual de usuario que explora todas las características de la plataforma web.

## Otros Materiales

Todos los materiales están disponibles en los USBs con las fuentes.

1. **Datasets**: Contiene los distintos conjuntos de datos utilizados para los sistemas de clasificación y recomendación.

2. **NextJS**: Contiene la aplicación de Frontend y Backend Principal.

3. **Ludwig**: Contiene las herramientas, notebooks y backend utilizados para el servicio de recuperación de la información musical y recomendación.

4. **Docker**: Contiene un fichero **tcompose.yml** que permite levantar todos los servicios automáticamente.

5. **Manual**: Contiene un fichero **readme.md** principal con el manual de la plataforma. Este fichero enlaza con archivos locales markdown más concretos que detallan una funcionalidad o vista.

6. **Docs**: Contiene los diagramas, ficheros fuente LaTeXde la memoria y la guía de estilo.

# Objetivos del proyecto

Este apartado explica de forma precisa y concisa cuales son los objetivos que se persiguen con la realización del proyecto. Se puede distinguir entre los objetivos marcados por los requisitos del software a construir y los objetivos de carácter técnico que plantea a la hora de llevar a la práctica el proyecto.

## 2.1. Objetivos Generales

1. Diseñar e implementar un servicio web que extienda la capacidades de un servicio ya existente.

2. Diseñar e implementar una arquitectura que interconecte múltiples servicios

3. Diseñar e implementar un servicio que permita analizar y recomendar canciones a partir de un fichero de audio musical.

## 2.2. Objetivos Técnicos

1. Desarrollar una aplicación web totalmente responsive[4].

2. Desarrollar una aplicación web con modo claro y oscuro.

3. Poner en práctica los conocimientos de Integración Continua / Despliegue Continuo (CI/CD)

[4]Los contenidos se adaptan al dispositivo.

4. Crear un sistema de recomendación de música colaborativo y basado en contenidos integrado con Spotify.

5. Conocer y emplear frameworks y bibliotecas de pruebas, unitarias, automáticas y End to End (E2E).

6. Conocer los distintos conceptos y practicar el uso de técnicas para trabajar con audio digital.

## 2.3. Objetivos Personales

1. Estudiar el diseño de una API comercial e integrarla en un proyecto.

2. Conocer el ciclo de desarrollo completo[5] de un proyecto web.

3. Diseñar, entrenar e integrar un modelo de deeplearning con una aplicación web.

4. Ampliar conocimientos de HTML5 y CSS3 a partir del diseño e implementación de componentes web.

[5] Análisis, especificación, diseño, desarrollo, pruebas, SEO y despliegue.

# Conceptos Teóricos

Una de las funcionalidades planteadas para la plataforma es etiquetar automáticamente los géneros, subgéneros y estados de ánimo de canciones a partir de su onda sonora. Para ello se han utilizado técnicas de aprendizaje aprendizaje profundo, ya que según el artículo [6] presentan un mejor resultado que las técnicas de aprendizaje automático tradicionales.[6]

## 3.1. Conceptos Sonido

### Onda Digital y Frecuencia de Muestreo

Una onda analógica es una señal continua a lo largo del tiempo. Para poder trabajar con una onda digitalmente, es necesario discretizar la señal analógica en una señal digital. Para ello, se declara una frecuencia de muestreo que indica cada cuantas medidas discretas se realizan sobre la amplitud de la onda en una señal analógica a lo largo del tiempo. En nuestro caso vamos a trabajar con una frecuencia de muestreo de $22050Hz$, es decir, se van a tomar 22050 puntos discretos de la amplitud de una onda cada segundo.
Este proceso de discretización de una señal digital es conocido como **modulación por impulsos codificados**. El formato de audio .wav almacena el audio sin comprimir directamente con esta modulación, por eso será utilizado a lo largo del proyecto en vez de otros formatos con mejor compresión como .mp3.

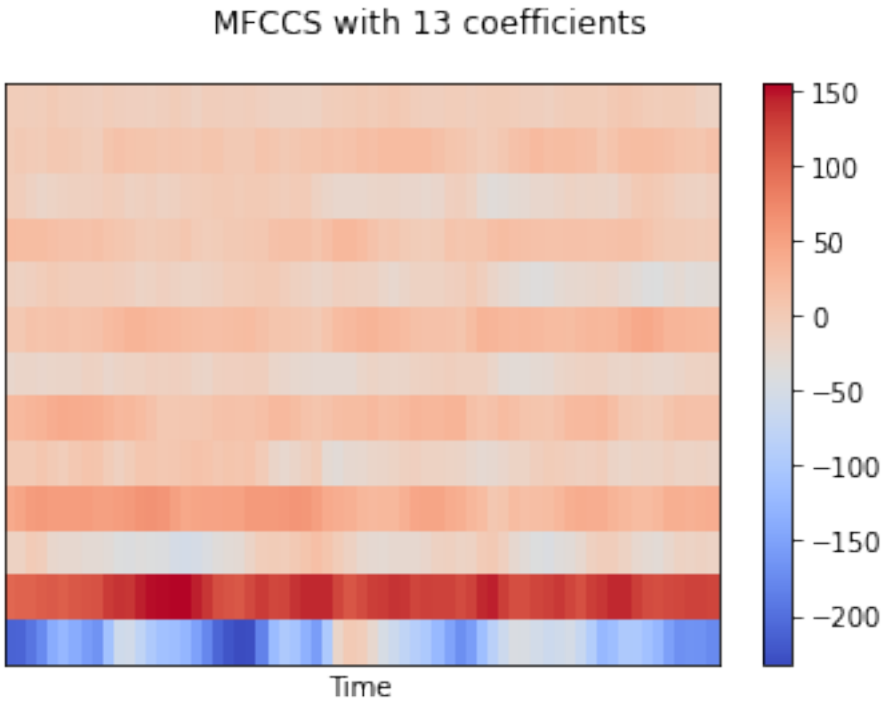



Figura 3.1: MFCC de un fragmento de 30s de una canción. Extraído de spotmyfm/Ludwig/notebooks/gtzan/mfcc.ipynb

## Coeficientes Cepstrales en las Frecuencias de Mel

Los coeficientes cepstrales en las frecuencias de mel o MFCC Fig.3.1 son un tipo de representación de la densidad espectral del sonido que intentan representar la percepción auditiva humana.

Son muy utilizados en tareas de reconocimiento del habla y recuperación de información musical.

Estos coeficientes están ajustados sobre la **escala de mel**, una escala que aproxima la escala de tono a una escala similar a la percepción humana, mejorando la representación del sonido.

Para obtener los MFCC partimos del espectrograma de mel, un espectrograma ajustado a la escala de mel. En este caso, estamos intentando obtener el espectrograma de un espectrograma, también conocido como cepstrum.

Un método para calcular los MFCCs puede[7] ser el siguiente:

[6]KNN, SVM, Random Forest, MLP y Regresión Logística.

[7]Cada biblioteca tiene una aproximación diferente.

1. Separar la señal en segmentos (hop_length)
2. Aplicar la Transformada Discreta de Fourier a cada segmento para obtener la **potencia espectral**.
3. Transformar los resultados del paso 2 a la escala de mel.
4. Aplicar el logaritmo a cada frecuencia de mel, (escala logarítmica en **decibelios**).
5. Aplicar la Transformada Discreta del Coseno al resultado del paso 4
6. Escoger $N$ MFFCs[8].

Los pasos 1-3 se corresponden con la obtención de un espectrograma de mel.

Se recomienda consultar el apéndice A de [7] para consultar los detalles de la extracción.

## 3.2. Clasificación mediante Redes Neuronales Convolucionales

Las redes neuronales convolucionales son una de las arquitecturas neuronales más utilizadas en visión artificial debido a su poder para detectar patrones en imágenes, en nuestro caso vamos a buscar patrones en una representación bidimensional del sonido, los MFCCs 3.1. Estas redes suelen estar formadas por una serie de bloques convolucionales, seguidos de una o varias capas densas en la salida de la red, como muestra 3.2. La zona aizquierda de la línea discontinua roja compone la parte convolucional de la red, mientras que a la derecha de la línea se encuentra la parte densa de la red. Las última capa densa, compuesta por 3 neuronas, hace referencia a la capa de salida.

### Bloque Convolucional

Estos bloques, formados por capas, buscan resaltar los elementos más relevantes en los datos a partir de las operaciones de convolución y pooling.

La operación de convolución está formada por un kernel o filtro Fig.3.2 que se aplica Fig.3.4 a través de los datos. En el caso de la capa de convolución, los pesos entrenables de dicha capa son el propio kernel.

[8]En 3.1 se han escogido $N = 13$ MFCCs

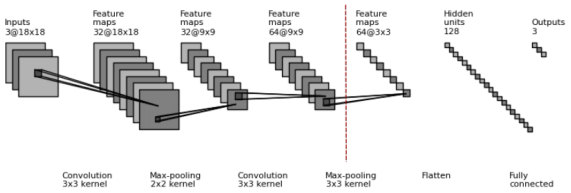



Figura 3.2: Arquitectura de una red convolucional. Extraído de [1]

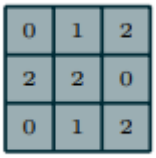



Figura 3.3: Kernel 3x3. Extraído de [2]

Esta operación puede estar acompañada por una función de activación para cada una de las neuronas, como por el ejemplo la función de activación RELU (3.1).

$$Relu(x) = max(0, x) \tag{3.1}$$

Por otro lado, la operación de pooling intenta reducir el tamaño de la entrada, dejando solo los datos relevantes de un área. Se aplica de la misma manera que una convolución, pero en vez de aplicar un kernel, realizamos una operación aritmética sobre el área, como obtener el valor más grande del área (Max Pooling) o la media de todos los valores (Average Pooling)

Un bloque convolucional puede contener otras capas, como capas de normalización, que normalizan de nuevo los datos para trabajar siempre en el intervalo [0,1] o [-1, 1].

## Densa

La salida de una red neuronal puede estar formada por capas densas, estas capas tienen una entrada unidimensional, por lo que si la salida del último bloque convolucional no es unidimensional, es necesario utilizar una operación de Flatten que transforma la matriz de entrada en una única fila concatenando cada una de las filas de la matriz.

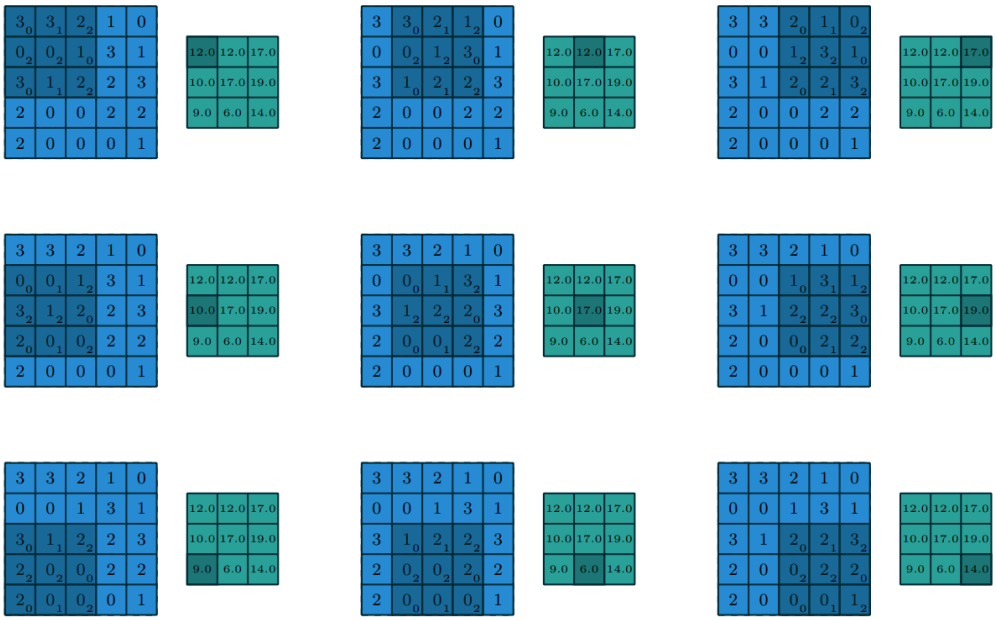


Figura 3.4: Aplicación del kernel 3x3 sobre un área 5x5. Extraído de [2]

En este tipo de arquitectura, cada una de las neuronas de una capa se interconectan con todas las neuronas de la capa siguiente.

Cada neurona está formada por:

- Entradas ($[x_1 \dots x_n]$)
- Salida ($y$)
- Vector de pesos entrenables ($[w_1 \dots w_n]$)
- Función de activación ($fn(x)$)
- Bias de activación entrenable ($[b_1 \dots b_n]$)

Y la salida (o salidas) de una neurona se calculan de la siguiente manera (3.2).

$$y = fn(x \cdot w + b) \tag{3.2}$$

La salida de cada una de estas neuronas se propaga a la siguiente capa como la entrada de cada neurona.

### Capa de Salida

En tareas de clasificación, la capa de salida está formada por una o varias neuronas [9], y una función de activación. Si debemos clasificar **varias clases**, utilizamos la función de activación Softmax, que aplica una normalización en la salida para resaltar la salida con la clase resultante.

Conociendo la salida de cada una de las neuronas, obtenemos la neurona con la salida más cercana a 1 (la salida más grande)

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad for\ i = 1, 2, \ldots, n \tag{3.3}$$

Para obtener la clase, calculamos cual es la neurona con la activación más elevada.

$$clase = argmax([y_0 \ldots y_n]) \tag{3.4}$$

En tareas de clasificación con **varias etiquetas** (multietiqueta), utilizamos la función de activación Sigmoide, que permite a cada neurona tener una salida entre 0 y 1, siendo este la confianza de que la clase contenga dicha etiqueta.

$$sigmoid(x) = \frac{1}{1 + e^{-}x} \tag{3.5}$$

Consideramos que una entrada contiene una etiqueta si el resultado de la sigmoide es mayor o igual que 0.5.

## 3.3. Cuantización de Redes Neuronales

La cuantización de redes neuronales es una de las técnicas de optimización de redes neuronales entrenadas más utilizada.
El proceso de cuantización consiste en convertir los distintos pesos (y por tanto operaciones) de la red neuronal a un tipo de datos con menos bits, lo que permite ahorrar memoria y acelerar las operaciones. Normalmente, la cuantización convierte los tensores de FLOAT32 (4 bytes por peso) a INT8 (1 byte por peso).

En el proceso de cuantización tenemos presente dos parámetros para cada uno de los tensores:

[9] Depende del número de clases a clasificar

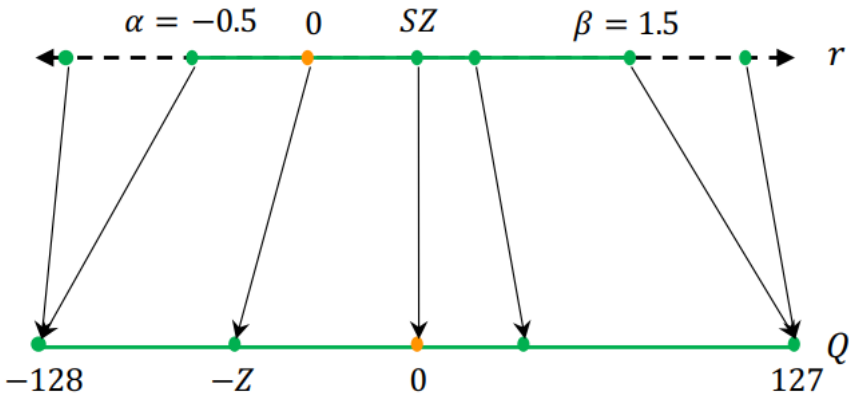



Figura 3.5: Cuantización INT8 en el intervalo $[-0{,}5, 1{,}5]$. Extraído de [3]

**Escala** ($S$): Es un número decimal que nos permite escalar los valores entre un número real y uno cuantizado.

**Zero Point** ($Z$)**:** Es un número dentro del intervalo de cuantización. Nos indica el valor del 0 real en valores cuantizados.

Estos valores son calculados a partir de analizar las distribución de los distintos tensores y activaciones.

$$S = \frac{\beta - \alpha}{\beta_q - \alpha_q} \tag{3.6}$$

$$Z = \left\lfloor \frac{\beta\alpha_q - \alpha\beta_q}{\beta - \alpha} \right\rfloor \tag{3.7}$$

En el caso de la distribución de valores reales, debemos conocer el valor mínimo ($\alpha$) y el máximo ($\beta$) del tensor/distribución a cuantizar.[10]

Para los tensores cuantizados, $\alpha_q$ y $\beta_q$ se corresponden con el mínimo y el máximo de los tipos de datos, para INT8, $\alpha_q = -128$ y $\beta_q = 127$.

[10]Existen otras técnicas más avanzadas que tienen en cuenta la existencia de outliers en la distribución, lo que puede aumentar el ruido de cuantización.

Una vez obtenidos $S$, $Z$, $\alpha_q$ y $\beta_q$, ya podemos cuantizar o decuantizar los distintos tensores con la siguiente ecuación.[11]

$$x_q = clip(round(\frac{1}{s}x + z), \alpha_q, \beta_q) \tag{3.8}$$

Siendo clip una operación de saturación que impide que resultado de la cuantización sea un valor sea menor que $\alpha_q$ o mayor que $\beta_q$. En la figura 3.5 esta operación se ve reflejada en los valores que están fuera del intervalo $[\alpha, \beta]$

## Técnicas de Cuantización

Existen dos técnicas de cuantización post entrenamiento:

### Fake Quantization o Cuantización Dinámica:

Los pesos se almacenan en un tipo de datos inferior, pero los parámetros de cuantización $S$ y $Z$ se calculan en tiempo de inferencia.

### Fixed Quantization:

En este caso los parámetros de la cuantización se calculan durante el proceso de cuantización. Este tipo de cuantización requiere utilizar un **conjunto de datos representativo de entrada** durante el proceso de cuantizado, con el que se calcularán los parámetros de cuantización de cada tensor.

## Resultados de la cuantización

Si bien este proceso es muy eficiente como se puede observar en la figura 3.6, tiene un inconveniente muy importante, y es que la salida de una neurona está limitada a tantos valores como como el tipo de dato que utilicemos. En el caso de INT8 o UINT8, una neurona solo puede tener 256 valores distintos en su salida.

Por otro lado, esto no debería ser ningún problema si trabajamos con tareas de clasificación con pocas etiquetas, pero es importante comprobar con un conjunto de prueba que la precisión de la red no se ha visto afectada por la cuantización.

[11] La operación de decuantización debe tener en cuenta los errores de redondeo y clip si se quiere recuperar $x$. Nosotros no tenemos en cuenta los errores ya que no afectan al rendimiento de la red.

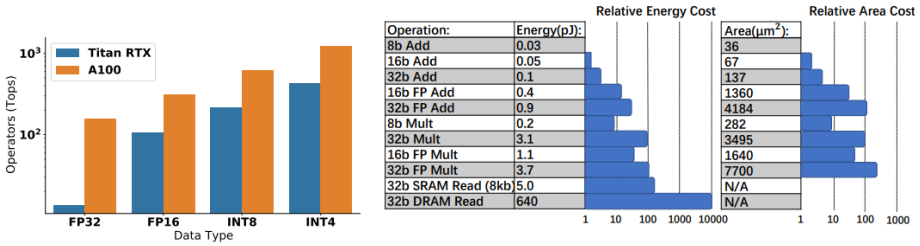
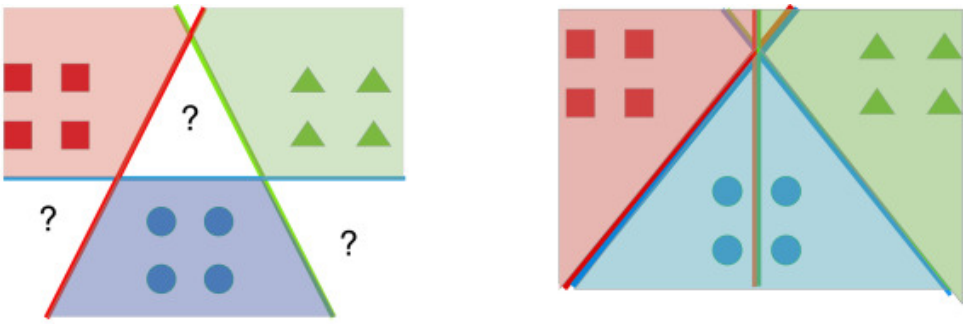



Figura 3.6: Resultados del coste de la operaciones dependiendo del tipo de dato de una red neuronal. Extraído de [3]

## 3.4. Ensembles, OVA y OVO

Existen múltiples técnicas para clasificar conjuntos de datos con múltiples clases. En este aparatado vamos a comentar dos tipos de ensembles que unen varios clasificadores binarios para obtener un único clasificador multiclase. En algunas situaciones, los ensembles binarios pueden ser mucho más robustos que un único clasificador multietiqueta. Hemos planteado el uso de ensembles para mejorar el rendimiento de los distintos clasificadores de clasificación de géneros y estados de ánimo.




(a) Separation with OvA. (b) Separation with OvO.

Figura 3.7: Comparación entre OVA y OVO. Extraído de [4]

### One vs All

El ensemble One vs All (OVA) divide un problema con $N$ clases en $N$ clasificadores binarios. Cada uno de estos clasificadores se encarga de identificar si una entrada se corresponde con una clase (1) o no (0). En este caso, es importante que el clasificador no esté calibrado, es decir, su salida

debe asemejarse con la confianza del resultado, ya que, si nos encontramos con un conflicto entre varios clasificadores base por tener la misma salida, no vamos a poder predecir la clase correctamente.

**Ejemplo Red Neuronal Binaria**

Si utilizamos la **función de activación escalón** en la neurona de salida de nuestra red neuronal, nos encontramos con que la salida únicamente tiene dos valores, 0 y 1, por lo que no conocemos la confianza de la predicción.

Por otro lado, si utilizamos la **función sigmoide** (3.5) como función de activación, nos encontramos con una función cuyo recurrido son valores reales en el intervalo [0, 1], por lo que podríamos considerar que la salida es la confianza que tiene la red ante una predicción.

## One vs One

El ensemble One vs One (OVO), a diferencia del ensemble OVA, busca una clasificación más robusta entrenando un mayor número de clasificadores Si en el ensemble OVA únicamente lidiábamos con las colisiones teniendo en cuenta la confianza, el ensemble OVO es capaz de conocer si un dato pertenece a una clase o a otra, ya que cada uno de los clasificadores se entrena con un subconjunto de datos formado solo por dos clases[12].

En este caso, tenemos que entrenar $L = K(K-1)/2$ clasificadores binarios, un número mucho más grande que en el ensemble OVA, por lo que el coste en memoria y tiempo de entrenamiento es mucho mayor.

En el caso del ensemble OVO, el clasificador base no solo no tiene que estar correctamente calibrado, sino que además las salidas deben estar comprendidas en el intervalo [-1, 1], siendo -1 la clase A, 1 la clase B, y 0 el valor correspondiente la clase desconocida.

Para evaluar la salida de todos los clasificadores, debemos construir una matriz $N \times L$ (3.4), que permita convertir el vector $L$ a un vector con $N$ elementos con la confianza de cada clase. Cada fila de la matriz se corresponde con cada clase, y debemos asignar el mismo valor (1 o -1) que hemos utilizado como etiqueta durante el entrenamiento

Haremos una multiplicación de matrices entre la matriz y la salida de los clasificadores, obteniendo una matriz N x L con la probabilidad de cada clase. Como hemos utilizado valores en el intervalo [-1, 1] como salida de cada clasificador, al realizar la multiplicación de matrices nos encontraremos

[12]Clasificación Binaria

$$M = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ -1 & 0 & 0 & 1 & 1 & 0 \\ 0 & -1 & 0 & -1 & 0 & 1 \\ 0 & 0 & -1 & 0 & -1 & -1 \end{bmatrix}$$

Figura 3.8: Matriz para N=4

con que únicamente tenemos valores positivos.
Si sumamos todas las columnas, de la matriz anterior obtendremos la confianza del ensemble para cada uno de los datos.

## 3.5. Ball Tree

Un ball tree o árbol de bolas es una estructura de datos jerárquica utilizada para la gestión de vecindarios de instancias. Esta estructura de datos permite conocer de una manera eficiente el vecindario de una instancia.

Se ha utilizado esta estructura de datos para buscar los vecinos más cercanos a una canción, como una posible técnica para identificar canciones similares.

Para construir un árbol de bolas, las instancias se dividen en dos clusters o bolas[13], siendo cada cluster una rama del árbol. Los cluseters son calculados de la siguiente manera:

1. Se selecciona una instancia aleatoria
2. Se busca la instancia más alejada a la instancia seleccionada
3. Se calcula el punto intermedio que hay entre las dos instancias
4. Se divide el espacio en dos mediante un hiperplano
5. Se calculan los centroides de los clusters que hay en cada división.
6. Se crea una esfera que cubra todos las instancias de cada clúster

Repetimos el proceso para cada una de las esferas hasta alcanzar el número de instancias por hoja buscado (Por defecto son 40 instancias por hoja en SKLearn).

[13]También conocido como esferas o hiperesferas

Hecho esto, podemos consultar rápidamente el vecindario correspondiente a una nueva instancia a partir de los centroides más cercanos a nuestra instancia, sin necesidad de explorar todos los vecinos posibles.

## 3.6. Sistemas de Recomendación

Un sistema de recomendación es un algoritmo que intenta predecir la valoración o interés que puede tener un usuario ante un nuevo elemento. Existen dos tipos principales de sistemas de recomendación, los sistemas basados en contenido y los filtrados colaborativos.

Los sistemas basados en contenido realizan recomendaciones a partir de las etiquetas o metadatos del elemento. Se ha usado este tipo de sistema de recomendación para recomendar canciones similares basados en características extraídas del audio de las canciones. Se ha explorado el uso de estos sistemas en el apartado 5.4.

Los sistemas colaborativos tienen en cuenta las valoraciones o gustos de otros usuarios para realizar las recomendaciones. Estos sistemas funciona a partir de la creación de una matriz usuario/elemento, en la que se almacena las valoraciones de los usuarios. De esta matriz se puede extraer una matriz usuario/usuario con la similitud que hay entre los gustos de los distintos usuarios. Para estos sistemas de filtrado se recomienda usar Suprise 4.1, una biblioteca que implementa una gran variedad de algoritmos para trabajar con sistemas colaborativos. Se ha usado este sistema de recomendación para rellenar playlists de usuario, como explora el apartado 5.4.

# Técnicas y Herramientas

## 4.1. Tecnologías y Dependencias

En este apartado se van a mencionar las diferentes tecnologías, bibliotecas y frameworks que han sido utilizados para construir el frontend y los dos servidores web de SpotMyFM.

### Frontend

El frontend es un cliente web ajeno a Spotify dedicado a expandir la funcionalidad de la plataforma mediante su API pública.

#### Typescript

Typescript es un superset del lenguaje de programación Javascript, por lo que cualquier código escrito en Javascript puede ser utilizado como código Typescript. La principal característica de este superset es la inclusión de un sistema de tipado estático fuerte.

Para el desarrollo de los componentes se ha utilizado TSX (Typescript Syntax Extensions), esta extensión de Typescript permite generar HTML junto con Typescript, facilitando la generación de HTML dinámico.
Estas extensiones de lenguaje no están reflejadas en el estándar de ECMAscript, por lo que es necesario utilizar un transpilador que convierta estas extensiones en código Javascript.

Se ha tenido en cuenta utilizar JavaScript y JSX como alternativa.

Página oficial: https://www.typescriptlang.org/

### NextJS

NextJS es un framework de Javascript compatible con Typescript que permite construir interfaces. NextJS está construido sobre la biblioteca ReactJS, y añade una gran variedad de características como:

- Rutas o Páginas (/home, /about, etc)
- Server Side Rendering
- Rutas de API basadas en NodeJS
- Empaquetamiento sin configuración
- Api de internacionalización

ReactJS es una biblioteca de desarrollo de interfaces web que incluye un Virtual DOM que permite montar y desmontar componentes JSX en el DOM de manera transparente.

Se ha tenido en cuenta utilizar ReactJS como una alternativa.

Página oficial: https://nextjs.org/

## i18next y Next Translate

Se ha utilizado la biblioteca i18next junto con el adaptador Next Translate para la internacionalización del frontend.

i18next es una mejora sobre i18, una de las biblioteca de internacionalización más utilizadas del mundo. Next Translate incluye un Hook

```
useTranslation()
```

Las traducciones se almacenan en ficheros .json, en el que la clave es el identificador de la traducción y el valor es el string traducido.

### Tailwind CSS

Tailwind CSS es una biblioteca de generación de hojas de estilo CSS de código abierto muy interesante. Su principal filosofía es la de la creación de clases de utilidad dinámicas con las que podemos seguir buenas prácticas de CSS. Esta biblioteca está acompañada con un preprocesador que se encarga de generar y optimizar las stylesheets a partir de las clases que hemos utilizado.

Una de sus características más interesantes es la generación de clases de utilidad con variantes que permiten cambiar el estilo dependiendo de algunos condicionales.
Algunas de estas variantes de estado pueden ser:

- hover:<clases de utilidad> estilo cuando el ratón está encima de un elemento
- sm:<clases de utilidad> estilo cuando el ancho del viewport es mayor a 640px
- dark:<clases de utilidad> estilo a aplicar cuando estamos utilizando el modo oscuro
- ...

Por último, esta biblioteca permite declarar nuestras propias paletas de colores, variantes de estado, clases de utilidad, etc, por lo que podemos expandirla manualmente.

Página oficial: https://tailwindcss.com/

### Twin Macro y Styled Components

En la actualidad, debido a las limitaciones de las hojas de estilo en cascada, existen múltiples métodos de escribir estilos CSS, como los ficheros .css, estilos en línea, estilos como componentes o CSS in JS.

Para esta aplicación hemos utilizado el paradigma CSS in JS, que nos permite declarar los estilos CSS en el propio código de la aplicación. La principal ventaja de este paradigma es que podemos modificar los atributos CSS directamente desde nuestro código.

Twin macro es un adaptador de distintas bibliotecas **CSS in JS**, que permite estilar mediante TailwindCSS con bibliotecas CSS in JS como PostCSS, Emotion o Styled Componets.

En este caso hemos escogido la biblioteca **Styled Coponentes**, que nos permite declarar o estilar componentes de ReactJS.

Repositorio oficial: Twin Macro
Página oficial: https://styled-components.com/

### Zustand

Zustand es una biblioteca de código abierto para la gestión de estado de React basada en hooks. Esta biblioteca permite declarar stores (espacios de almacenaje de estados) en los que podemos almacenar datos y sus operaciones que deseamos compartir entre varios componentes.

En el caso de la aplicación, hemos utilizado esta biblioteca para almacenar estados globales, como el estilo de la web, la pestaña que tenemos activa o instancias únicas de los distintos clientes REST de la aplicación.

Página oficial: https://zustand.surge.sh/

### DexieDB

DexieDB o DexieJS es un wrapper de IndexedDB, una **base de datos** NO-SQL que se encuentra en todos los navegadores modernos.

Este wrapper, a parte de contener toda la funcionalidad de DexieDB, nos permite declarar los esquemas de la base de datos junto sus distintas operaciones y comprobaciones.
DexieDB es compatible con Typescript, por lo que permite operar las tablas con supervisión de tipos.

Página oficial: https://dexie.org/

### Jest

Jest es una biblioteca de código abierto de pruebas unitarias desarrollada por Facebook. Esta biblioteca tiene una gran variedad de características, como Mocking, compatibilidad con código asíncrono, compatibilidad con React, Typescript, Node, etc.

Una de las principales ventajas de Jest son las extensiones de su comunidad. En este caso se ha utilizado la extensión **React Testing Library**, una extensión que permite renderizar internamente componentes de React y hacer comprobaciones sobre el DOM Virtual.

Página oficial: https://jestjs.io/

### Cypress

Cypress es un framework de pruebas punto a punto de código abierto. Cypress es muy similar a Selenium, ambas realizan pruebas automatizadas sobre un navegador físico, pero Cypress es mucho más sencillo de configurar.

Algunas de las características más relevantes de Cypress son: velocidad de ejecución, grabación de los tests fallidos, tamaño de viewport variable, ejecución headless y compatibilidad con Github Actions 4.2

Página oficial: https://www.cypress.io/

**Otras:**

- **Recharts**: Es una biblioteca de código abierto para React que permite generar gráficas interactivas. Página oficial: https://recharts.org/
- **React Icons**: Una colección de iconos vectoriales que junta múltiples bibliotecas de iconos. Página oficial: https://react-icons.github.io/react-icons
- **Framer Motion**: Biblioteca de código abierto que permite añadir animaciones que dependen de los estados internos de un componente. Página oficial: https://www.framer.com/motion/

## Servidor NextJS

El servidor de NextJS se encarga de la gestión de la base de datos, inicio de sesión, generación de tokenes de identificación JWT y de la comunicación con el servidor de recuperación de información musical.

### NextJS API REST

NextJS incluye una API REST funcional basada muy simple basada en Node/Express. Esta API tiene como principal objetivo apoyar al funcionamiento del cliente en dos situaciones.

- Renderizar en el servidor (SSR)[14] elementos dinámicos de la web. Como por el ejemplo vistas individuales para usuarios registrados.
- Trabajar con peticiones que requieran autenticación.

En este api, un endpoint se define como un fichero .ts/.js que recibe dos objetos, la petición HTTP y la respuesta HTTP. Como cada endpoint se corresponde con un fichero único, la API permite el paradigma de cloud computing FaaS (Functions as a Service), en el que cada función puede actuar como un pequeño microservicio.

[14]Server Side Rendering

Cada endpoint puede alojarse de manera independiente en plataformas como Cloudflare Workers o AWS Lambda.

### JWT

JWT (Json Web Tokens) es un método de representación de tokens en formato json basado en el estándar RFC 7519. Los tokens están formados por tres segmentos:

- Cabecera: Es un json serializado en base64 que incluye el tipo del token y el algoritmode firma utilizado (RSA, SHA256, etc)
- Payload: Es otro json serializado en base 64 que incluye los contenidos del token.
- Firma de verificación: Firma hash generada a partir de una clave privada y los campos anteriores. Permite comprobar que el payload no ha sido modificado.

Una vez construidos y serializados los 3 segmentos, estos son concatenados mediante un punto (".") y pueden ser deserializados por cualquier receptor.

Se han utilizado estos tokenes para poder identificar al usuario y sus operaciones permitidas. Este token se envía a la API para todas las peticiones, y a diferencia de identificar al usuario con su token de usuario de Spotify, al solo comprobar la validez de la firma no se tienen que enviar más peticiones a Spotify, mejorando la latencia de cada endpoint.

Biblioteca específica utilizada: https://www.npmjs.com/package/jsonwebtoken

### DynamoDB y Dynamoose

**DynamoDB** es una base de datos NO-SQL basada en documentos clave valor desarrollada por Amazon para AWS, su plataforma de Cloud Computing.
Algunas características de esta base de datos son:

- Capacidad bajo demanda
- Base de datos sin servidor (No utiliza sockets / conexiones físicas, por lo que es apta para ser accedida por la arquitectura sin servidor, como CaaS o FaaS.

- Compatibilidad con AWS OpenSearch (E lasticSearch)
- Copias de seguridad
- Replicas
- Transacciones ACID

Además, DynamoDB está incluida en el plan gratuito de AWS, y ofrece 25GB de por vida.

**Dynamoose** es una herramienta de modelado de DynamoDB que permite utilizar las distintas características de AWS desde un api de alto nivel.

La principal ventaja de Dynamoose es que permite modelar los esquemas de la base de datos en Typescript/Javascript. Estos esquemas serán validados por Dynamoose antes de crear, modificar o leer documentos.

**Otras Bases de datos planteadas**

| | Firestore | MongoDB | Supabase | DynamoDB | Heroku |
|---|---|---|---|---|---|
| **Tipo** | Documentos | Documentos | Relacional | Documentos | Relacional |
| **Capacidad** | 1GB | 0.5GB | 0.5GB | 25GB | 1000 filas |
| **Lim. Peticiones** | 50k R/doc/day<br>20k W/ doc/day | No hay | No Hay | 200M R+W /mes | No |
| **Self-hosted** | No | Si | Si | NO | Si |
| **SSL** | Si | Si | Si | Si | Si |
| **Escalable** | Si | Si | Si | Si | Si |
| **Pausa** | No | No | Si (7 días) | No | Si (7 días) |
| **ORM/ Bussiness Logic / Model / Validation** | No | Si (Mongoose) | No | Si (Dynamoose) | Si (Postgres) |
| **Editor** | Si | Si | Si | Si (De pago) | Si |

Página oficial de DynamoDB: https://aws.amazon.com/es/dynamodb/

Página oficial de Dynamoose: https://dynamoosejs.com/

## Backend MIR

El backend de **recuperación de la información musical** está basando en Python 3.8 y permite obtener las etiquetas y canciones similares a partir de un fichero de audio.

### FASTAPI

FastAPI es un framework web diseñado para construir APIs con Python 3.6+.

La principal característica de FastAPI es que utiliza el sistema de sugerencia de tipos de Python para validar las distintas peticiones HTTP.

Este sistema permite asignar un tipo de forma estática a una variable, estos tipos son ignorados en tiempo de ejecución, pero pueden se utilizados para documentar o como comprobación en tiempo de desarrollo mediante un linter.

```
foo: int = 1 # No error
foo: int = "bar" # typechcking error
```

Además de validar las peticiones de nuestra aplicación, FastAPI genera automáticamente **documentación interactiva** para cada endpoint con **OpenAPI y Redocs**.

Por último, FastAPI utiliza el api de **corrutinas** de Python basada en la sintaxis Async/Await. Cada uno de los endpoints puede ser definido como una corrutina asíncrona con la que podemos esperar a ejecución de otras corrutinas utilizando la palabra clave await, facilitando el uso de bibliotecas concurrentes.

Página oficial: https://fastapi.tiangolo.com/

### AIOHTTP

**aiohttp** es una biblioteca de Python basada en **requests** que permite realizar **peticiones http concurrentes**. aiohttp utiliza el api de corrutinas, por lo que puede sen consumida directamente desde un endpoint de FastAPI.

Página oficial: https://docs.aiohttp.org/en/stable/

### ONNX Runtime Python

**ONNX Runtime** [8] es una biblioteca de código abierto mantenida por Facebook y Microsoft. Esta biblioteca implementa múltiples motores de inferencias compatibles con el estándar **Open Neural Network eXchange (ONNX)**, un estándar que busca la interoperabilidad de modelos neuronales a partir de una **representación intermedia (IR).**

La filosofía de ONNX es que un modelo neuronal pueda ser utilizado en cualquier hardware independientemente de su representación interna, para ello existen una serie herramientas para convertir y optimizar modelos basados en Pytorch, Keras, Caffé, etc en representación intermedia.

Un modelo IR puede ser utilizado en múltiples **plataformas**, como Windows, Linux, Android, Ios, navegadores web, . . . , e independientemente de la arquitectura del procesador (X86, X64, ARM, . . . ).

La principal ventaja de ONNX es su compatibilidad con diferentes lenguajes de programación, ya que permite inferir modelos desde C++, Python, Java, C# o incluye NodeJS/JS/TS. Por último, ONNX incluye aceleración por hardware con una gran cantidad de APIs, como Nvidia CUDA, Nvidia TensorRT, Intel OpenVINO, Windows DirectML, . . . .

Página oficial: https://onnx.ai/

### Librosa

Librosa es una biblioteca de Python enfocada al análisis de audio y música. Esta biblioteca está basada en Scipy y es muy utilizada a la hora de trabajar con sonido en Python.

Esta biblioteca incluye todos los componentes necesarios para poder trabajar con audio, desde la lectura / creación de ficheros de audio, conversión entre formatos, muestreo, visualización y extracción de características.

En este apartado del proyecto utilizamos librosa para leer ficheros wav y extraer algunos datos como los MFCCs 3.1, Tempo o el Beat:

Se ha tenido en cuenta el uso de Torch Audio para las tareas de preprocesamiento.

Página oficial: https://librosa.org/

### FFMPEG

FFMPEG es conjunto de bibliotecas de **alto rendimiento** que permiten trabajar con datos multimedia (ficheros y streams de vídeo, audio, imágenes, etc). FFMPEG permite grabar, editar, convertir, transcodificar o escalar datos multimedia.

El servidor de recuperación de información musical utiliza FFMPEG para convertir y muestrear ficheros .mp3 a .wav en 22050HZ, ya que es la representación de onda en formato .wav es la misma con la que trabaja

Librosa. Librosa es compatible con formatos .mp3, pero su implementación no es tan eficiente como la de FFMPEG.

Página oficial: https://ffmpeg.org/

### Sklearn

Sklearn [9] es una biblioteca basada en SciPy [10] diseñada para tareas de aprendizaje automático.

Incluye una gran variedad de algoritmos para tareas de clasificación, regresión, clustering, reducción de dimensionalidad, selección de modelo o preprocesado.

Este backend utiliza dos componentes de Sklearn:

- **BallTree**: Utilizado para encontrar los vecinos más cercanos a un dato.
- **MinMaxScaler**: Utilizado para normalizar la entrada al BallTree

Página oficial: https://scikit-learn.org/

Knime Knime o Konstanz Information Miner es una herramienta de código abierto usada en minería de datos y aprendizaje automático. Esta herramienta permite construir de manera visual mediante nodos interconectados flujos de trabajo para visualizar, modificar y analizar datos. Se ha utilizado KNime para el análisis de los distintos conjuntos de datos.

Página oficial: https://www.knime.com/

### Scikit Surprise

Surprise [11] es una biblioteca de código abierto especializada en sistemas de recomendación colaborativos.

El funcionamiento de esta biblioteca es muy simple, ya que permite generar la matriz de recomendación a partir de una tabla con tres columnas, el ID del usuario, el ID del Elemento y un escalar con la votación del usuario.

Surprise implementa una gran variedad de algoritmos, como SVD o NMF utilizados en descomposición de matrices, y otros más tradicionales como KNN, Co Clustering o Slope One.

Por otro lado, Surprise además implementa un sistema de evaluación de recomendaciones, que valora la precisión del modelo a partir de intentar completar recomendaciones para usuarios incompletos.

Página oficial: http://surpriselib.com/

## 4.2. Herramientas y Servicios

### Kaggle

Kaggle es una comunidad de ciencia de datos en la que se publican conjuntos de datos y competiciones.

Uno de los servicios que incluye Kaggle es **Code**, un servicio para publicar y ejecutar notebooks que utilicen el Dataset con aceleración por hardware. En este caso se ha utilizado Kaggle para publicar los datasets y entrenar los modelos de forma remota. Se ha planteado el uso de Google Cloud Notebooks y Google Colab para el entrenamiento de los modelos.

Página oficial: https://www.kaggle.com/

### Tensorflow Keras

Tensorflow [12] es una biblioteca de código abierto destinada a el entrenamiento e inferencia de modelos neuronales. Tensorflow está pensada para ser usada con Python, pero es compatible con otros lenguajes como C++ o incluso JavaScript.

Keras es una API de Tensorflow para Python que simplifica la implementación y entrenamiento de redes neuronales mediante Tensorflow.

Se ha planteado el uso de Pytorch como alternativa a Tensorflow Keras.

Página oficial: https://www.tensorflow.org/

### ONNX Tensorflow Converter y ONNX Quantizer

ONNX Tensorflow Converter [8] es una herramienta de línea de comandos que permite convertir un modelo de Tensorflow en un modelo del estándar ONNX.

ONNX Quantizer es una biblioteca de Python que permite cuantizar modelos de ONNX. Esta herramienta es compatible tanto con la cuantización dinámica como con la estática.

Ambas herramientas forman parte del proyecto de código abierto ONNX, mantenido por Microsoft y Facebook.

Se ha tenido en cuenta el uso de TFLITE para cuantizar e inferir los modelos, pero este motor únicamente está optimizado para dispositivos móviles (ARM).

Página oficial: https://onnxruntime.ai/

### Github

Github es una plataforma para alojar repositorios git. Esta plataforma permite crear y compartir repositorios públicos y privados, utilizar ramas, crear y gestionar pull requests, y muchas otras características como la creación de flujos de trabajo con Github Actions o el control de dependencias con Dependabot.

Se ha utilizado Github y su sistema de Issues para el control de versiones del proyecto y la planificación temporal del proyecto.

Repositorio del proyecto https://github.com/JorgeRuizDev/SpotMyFM

### Zenhub

Zenhub es una herramienta de gestión de proyectos que intenta integrarse con Github. Esta biblioteca presenta un tablero Kanban en el que podemos encontrarnos cada una de las tareas del propio repositorio.

Zenhub, además tiene un sistema de sprints, épicas y puntos de historia con el que podemos gestionar proyectos utilizando la metodología Scrum.

Página oficial: https://www.zenhub.com/

### Dependabot

Dependabot es un servicio integrado en Github que permite analizar un respositorio en busca de brechas de seguridad.
Este servicio analiza el código en busca de **filtraciones de claves o tokens secretos**, y como su propio nombre indica, analiza las **dependencias del proyecto** en busca de dependencias vulnerables.
Si Dependabot detecta una vulnerabilidad en el proyecto, este intentará corregirla y publicará una **Pull Request** con la vulnerabilidad solucionada.

## Docker

Docker es una plataforma que permite crear y desplegar contenedores. Un contenedor es un "paquete" que contiene todos los requisitos para que una aplicación se pueda ejecutar en un entorno aislado, como el binario de la aplicación, dependencias, configuraciones de red, etc.

Estos contenedores pueden ser considerados una pequeña máquina virtual, ya que cada contenedor tiene su propio sistema de ficheros, unidades, o LAN, pero a diferencia de una máquina virtual, el contenedor se ejecuta sobre el **Docker Engine** y no sobre un **hipervisor.**

En este caso se ha utilizado Docker para crear los contenedores con los que poder desplegar las aplicaciones de NextJS y FastAPI.

Página oficial: https://docs.docker.com/

## Github Copilot

Github Copilot es un servicio online que permite sugerir cambios o implementaciones de código basado en el contexto actual del fichero que estamos editando. Copilot se integra con el editor (VSCODE) o IDE (Webstorm) y necesita conexión a internet para procesar las sugerencias.

La principal ventaja de Copilot frente a otros servicios de autocompletado como Intellisense es que Copilot utiliza los tipos de datos, comentarios, estilo del código, nombres de variables y nombres de funciones para sugerir cambios o implementaciones.

Página oficial: https://github.com/features/copilot

## Github Actions

Github Actions es un servicio integrado en Github que permite ejecutar flujos de trabajo sobre un repositorio a partir de distintos estados.

En este caso, se han programado las siguientes acciones para cada push al repositorio:

- **Prettier**: Aplica el formateador de código Prettier a todos los ficheros compatibles (Typescript, Javascript, JSON, CSS, etc)
- **Integración Continúa**: Asegura que el proyecto se compila correctamente y ejecuta las pruebas unitarias sobre las distintas clases del proyecto.

- **Cypress:** Ejecuta las pruebas punto a punto sobre el despliegue de la aplicación.

Se ha considerado el uso de los servicios TravisCI y CircleCI.
Página oficial: https://github.com/features/actions

## Vercel (Plataforma)

Vercel es una plataforma para publicar proyectos de frontend y webs estáticas. Está desarrollado por la empresa Vercel, la misma empresa que desarrolla NextJS 4.1, por lo que es compatible con todas las funcionalidades de NextJS.
Vercel ofrece muchas otras características como la compatibilidad con Edge Functions (FaaS), despliegue continuo, SLL o dominios personalizados.

Se ha planteado el uso de la plataforma Netlify pero se ha escogido Vercel por su integración con NextJS.

Página oficial: https://vercel.com/

## Google Cloud Run

Google Cloud Run es un servicio de Google Cloud Platform que permite el despliegue de contenedores (CaaS)[15] de forma escalable.
Su funcionamiento es muy simple, a partir de un contenedor con un puerto expuesto, Cloud Run creará un proxy para dicho puerto y dependiendo del tráfico el orquestador instanciará o mantendrá activos un número variable de contenedores.

Página oficial: https://cloud.google.com/run

## Visual Studio Code

Visual Studio Code es un editor de código de código abierto desarrollado por Microsoft. Este editor está construido sobre Chromium y Electron, con una filosofía en la que la interfaz está totalmente separada del editor, por lo que el editor en si puede encontrarse en un servidor, máquina virtual o local, mientras que editamos desde un navegador web o la propia aplicación de escritorio.

[15] Containers as a Service / Contenedores como Servicio

La principal ventaja de Visual Studio Code es su gran comunidad, que ha desarrollado un gran número de extensiones para ajustar el desarrollo del editor para cada uno de los casos de uso.

Página oficial: https://code.visualstudio.com/

### OpenAPI

La especificación OpenAPI es una especificación para definir la interfaz de un servicio REST. Esta interfaz puede declararse mediante un fichero .json o .yaml, y permite declarar los distintos endpoints de un servidor, así como las peticiones y respuestas esperados.

La principal ventaja de OpenAPI es que puede ser integrada con un middleware para ofrecer validación de la API.

Por otro lado, la especificación OpenAPI permite generar una interfaz interactiva con la que poder generar automáticamente peticiones de muestra a partir de la especificación de peticiones.

Página oficial https://www.openapis.org/

### Lucidchart

Lucidchart es una aplicación web progresiva diseñada para la creación y exportación de diagramas. Su principal característica es su gran cantidad de herramientas, así como la posibilidad de editar un diagrama de forma colaborativa. Lucidchart dispone de un plan gratuito que incluye hasta tres proyectos, cada uno puede albergar infinitos diagramas.

Se ha planteado el uso de draw.io, pero se ha preferido usar Lucidchart por su servicio en la nube.

Página oficial: https://lucid.app/lucidchart

### Overleaf y LaTeX

Overleaf es un editor web de documentos LaTeX. LaTeXes una extensión de TeX, un sistema de composición de textos orientado al ámbito científico y matemático, es ampliamente utilizado debido a su estilo y a su procesador de expresiones matemáticas. LaTeX, a diferencia de otros sistemas como DOCX, está pensado para que los textos sean editados directamente en texto plano, sin necesidad de un editor específico. Esto es posible gracias al sistema de macros y comandos que permiten estilar y organizar un documento.

Página oficial: https://www.overleaf.com/

### Figma

Figma es una aplicación web progresiva destinada a la edición de imágenes vectoriales. Su caso de uso principal es de prototipado de interfaces de usuario, pero puede usarse para muchos otros casos de uso, como creación de esquemas, creación de logos o incluso para la edición de diagramas UML.

Página oficial: figma.com

## 4.3. Técnicas y Metodologías

### Test Driven Development

Test Driven Development [13] es una metodología de desarrollo agile en la que se implementan los tests unitarios antes que la funcionalidad de la aplicación. Sigue el esquema de tres fases **red**, **green**, **refactor**. La fase red indica que los tests deben fallar. La fase green indica que se debe implementar de forma correcta la utilidad para que los tests se ejecuten satisfactoriamente, y por último la fase refactor indica que implementación debe ser refactorizada para mejorar la mantenibilidad.

Se ha empleado este paradigma de desarrollo durante la implementación de los endpoints de las APIs, clientes REST y componente auxiliares del proyecto.

### Metodología Scrum

Scrum es una metodología de gestión de proyectos a nivel de equipo. Scrum se centra en dividir las tareas del equipo en pequeñas tareas que puedan ser cumplidas durante un sprint.
No se ha seguido esta metodología al pie de la letra debido a que solo ha participado un único desarrollador en el proyecto, por lo que algunos conceptos como los roles de miembros o las reuniones diarias no se aplican. En este caso, se han realiza sprints con una duración de 2 semanas y reuniones para revisar los sprints al final de cada sprint. Se ha acompañado esta metodología con Zenhub 4.2 para definir las pilas de sprint y producto, estas pilas contienen las tareas que se van a realizar en el sprint actual y a lo largo del proyecto ordenadas por prioridad.

# Aspectos relevantes del desarrollo del proyecto

## 5.1. Datasets

### GTZAN Dataset

Las versiones iniciales del clasificador fueron desarrolladas sobre el conjunto de datos para clasificación de música GTZAN.

Este conjunto de datos [14] fue creado por George Tzanetakis en el año 2001 a partir de pequeños fragmentos de 30s extraídos de CDs, DVDs, Casettes y grabaciones de la radio. Es uno de los conjuntos de datos más utilizados en los sistemas de reconocimiento de géneros musicales.

Este conjunto de datos está formado por 10 géneros, cada uno con 100 segmentos diferentes en formato .wav, mono, 16 bits de profundidad y una frecuencia de muestreo de 22050HZ. Los géneros son: Blues, Classical, Country, Disco, Hip Hop, Jazz, Metal, Pop, Reggae y Rock.

Si bien este conjunto de datos es muy variado al contener 10 géneros muy diferentes entre sí y múltiples fuentes de audio, presenta varios problemas:

- **Etiquetas incorrectas**: Estos es debido a que la clasificación de música es una tarea complicada incluso para el ser humano, siendo la precisión media del 70 %, como explora [15].
- **Distorsiones**: Si bien las distorsiones son una característica del conjunto de datos, es recomendable distorsionar la música mediante preprocesamiento, ya que no todos los casos de usuo requieren de un

conjunto de datos con distorsiones. Los problemas relacionados con la calidad del conjunto de datos han sido explorados en [16].

- **Conjunto Reducido**: Únicamente disponemos de 100 segmentos de 30s por etiqueta.
- **Conjunto Obsoleto**: El conjunto fue creado a principios de los años 2000, por lo que no refleja los estilos de las dos últimas décadas de música.

### GTZAN Extended

Como alternativa al conjunto de datos GTZAN, se decidió crear una versión extendida con 300 canciones por cada género. Este conjunto de datos fue creado a partir de playlists de Spotify creadas por la comunidad o por la propia Spotify.

Para esta tarea se ha utilizado la herramienta **dataset-tools node**, introducida en el manual de programador, que permite descargar las previsualizaciones de canciones de 30s. Para seleccionar las playlists se ha usado el explorador de Spotify, y se han descargado playlists del género en concreto.

Si bien este nuevo conjunto de datos extendido incluye 2000 canciones nuevas, fue creado a partir de las canciones más populares de Spotify del siglo XX, por lo que no tenemos una variedad real de canciones.

Este conjunto de datos extendido ha sido publicado en Kaggle [17].

### Ludwig Dataset

El Ludwig Dataset [18] es un conjunto de datos de música creado específicamente para este proyecto.

Existen una gran variedad de conjuntos de datos destinados al análisis de música, pero no hay ninguno que cumpla con todos los requisitos del proyecto.

- Géneros
- Subgéneros
- Estados de Ánimo
- Ficheros de Audio
- Canciones asociadas con Spotify
- Etiquetas Supervisadas.

Para poder crear este conjunto de datos, ha sido necesario cruzar tres fuentes de datos distintas. El proceso de generación ha sido detallado mediante un diagrama en el anexo C (Diseño).

- **Discogs**: Discogs [19] es el Marketplace de música más importante del mundo, cada item está categorizado por género y subgéneros. Discogs dispone de una API pública que permite obtener datos sobre los distintos elementos.
- **AcousticBrainz y MusicBrainz**: MusicBrainz [20] es una base de datos colaborativa de música, no es muy completa pero cada elemento tiene un MBID que es compartido con AcousticBrainz y Discogs.

  AcousticBrainz es un proyecto del Music Technology Group de la Universitat de Pompeu Fabra [21] que permite analizar y clasificar canciones a partir de la onda sonora.
  Este proyecto incluye una API pública en la que se pueden consultar detalles de canciones, como estados de ánimo o etiquetas obtenidas a partir de múltiples clasificadores. Se puede consultar los datos de una canción a partir de su MBID.
- **Spotify**: La API pública de Spotify [22] dispone de un endpoint de búsqueda a partir del nombre. La gran mayoría de las canciones de Spotify permiten descargar un fragmento de 30s con la parte más relevante de la canción.

Además de las canciones en formato .mp3, cada una de las canciones tiene dos ficheros de numpy (*.npy*) asociados con 10 segmentos de 3s con los MFFCs y Espectrogramas en Frecuencias de Mel extraidos mediante Librosa.

El dataset contiene las siguientes etiquetas, e incluye un fichero **subgenres.json** que incluye una 600 canciones por cada uno de los subgéneros.
**Géneros y Subgéneros**

- Rock

  **Subgéneros:** Pop Rock, Hard Rock, Prog Rock, Rock Alternativo, Goth Rock, Art Rock, New Wave, Shoegaze.
- Metal

  **Subgéneros:** Heavy Metal, Nu Metal, Death Metal.

- Punk

  **Subgéneros:** Punk, Post Punk.

- Blues

  **Subgéneros**: Country Blues, Electric Blues.

- Latina

  **Subgéneros**: Reggaeton, Salsa, Flamenco, Reggae, Samba, Cubano.

- Hip Hop

  **Subgéneros**: Trap, Pop Rap, Instrumental, Conscious, Gangsta, Trip Hop

- Pop **Subgéneros**: Indie Pop, Europop, Ballad Pop

- Jazz

  **Subgéneros**: Jazz Contemporáneo, Swing Jazz, Soul Jazz

- Clásica

  **Subgéneros**: Clásica, Romántica, Barroca, Moderna, Ópera,

- Electrónica

  **Subgéneros**: Ambient, Synth Pop, Disco, House, Drum n Bass, Downtempo, Electro, Trip Hop

- Funk / Soul

  **Subgéneros**: Dsico, R&B, Soul

**Estados de Ánimo**

- Feliz / Triste
- Electrónica / Acústica
- Relajada / Fiesta
- Agresivo

Este conjunto de datos ha sido publicado en Kaggle [18].

### Dataset Recomendaciones

Se ha generado un dataset de recomendaciones a partir de playlists aleatorias de Spotify extraídas del dataset oficial Millon Playlist Dataset [23]. El dataset aleatorio contiene 32.000 canciones[16] con una popularidad de al menos el 30 %.

Este dataset está diseñado para intentar completar una playlist incompleta mediante la biblioteca Surprise 4.1 mediante filtrado colaborativo. No obstante, se han obtenido todos los ficheros .mp3 de las canciones para intentar complementar este sistema de filtrado colaborativo con un sistema basado en los contenidos de las canciones.

## 5.2. Unificación de las fuentes de datos

Uno de los mayores desafíos encontrados durante el diseño e implementación del Frontend ha sido la unificación de las distintas fuentes de datos en la capa de presentación.

Esto es debido a que ninguna de las APIs comerciales que han sido utilizadas permiten lanzar consultas o realizar búsquedas sobre elementos específicos.
Por ejemplo, la API de Spotify nos permite conocer todos los álbumes que ha guardado un usuario, pero no podemos limitar esta consulta a todos los álbumes guardados por el usuario de un artista en concreto.

Si queremos implementar una funcionalidad similar en nuestra aplicación necesitamos conocer de antemano todos los elementos del usuario, así como los distintos atributos de cada uno de los elementos.

### Spotify

Uno de los principales desafíos a la hora de integrar las distintas fuentes de datos con el frontend ha sido la API de Spotify. Esta API no devuelve los mismos datos a la hora de obtener un mismo recurso de varios endpoints distintos, por lo que los datos con los que trabajamos no son homogéneos.

Desde el punto de vista de nuestra implementación, este diseño de la API no nos beneficia ya que en algunos casos nos obliga a realizar varias peticiones sobre un mismo recurso para obtener todos sus atributos.
Por otro lado, esta implementación tiene sentido desde el punto de vista

[16] El nombre de cada canción contiene el id de Spotify, {id_spotify}.mp3

del tamaño de las consultas, ya que si deseamos conocer las últimas 20 canciones que ha escuchado un usuario, es posible que no en todos los casos necesitemos conocer en que año se publicaron cada una de las canciones.

Tras realizar un estudio sobre la API, nos encontramos que todos los objetos que devuelve la API tienen tres tipos de objetos dependiendo de sus atributos:

- **Objeto Base**: Sus atributos son compartidos por todos los objetos de la API.
- **Objeto Simple**: Compuesto por los atributos que van a tener como mínimo todos los objetos de un tipo específico.
  Por ejemplo, una canción además de tener sus atributos base, siempre va a incluir el objeto base del álbum en cualquier petición.
- **Objeto Completo**: Este objeto contiene todos los atributos posibles de un objeto.

El Frontend consume objetos completos, por lo que si al realizar una petición recibimos un objeto que no sea completo, debemos obtener el objeto completo de dichos elementos para poder trabajar con datos homogéneos.

### LastFM

El principal inconveniente del diseño de la API de LastFM[24] es que únicamente permite realizar una consulta por recurso, por lo que si queremos detallar los datos de 50 álbumes distintos debemos realizar 50 peticiones por separado.
Realizar un gran número de peticiones simultáneas es ineficiente por los siguientes motivos:

- Se realizan 50 consultas sobre la base de datos en vez de una única consulta.
- Se encolan múltiples peticiones HTTP.
- Se realizan múltiples consultas HTTP desde el cliente, saturando el único hilo de ejecución de JavaScript.

La solución propuesta ha sido acumular todas las peticiones individuales en bloques de 50 peticiones, delegando la tarea de lanzar 50 peticiones individuales al backend.

Por último, para evitar que estas peticiones en bloque sean extreamadamente lentas, el backend las realiza de forma concurrente para acelerar el proceso.

## Analizador de Canciones

Una de las características del frontend es poder visualizar y filtrar por los resultados del servidor de análisis de canciones a partir de un fichero de audio correspondiente a un fragmento de la canción.

Este proceso es relativamente lento, ya que para cada canción es necesario realizar los siguientes procesos:

- Descargar el fichero de audio
- Convertir el audio a formato WAV.
- Muestrear el audio a una tasa de muestreo común.
- Extraer los MFCCs
- Inferir los coeficientes en un clasificador de estados de ánimo.
- Inferir los coeficientes en múltiples clasificadores de géneros y subgéneros.

De todos los procesos mencionados, los dos primeros pasos son los más lentos.

- La descarga está limitada por la latencia red de distribución de contenidos.
- La conversión a WAV es un proceso pesado.

Para poder acelerar estos procesos se ha planteado la arquitectura CaaS 5.1, que permite distribuir las peticiones contra este analizador entre varios contenedores. Estos contenedores contienen una API REST que permite analizar un bloque de hasta 25 canciones. Como balanceador de carga se ha utilizado Cloud Run 4.2, que expone un único Endpoint e instancia nuevos contenedores a medida que el sistema recibe peticiones.
Para evitar saturar la memoria de los contenedores, se ha implementado una cola de espera que evite que los contenedores pueden analizar una gran cantidad de peticiones de forma concurrente.

Como el proceso es extremadamente pesado, se ha implementado en la base de datos una tabla que almacena los resultados del analizador.

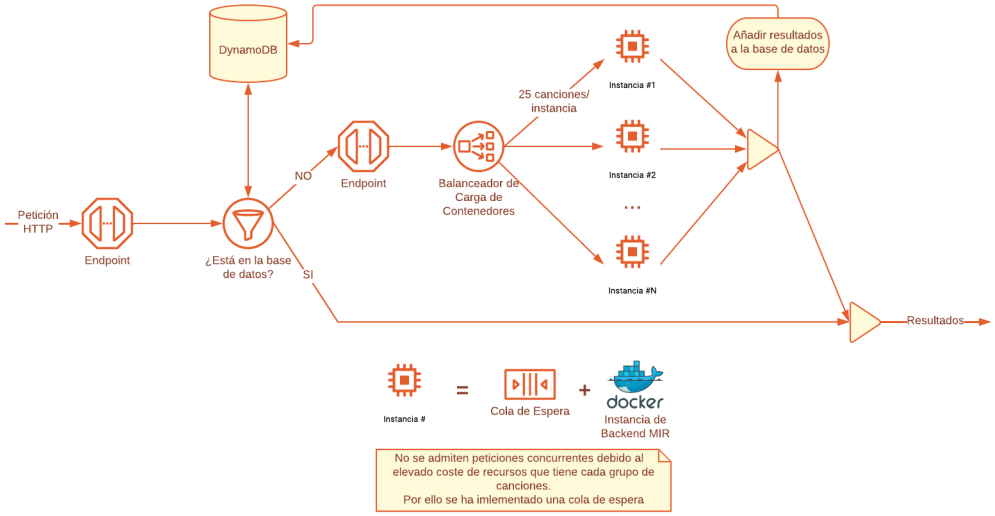



Figura 5.1: Diagrama de la arquitectura CaaS del analizador.

## Etiquetas de Álbumes

La última fuente de datos del sistema es una tabla NOSQL que almacena las etiquetas personalizadas de los distintos usuarios.

Esta fuente de datos contiene datos dinámicos, que se pueden modificar tanto dentro del frontend, como desde otro cliente, por lo que deben mantenerse actualizados en todo momento.

## Caché de fuentes de datos

Algunos de estos procesos son muy lentos, como el analizador de canciones o LastFM, y otros requieren de un gran número de peticiones, como Spotify o LastFM, por lo que el proceso de detallar al completo una canción o álbum puede afectar negativamente a la experiencia de usuario.

Para solventar este problema se ha implementado una caché de datos local que permita almacenar los datos estáticos de las canciones, álbumes y artistas en el propio navegador. Esta caché está implementada directamente en el frontend gracias a IndexedDB, una base de datos NOSQL presente en los estándares de W3C que está disponible en todos los navegadores modernos.

La caché local contiene tres tablas, Track, Album y Artist, que almacenan de forma conjunta todos los datos de cada una de las fuentes de datos. En

este caso, una canción está compuesta por un álbum y uno o más artistas, y un álbum está compuesto por uno o más artistas.

### Fachada de Datos

Una vez planteadas las distintas fuentes de datos y sus limitaciones, nos encontramos con que coordinar la obtención de estos datos puede ser una tarea muy compleja debido a los distintos estados de cada dato.

- Un dato puede estar incompleto
- Un dato puede estar almacenado localmente
- Una canción puede estar cacheada parcialmente (por ejemplo, puede estar cacheado el artista al que pertenece, pero no su álbum)
- Un dato contiene datos dinámicos

Para facilitar la obtención de datos se ha implementado una fachada que permite obtener todos los datos de manera eficiente.
Esta fachada permite obtener a partir de un ID o respuesta de Spotify, un objeto completo con todos los atributos de las distintas fuentes de datos. Los destalles de implementación de la fachada pueden encontrarse en el Anexo C (Diseño).

## 5.3. Aprendizaje Automático

Para las tareas de aprendizaje automático se han utilizado redes neuronales convolucionales 3.2.

### Clasificador de Géneros y Subgéneros

Para el clasificador de géneros se ha utilizado una red neuronal convolucional basada en la arquitectura EfficientNetB0 [25] de Google. Se ha utilizado esta red como base debido a su capacidad de generalización para este problema.
En este caso, la función de activación de la capa de salida es una función $softmax()$, que normaliza la salida para que solo se active una única neurona.

Esta red se ha entrenado con todos los géneros del Ludwig Dataset 5.1, pero se han agrupado los géneros rock, metal y punk en un único género

para reducir el número de etiquetas. Para poder diferenciar estos géneros se ha utilizado una pequeña red entrenada para diferenciar entre estos 3 géneros.

#### Transfer Learning

Durante el proceso de experimentación se ha detectado una mejora en la precisión del clasificador gracias a inicializar los pesos a partir de otra red basada en EfficientNetB0. Esta red inicial ha sido entrenada con el conjunto de datos GTZAN Extended 5.1.

Una vez entrenada la red al completo con GTZAN Extended, se ha reemplazado la capa de salida de 10 neuronas por una capa de 9 neuronas, y se han vuelto a entrenar todas las capas con los nuevos datos.

#### Clasificador de Subgéneros

La clasificación de subgéneros es un problema de clasificación multietiqueta, ya que una canción puede tener uno o más subgéneros.

En este caso, debido a la gran cantidad de subgéneros que puede tener cada género, se han entrenado 11 clasificadores multietiqueta, por cada uno de los géneros principales del Ludwig Dataset 5.1.
La función de activación de la capa de salida [17] es la función *sigmoide*(), en la que cada neurona puede tomar un valor entre 0 y 1 para indicar si la canción se corresponde a una etiqueta o no.

### Clasificador de Estados de Ánimo

La clasificación de los estados de ánimo es un problema de clasificación multietiqueta, en el que el una canción puede tener varios estados de ánimo.

En este caso, la clasificación de los estados de ánimo se ha realizado mediante un ensemble de 7 CNNs binarias de tipo OVA 3.4. Cada modelo que forma el ensemble se especializa en detectar si una canción tiene un estado de ánimo en específico o no.
Por último, se han juntado todos los modelos en una única red neuronal para crear el ensemble, concatenando todas las salidas binarias en un único tensor de salida.

[17] La salida de la red tiene tantas neuronas como subgéneros.

### Datos

Para el entrenamiento se ha utilizado undersampling para balancear el número de datos por etiquetas.

El clasificador de géneros ha sido entrenado con 5000 canciones por etiqueta.
Cada clasificador de subgéneros ha sido entrenado con 600 canciones por cada subgénero.
Cada clasificador de estados de ánimo ha sido entrenado con conjunto de datos en el que la mitad de los datos contienen el estado de ánimo que se desea identificar y la otra mitad no.

Estos conjuntos balanceados se han dividido en dos subconjuntos aleatorios. Por un lado, tenemos el conjunto de entrenamiento que contiene el 70 % de las canciones que vamos a utilizar, este subconjunto será utilizado para ajustar el modelo. Por otro lado, tenemos un pequeño conjunto de validación con el 30 % de las canciones restantes, que utilizaremos para evaluar la precisión final de los modelos.

Para aumentar la variedad de datos, mejorar la generalización de la red, y evitar entradas muy grandes, se han dividido todas las canciones, de 30 segundos de duración, en segmentos de 3 segundos. Para cada uno de los segmentos se han extraído 32 MFCCs [18] 3.1, que serán utilizados como entrada de la red.

### Votación

Por último, para mejorar la confianza de la predicción de cada clasificador se realiza una votación para decidir la clase final de la canción. Esta votación se realiza a partir de inferir un conjunto de fragmentos de 3s y se selecciona la etiqueta más popular de los resultados como la etiqueta final de la canción.

## 5.4. Sistemas de Recomendación

### Vecindad como métrica de similitud entre canciones

Para crear un sistema de recomendación basado en contenidos se ha decidido utilizar un sistema de vecindad. Este sistema permite encontrar las canciones más cercanas o similares a una canción específica.

[18] La entrada a la red es una matriz de $32 \times 130$

Para generar las instancias que van a ser comparadas, se ha decidido extraer el máximo número de datos de una canción mediante Librosa. Para resumir los resultados vectoriales o matriciales, como los MFCCs o los espectrogramas de mel 3.1, se han reemplazado por su media y varianza. Debido a las limitaciones del sistema, se han eliminado todos los datos cuyas operaciones de extracción tardaban más de 0.1s respeto a la media de operaciones[19]. Y para reducir aún más el tamaño del vecindario, realizó un estudio de la correlación de los datos mediante Knime 4.1.

Los datos finales utilizados para crear el vecindario son:

1. 32 medias y varianzas de los MFCCs 3.1
2. Tempo. El tempo permite conocer como de rápido va la música.
3. Media y varianza del tempo obtenidas de ventanas de tamaño *hop_length*.
4. 12 medias y varianzas del cronograma. El cronograma permite conocer los tonos más importantes del espectrograma[20].
5. Media y varianza de los centroides del espectrograma
6. Media y varianza del ancho de banda del espectrograma. Se divide el espectrograma en ventanas de tamaño *hop_length*, el ancho de banda es la porción de frecuencias más representativa en la que se representa la señal, la frecuencia central del ancho es el centroide.
7. Media y varianza del contraste del espectrograma. Comparación entre la energía media (db) de cada ventana del espectrograma y la energía más elevada.

Una vez creado el vecindario a partir del conjunto de datos 5.1, se normalizan los datos y se entrena un ball tree 3.5 para poder consultar los ids de Spotify de las canciones más similares de forma eficiente.

## Sistema de Recomendación Colaborativo basado en Playlists

Se ha implementado un prototipo de sistema de recomendación colaborativo basado en los contenidos de 200.000 playlists. Estas playlists de Spotify

[19] Algunas de las características eliminadas son los tempogramas, centroides tonales, centroides espectrales o descomposición de espectrogramas

[20] Espectrograma en frecuencias de mel

han sido recopiladas y anonimizadas por la promia compañía en [23]. Para la creación y evaluación de las recomendaciones se ha utilizado la biblioteca Scikit Suprise [11].
Este sistema de recomendación está basado en una puntuación a cada uno de los elementos de la playlist con una nota del 0 al 5, calculada a partir de la popularidad de la playlist, frecuencia de actualización, % de canciones del mismo artista o álbum, etc.
Las recomendaciones iban a integrarse con la plataforma como un sistema para completar la playlist de un usuario a partir de canciones ya existentes. Se ha descartado la idea por el elevado coste de añadir nuevos datos a una matriz dispersa, ya que es necesario volver a ajustar la matriz dispersa para cada una de las peticiones de recomendación.
Por otro lado se han intentado obtener las recomendaciones más cercanas a a otra mediante la implementación de un algoritmo de vecindad de Surprise, pero debido a que esta implementación no usa matrices dispersas, el consumo de memoria necesario para obtener las canciones más cercanas es demasiado alto, por lo que la implementación de esta característica no es viable a nivel de recursos.

## 5.5. Arquitectura Contenedores

Se ha dividido la arquitectura en cuatro servicios, **frontend**, **servidor principal o servidor NextJS**, **base de datos** y **servidor recuperación de información musical**[21].

Estos cuatro servicios se han agrupado en tres contenedores Docker4.2 distintos interconectados, como muestra la figura 1.1.
El servicio puede levantarse en una sola máquina gracias a Docker Compose, se recomienda consultar el manual de instalación y despliegue en los anexos.

### Contenedor NextJS

El contenedor NextJS contiene el frontend y el servidor principal, ya que ambos se ejecutan sobre un mismo servidor de NextJS.

El servicio de frontend se encarga del renderizado parcial en el servidor de las distintas páginas y componentes web, así como de la distribución de las páginas y ficheros estáticos.

El servicio de servidor NextJS es una pequeña API REST que se encarga de gestionar operaciones con datos sensibles, como la autenticación del

[21] También referido como Backend MIR o Backend Ludwig

usuario, eliminación de la cuenta o la gestión de tokenes JWT 4.1. Por otro lado, para reducir el acoplamiento del sistema, se ha decidido que este servicio sea el único servicio con acceso a la base de datos.
Esta API Rest ha sido configurada para permitir la política de seguridad CORS[22], por lo que el frontend y el servidor pueden desplegarse por separado.

### Contenedor DynamoDB

El contenedor DynamoDB contiene una instancia de DynamoDB Local 4.1, la única base de datos del sistema[23]. Este contenedor no está pensado para ser desplegado a producción, ya que una de las principales ventajas de DynamoDB es que es una base de datos administrada que se despliega en un cluster para mejorar su rendimiento. Para más información se recomienda consultar el apartado de despliegue 5.7.

### Contenedor Ludwig MIR

El contenedor Ludwig MIR contiene una API de python que se encarga del preprocesamiento y análisis de canciones en cualquier formato de audio[24].
La política de seguridad de este contenedor obliga a todas las peticiones a portar un token de seguridad en la cabecera `Authentication`. Este token es un código secreto estático que debe ser compartido entre todos los servicios que desean interactuar con el contenedor. En este caso, únicamente el contenedor NextJS tiene acceso a este código, por lo que actúa como intermediario.
Debido al elevado coste de descargar, preprocesar y analizar canciones, se ha planteado el uso de una pequeña caché que almacene los resultados en DynamoDB. La lógica de esta caché está gestionada por el servidor de NextJS.

## 5.6. Pruebas e Integración Continua

### Pruebas Unitarias

Se han realizado pruebas unitarias sobre los componentes de la capa de presentación, como los clientes REST, y clases y funciones auxiliares. Debido

[22]Intercambio de Recursos de Origen Cruzado, el frontend puede acceder a recursos del servidor sin necesidad de compartir dominio.
[23]Sin tener en cuenta la caché del frontend con DexieDB
[24]Gracias a FFMPEG

a la dificultad de renderizar componentes fuera de un navegador, ya que no todas las funcionalidades se pueden ejecutar fuera de un navegador sin el uso de mocks[25], se ha descartado realizar pruebas unitarias exhaustivas en la capa de la Vista.
Estas pruebas han sido realizadas mediante la biblioteca JEST 4.1

### Pruebas Punto a Punto

Se han realizado pruebas automáticas y de sistema a la interfaz web, así como a los distintos endpoints de las APIS mediante Cypress 4.1. Estas pruebas se realizan directamente sobre la web desplegada. Los endpoints han sido implementados siguiendo la metodología TDD 4.3.

### Integración Continua

Se ha programado mediante Github Actions 4.2 la ejecución de las distintas pruebas tras cada operación de push y merge al repositorio. Estas acciones permiten generar reportes sobre el estado del proyecto, y detectar los fallos en el momento en el que se actualiza el proyecto.

## 5.7. Despliegue Continuo

### Google Cloud Run

Se han desplegado en contenedores de 4.2 los servicios NextJS Backend y Ludwig Backend. Estos contenedores se generan y despliegan automáticamente[26] tras cada push que modifique los backends en el proyecto.

El contenedor NextJS se ha alojado en máquinas con apenas 256MB de RAM y un núcleo, ya que este contenedor apenas realiza operaciones con un elevado coste de recursos.
Este contenedor tiene libertad para escalar automáticamente.

El contenedor Ludwig Backend se ha desplegado en máquinas con 2GB de RAM y un núcleo. Estos contenedores están pensados para procesar hasta 25 canciones al mismo tiempo, por lo que necesitan un mayor número de recursos para inferir con todas la redes neuronales.

[25] Clases o funciones esqueleto que contienen todas las operaciones de la clase real, no todas las bibliotecas incluyen mocks.

[26] Gracias a Google Container Registry, esta operación se puede delegar a Github Actions

Para reducir el consumo de recursos, se han cuantizado 3.3 los modelos a UINT8, reduciendo el consumo de memoria un contenedor de 6.8GB a menos de 2GB de RAM. Los consumos de memoria son tan elevados debido a que trabajamos con bloques de canciones para acelerar la inferencia.

Para acelerar el proceso de inferencia de un gran grupo de canciones, el orquestador está configurado para levantar el máximo número de contenedores en vez de encolar las peticiones de inferencia en grupos de 25 canciones. Se ha configurado un límite de 350 contenedores simultáneos antes de que el sistema empiece a encolar las peticiones de inferencia.

### Vercel

El propio frontend de NextJS es transpilado, optimizado y desplegado por la plataforma Vercel 4.2. Vercel, a diferencia de Google Cloud Run, no admite contenedores personalizados, pero está diseñada para optimizar, desplegar y escalar proyectos de NextJS.
Se ha configurado Vercel para realizar un despliegue automáticamente por cada push y pull request que se realice en el repositorio del proyecto[27].

El servidor web de NextJS no se ha desplegado junto a su frontend por las limitaciones del plan gratuito de Vercel, ya que una petición al servidor no puede durar más de 10s, un tiempo muy bajo para que Ludwig pueda inferir los bloques de 25 canciones.

### Amazon Web Services

En este caso se ha utilizado el servicio DynamoDB 4.1 para la base de datos. Este servicio permite controlar y escalar la base de datos en al menos dos clusters, dividiendo los datos a patir del hash de los índices. Las tablas son gestionadas automáticamente gracias a Dynamoose, por lo que todas las actualizaciones se propagan a la base de datos de AWS.

[27] Al igual que GCP, se puede automatizar mediante Github Actions

# Trabajos relacionados

## 6.1. Similitud de Audio Mediante Redes Neuronales Siamesas

En [5] presentan una aproximación mucho más interesante a la hora de calcular la similitud entre dos canciones usando redes neuronales siamesas. Este sistema utiliza dos redes neuronales exactamente iguales[28] Fig.6.1, con la que se obtiene una salida de 128 elementos. Esta salida es comparada mediante una función de pérdida que permite obtener la diferencia entre cada una de las entradas de la red. Se ha descartado esta aproximación de similitud por varias razones.

**Entrenamiento:** Para poder entrenar una red siamesa es necesario agrupar manualmente los datos según su similitud. Cada grupo deben contener $N$ canciones similares y $N$ canciones diferentes.[29]
**Inferencia:** Para poder obtener la similitud de una canción es necesario comparar la salida de la red con todas las canciones mediante la función de comparación.

## 6.2. Servicios Similares

### Spotify

Desde el inicio del proyecto, Spotify [26] ha añadido algunas funcionalidades que se asemejan a las implementadas en este proyecto. Por un

[28] misma arquitectura y pesos
[29] N puede ser un número pequeño, de 4 o 5 canciones

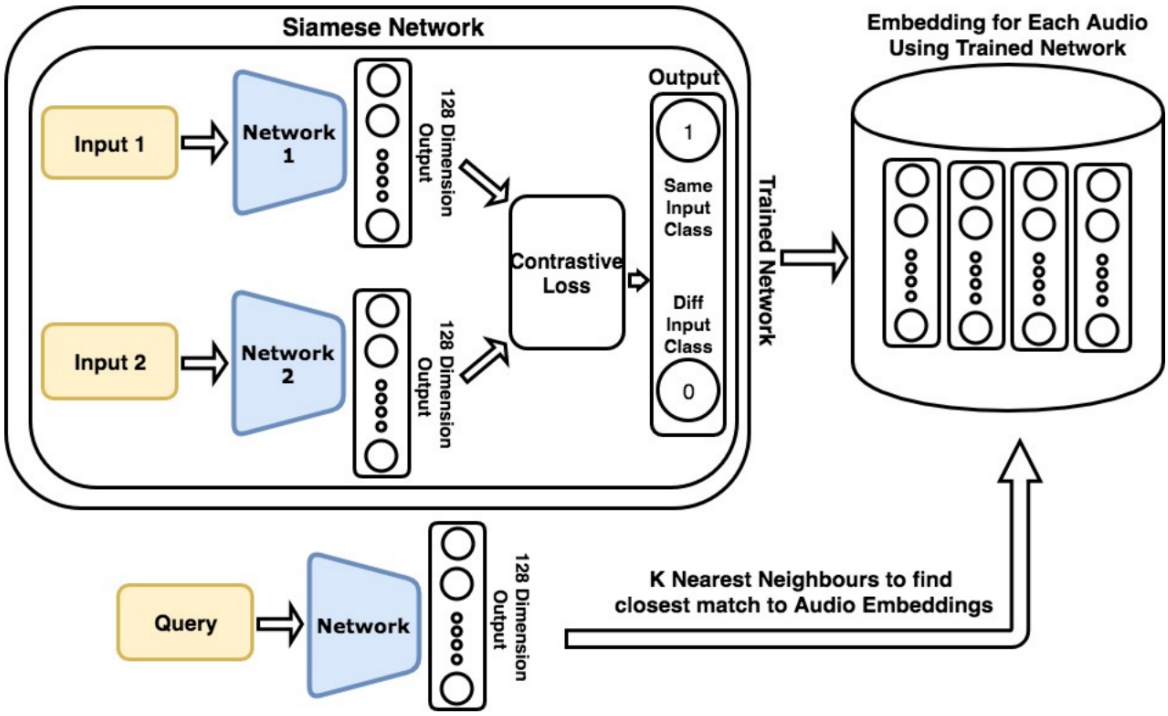



Figura 6.1: Sistema de Recomendación basado en RNS. Extraído de [5]

lado, Spotify ha añadido un sistema para rellenar playlists con canciones recomendadas, similar al sistema de recomendación de SpotMyFM. Por otro lado, Spotify crea periódicamente playlists personalizadas sobre un tema específico, como por el ejemplo **géneros**, **décadas**, **estados de ánimo** o **artistas**. Estas playlists están formadas por canciones de la biblioteca del usuario, así como recomendaciones.

## Rate Your Music

Rate Your Music [27] (RYM) es una plataforma colaborativa destinada a la clasificación y calificación de música. RYM no tiene integración con Spotify, pero gracias a su base de datos se incluyen características similares a SpotMyFM, como el poder etiquetar álbumes y añadir filtros a nivel de etiquetas, géneros o fechas. Por otro lado, RYM incluye un sistema de recomendación basado en las calificaciones del usuario. Este sistema de recomendación colaborativo, al igual que el que se ha planteado para SpotMyFM, no funciona a tiempo real.

Tabla 6.1: Comparativa entre Obscurify y SpotMyFM

| Funcionalidad | SpotMyFM | Obscurify |
|---|---|---|
| Alcance del Análisis | Biblioteca<br>Playlists<br>Canciones más escuchadas | Canciones más escuchadas |
| Géneros Favoritos | Si | Si |
| Valoración General de Popularidad | No | Si |
| Canciones/Artistas Menos Populares | Si (Métrica de Spotify) | Si (Métrica especial) |
| Canciones/Artistas Más Populares | Si | No |
| Artistas Favoritos | Si (Siempre) | Si (último mes) |
| Agrupamiento por Décadas | Si | Si |
| Recomendaciones | Si (Contenido) | Si (Colaborativo) |
| Estados de Ánimo | Si (análisis de la canción) | Parcial (Felicidad, Energía y Acústico) |
| Evolución de los géneros a lo largo del tiempo | Si | No |
| Filtros | Si | No |
| Creación de Playlists | Si (Cualquier Item) | Si (Solo recomendaciones) |
| Integración con LastFM | Si | No |
| Detalles de canciones / Artistas | Si | No |

## Obscurify

Obscurify [28] es una web de código abierto que permite consultar las estadísticas personales de Spotify de manera similar a SpotMyFM. La principal diferencia entre Obscurify y SpotMyFM es que Obscurify únicamente muestra las estadísticas de artistas y canciones más escuchados durante el **último mes**, **último medio año** y **desde el inicio**, mientras que SpotMyFM permite consultar las estadísticas de dichos intervalos, la biblioteca completa del usuario y playlists individuales.

Obscurify está centrada en el análisis de popularidad de una biblioteca, y realiza este análisis a partir de una métrica que tienen en cuenta la popularidad de una canción o artistas, así como su posición en el top personal del usuario. Obscurify, a diferencia de SpotMyFM, no está centrado en la gestión de la biblioteca.

La tabla 6.2 contiene una comparativa entre las funcionalidades de análisis de SpotMyFM y Obscurify.

## Organize Your Music

Organize Your Music [29] (OYM) es una web que permite crear playlists de Spotify a partir de una selección de canciones. En este caso, la plataforma

OYM tiene integración con la API de Spotify, y está centrada en la creación de playlists a partir de otras playlists d o la biblioteca del usuario, por lo que no incluye ningún tipo de detalles de las canciones. Esta plataforma clasifica las canciones a partir de sus géneros, duración, popularidad, décadas o duración, y permite seleccionar canciones individuaeles o grupos enteros. La principal diferencia entre el sistema de selección de SpotMyFM y OYM, es que OYM incluye un gráfico de dispersión de dos dimensiones con todas las canciones, en la que se puede seleccionar con un círculo las canciones que se encuentran en dicha gráfica. Los ejes del gráfico son personalizables, y el usuario puede escoger entre todas las características de las canciones, como la duración, popularidad, etc.

# Conclusiones y Líneas de Trabajo Futuras

## 7.1. Conclusiones

### Rendimiento de los clasificadores

La clasificación de música es un problema muy complejo, ya que cada artista tiene un estilo muy concreto, que puede estar influenciado por otros estilos y géneros, dificultado la tarea de clasificación de una pieza concreta. Por otro lado, las representación del audio, así como las técnicas para clasificarlo llevan varios años estancadas.

La OVA dedicada a la clasificación de estado de ánimo tiene una precisión media del 80 %, siendo esta una precisión muy similar a la que podemos encontrarnos en la literatura [30]. Por otro lado, el clasificador principal de géneros alcanza un 78 % de precisión en el conjunto de entrenamiento GTZAN Extended, superando a la precisión del ser humano, pero apenas alcanza el 59 % con el Ludwig Dataset, un conjunto de datos mucho más variado.

### Limitaciones de depender de APIs

Si bien el uso de APIs pública es el único método para disponer de todos los datos y recursos del servicio, el no disponer de una base de la base de datos completa ha causado un gran número de inconvenientes.

**Diseño de Datos**, tal y como se ha expuesto en el punto 5.2, ha sido

necesario unificar los datos y diseñar una caché local debido al elevado número de peticiones y heterogeneidad de las fuentes de datos.

**Desconocimiento de Datos**, debido a no tener acceso a todos los datos, no podemos disponer de conjuntos de datos más completos para los sistemas de recomendación, ya sea basada en contenido (etiquetas y similitud) o colaborativa.

**Estabilidad del Sistema**, debido a la elevada dependencia de tantas fuentes de datos ajenas al proyecto, si alguna de ellas falla, por ejemplo el inicio de sesión de Spotify, el frontend no se podrá utilizar.

## 7.2. Líneas de Trabajo Futuras

### Desarrollo Web

#### Ventanas Virtuales

Los navegadores webs no están diseñados para tener un gran número de elementos en el DOM, por lo que la simple tarea de mostrar un gran número de canciones, álbumes, playlists o artistas puede empeorar bastante el rendimiento, y con ello la experiencia de usuario.

Para evitar estos problemas de rendimiento se ha implementado un sistema de carga diferida (únicamente se muestran los elementos a medida que el usuarios se desplaza por la página) y de paginación, que limita el número de elementos por página para evitar sobrecargar al navegador.

Existe una solución mucho más elegante, la virtualización del HTML. Esta solución es muy utilizada en la web moderna por grandes plataformas como Twitter o Spotify. La virtualización consiste en añadir al DOM un número limitado de elementos (por ejemplo 20 tweets), y a medida que el usuario realiza scroll, desmontar del DOM los tweets antiguos y añadir los siguientes tweets en la parte inferior. Si bien esta técnica parece sencilla de implementar, aparece una pequeña limitación, y es mantener la barra de scroll en su posición correcta a medida que la ventana virtual avanza, así como cargar los elementos correctos al mover la barra de scroll a un punto intermedio.
La solución más sencilla consiste en utilizar elementos con altura fija, ya que así podemos añadir un espacio en blanco que rellene los elementos que se han desmontado, manteniendo la posición del la barra de scroll. Por ejemplo, si sabemos que hemos desmontado 100 elementos con una altura de 100px

cada uno, debemos añadir un espacio en blanco de 10000px para mantener la posición del scroll, y si el scroll está a 2000px del inicio, sabemos que estamos frente al elemento número 20 de la lista.

En el caso de SpotMyFM, este tipo de cálculos no es posible realizarlo de forma eficiente ya que cada una de las tarjetas tiene una altura diferente, y el número de tarjetas por fila depende del ancho de la pantalla. Se plantea realizar un estudio que permita conocer los cambios necesarios para poder implementar esta técnica en el proyecto.

### Server Side Log In

Se propone controlar el renderizado de una página dependiendo de si el usuario ha iniciado sesión o no en el servidor. Esta aproximación leerá la Cookie y comprobará si es correcta.
El principal inconveniente está relacionado con el inicio de sesión, ya que sería necesario realizar una segunda comprobación en el propio frontend para iniciar la sesión local, por lo que la única mejora se vería reflejada en el tamaño de la página y no en los tiempos de carga.

### Refactorización de Enums con Strings

En algunos puntos del frontend se han utilizado Enums o Records con strings para identificar selecciones (principalmente en los selectores y menús desplegables). Esto ha causado que algunos elementos no hayan podido ser internacionalizados correctamente debido a que los enums han sido declarados estáticamente fuera del alcance del hook[30] de traducción, y este hook únicamente funciona dentro de componentes ReactJS.

## Diseño de Datos

### Elastic Search

Se propone la creación de una base de datos con las instancias de los datos necesarios para el sistema de recomendación. Esta base de datos se debería ir expandiendo con nuevos contenidos a partir de las búsquedas de un usuario. Para realizar la operación de búsqueda de vecinos de manera eficiente, se propone el uso de Elastic Search[31], un motor de búsqueda avanzado para bases de datos que incluye un sistema de vecindad.

[30] Una función especifica de ReactJS que permite re-renderizar la interfaz de usuario para actualizar cambios de manera declarativa

DynamoDB tiene integración con Elastic mediante OpenSearch, un fork de Elastic Search de AWS.

### Expandir Dataset y Otras Representaciones

Si bien se han empleado un gran número de técnicas para mejorar el rendimiento del clasificador de géneros, se ha llegado a la conclusión de que los datos son el principal limitante.
Por un lado se ha planteado reemplazar los coeficientes cepstrales en las frecuencias de mel por espectrogramas en las frecuencias de mel. Este cambio de representación apenas mejora la confianza y tiene un elevado coste de memoria.
Se plantea el uso de un dataset mucho más grande, ya que la augmentación de datos apenas tiene impacto sobre el resultado como expone [32]. Se plantea el uso de Million Song Dataset [33] para intentar mejorar el rendimiento de los modelos.

## Ciencia de Datos

### Otras Técnicas de Extracción de Información Musical

Existen muchas otras aproximaciones para trabajar con música, como la separación de las pistas que forman una canción [34], extraer las letras de canciones a partir del audio o extraer los sentimientos a partir de la letra para acompañar al clasificador de géneros, como explora [30]. Se plantea el estudio e implementación de algunas de estas técnicas para expandir el funcionamiento de la plataforma.

# Bibliografía


[1] C. Mitcheltree and H. Koike, "White-box audio vst effect programming," 02 2021.

[2] V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning," 2016.

[3] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, "A survey of quantization methods for efficient neural network inference," 2021.

[4] P. Pawara, E. Okafor, M. Groefsema, S. He, L. R. Schomaker, and M. A. Wiering, "One-vs-one classification for deep neural networks," *Pattern Recognition*, vol. 108, p. 107528, 2020.

[5] P. Manocha, R. Badlani, A. Kumar, A. Shah, B. Elizalde, and B. Raj, "Content-based representations of audio using siamese neural networks," 2017.

[6] D. S. Lau and R. Ajoodha, "Music genre classification: A comparative study between deep learning and traditional machine learning approaches," in *Proceedings of Sixth International Congress on Information and Communication Technology* (X.-S. Yang, S. Sherratt, N. Dey, and A. Joshi, eds.), (Singapore), pp. 239–247, Springer Singapore, 2022.

[7] K. Rao and M. k e, *Speech Recognition Using Articulatory and Excitation Source Features*. 01 2017.

[8] O. R. developers, "Onnx runtime." https://onnxruntime.ai/, 2021. Version: x.y.z.

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

[10] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[11] N. Hug, "Surprise: A python library for recommender systems," *Journal of Open Source Software*, vol. 5, no. 52, p. 2174, 2020.

[12] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[13] Wikipedia contributors, "Test-driven development — Wikipedia, the free encyclopedia," 2022. [Online; accessed 1-July-2022].

[14] G. Tzanetakis and P. Cook, "Musical genre classification of audio signals," *IEEE Transactions on Speech and Audio Processing*, vol. 10, no. 5, pp. 293–302, 2002.

[15] M. Dong, "Convolutional neural network achieves human-level accuracy in music genre classification," 2018.

[16] B. L. Sturm, "The state of the art ten years after a state of the art: Future research in music information retrieval," *Journal of New Music Research*, vol. 43, pp. 147–172, apr 2014.

[17] "Gtzan extended wav | kaggle." https://www.kaggle.com/datasets/jorgeruizdev/gtzan-extended-wav/settings. (Accessed on 06/21/2022).

[18] "Ludwig music dataset (moods and subgenres) | kaggle." https://www.kaggle.com/datasets/jorgeruizdev/ludwig-music-dataset-moods-and-subgenres?select=mfccs. (Accessed on 06/21/2022).

[19] "Discogs: la base de datos y el mercado online de la música," 2022.

[20] A. Swartz, "Musicbrainz: A semantic web service.," *IEEE Intelligent Systems*, vol. 17, no. 1, pp. 76–77, 2002.

[21] H. S. J. U. Dmitry Bogdanov, Alastair Porter and S. Oramas, "The AcousticBrainz Genre Dataset: Multi-Source, Multi-Level, Multi-Label, and Large-Scale," in *International Society for Music Information Retrieval Conference*, pp. 360–367, 2019.

[22] "Documentation | spotify for developers." https://developer.spotify.com/documentation/. (Accessed on 07/02/2022).

[23] C.-W. Chen, P. Lamere, M. Schedl, and H. Zamani, "Recsys challenge 2018: Automatic music playlist continuation," in *Proceedings of the 12th ACM Conference on Recommender Systems*, RecSys '18, (New York, NY, USA), p. 527–528, Association for Computing Machinery, 2018.

[24] "Api docs | last.fm." https://www.last.fm/api. (Accessed on 07/02/2022).

[25] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," 2019.

[26] "Spotify." https://spotify.com/. (Accessed on 06/20/2022).

[27] "Welcome! - rate your music." https://rateyourmusic.com/. (Accessed on 06/20/2022).

[28] "Obscurify music." https://obscurifymusic.com/home. (Accessed on 06/20/2022).

[29] "Organize your music." http://organizeyourmusic.playlistmachinery.com/. (Accessed on 06/20/2022).

[30] C. Laurier, J. Grivolla, and P. Herrera, "Multimodal music mood classification using audio and lyrics," in *2008 Seventh International Conference on Machine Learning and Applications*, pp. 688–693, 2008.

[31] "Búsqueda gratuita y abierta: Los creadores de elasticsearch, elk y kibana | elastic." https://www.elastic.co/es/. (Accessed on 07/04/2022).

[32] R. L. Aguiar, Y. M. Costa, and C. N. Silla, "Exploring data augmentation to improve music genre classification with convnets," in *2018 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, 2018.

[33] T. Bertin-Mahieux, D. P. W. Ellis, B. Whitman, and P. Lamere, "The million song dataset.," in *ISMIR* (A. Klapuri and C. Leider, eds.), pp. 591–596, University of Miami, 2011.

[34] A. Défossez, N. Usunier, L. Bottou, and F. Bach, "Music source separation in the waveform domain," 2019.