RECOMENDACIÓN PERSONALIZADA DE CANCIONES

PERSONALIZED MUSIC RECOMMENDER

TRABAJO FIN DE GRADO
CURSO 2023-2024

AUTORAS
LEIRE JIMÉNEZ GONZÁLEZ
LAURA MARTÍNEZ TOMÁS

DIRECTORA
Mª BELÉN DÍAZ AGUDO

GRADO EN INGENIERÍA INFORMÁTICA
FACULTAD DE INFORMÁTICA
UNIVERSIDAD COMPLUTENSE DE MADRID

RECOMENDACIÓN PERSONALIZADA DE CANCIONES

PERSONALIZED MUSIC RECOMMENDER

TRABAJO DE FIN DE GRADO EN INGENIERÍA INFORMÁTICA

AUTORAS

LEIRE JIMÉNEZ GONZÁLEZ

LAURA MARTÍNEZ TOMÁS

DIRECTORA

Mª BELÉN DÍAZ AGUDO

CONVOCATORIA: JUNIO 2024

GRADO EN INGENIERÍA INFORMÁTICA

FACULTAD DE INFORMÁTICA

UNIVERSIDAD COMPLUTENSE DE MADRID

27 DE MAYO DE 2024

DEDICATORIA

A todos los que nos han apoyado en todo este proceso.

AGRADECIMIENTOS

Queremos agradecer a todas las personas que nos han rodeado y apoyado durante la elaboración de este proyecto. Este trabajo representa un camino lleno de aprendizaje y superación de desafíos.

A Belén por guiarnos y orientarnos en todas las etapas del proyecto.

A nuestros amigos y compañeros por ayudarnos y compartir con nosotras todos esos buenos y malos momentos, haciendo que el camino fuera más fácil.

A nuestras familias por el apoyo constante.

RESUMEN

Music4u, recomendación personalizada de canciones

Hoy en día la música es una de las representaciones artísticas más usadas y accesibles para todo el público. Es por ello por lo que cada día son más las personas que utilizan plataformas digitales que ofrecen contenido musical, como Spotify, Amazon Music o YouTube, entre otras.

La amplia variedad y el aumento progresivo de canciones en estas plataformas puede hacer que sea abrumador para el usuario. Los recomendadores suponen una buena solución a este problema, siendo de gran ayuda para clasificar y mostrar al usuario contenido relacionado con sus gustos y preferencias.

En este Trabajo de Fin de Grado hemos implementado un algoritmo de recomendación híbrido de canciones mediante la investigación y análisis de diferentes técnicas. En nuestro caso hemos optado por un recomendador que aproveche las ventajas de los recomendadores basados en contenido y de filtrado colaborativo atendiendo a los problemas que pueden surgir, como es el *cold-start* y *long tail,* y la confianza del usuario con la recomendación, mejorándola con la explicabilidad. Asimismo, se ha desarrollado una interfaz de usuario para mostrar visualmente los resultados del algoritmo de recomendación al usuario.

Palabras clave

Música, recomendador, Python, explicabilidad, playlist.

ABSTRACT

Music4u, personalized music recommender

Music is one of the most widely used and accessible forms of artistic expression today. As a result, more and more people are using digital platforms that offer music content, such as Spotify, Amazon Music, or YouTube, among others.

The wide variety and the progressive increase of songs on these platforms can make it overwhelming for users. Recommenders are a good solution to this problem, being of great help to classify and show users content related to their tastes and preferences.

In this Final Degree Project, we have implemented a hybrid song recommendation algorithm through the research and analysis of different techniques. In our case, we have opted for a recommender that takes advantage of the advantages of content-based recommenders and collaborative filtering, considering the problems that may arise, such as cold-start and long tail, and the user's trust in the recommendation, improving it with explainability. Likewise, a user interface has been developed to visually show the results of the recommendation algorithm to the user.

Keywords

Music, recommendation, Python, explainability, playlist

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

Capítulo 1 - Introducción

1.1. Motivación

La recomendación ha existido como un proceso natural entre las personas. Por ejemplo: tenemos dos amigos que van a un restaurante y uno de ellos está indeciso sobre el plato que pedir, el otro le intentará aconsejar en función a sus gustos y preferencias, con el fin de que la decisión final le satisfaga. Esta es una acción muy repetida en el día a día que, con la llegada de Internet se ha automatizado y resulta indispensable para solucionar el problema de la sobrecarga de información en el mundo digital.

Los sistemas de recomendación (Ricci et al. 2010) son herramientas que ofrecen sugerencias de artículos útiles para el usuario, como qué comprar, qué música escuchar o qué libro leer a continuación, etc. Estos sistemas se centran en un elemento o ítem y acorde a ello se adapta el diseño de estos, la interfaz y la técnica o técnicas de recomendación a usar para proporcionar sugerencias efectivas y útiles. Esta herramienta está diseñada para personas que no tienen suficiente experiencia o tiempo para evaluar la gran cantidad de opciones que se les presentan, sirviendo como ayuda para mejorar esta experiencia.

Es un gran desafío el poder ofrecer al usuario un catálogo musical completo en base a sus preferencias, y que a la vez consiga encontrar nuevos registros musicales para completar la experiencia del usuario, por eso surgió la idea de los sistemas de recomendación. Aunque esta idea ya existe y los sistemas de recomendación actuales son muy precisos, nos resulta muy interesante el poder llegar a replicar y conocer más en detalle el funcionamiento e implementación de estas herramientas (Spotify AB 2024).

Hoy en día son muchas plataformas como *Netflix*, *Instagram* y *Google*, las que tienen implementadas estos sistemas. Existen muchos informes como el de la consultora McKinsey (Criado González 2018) que afirma que en 2018 el 35% de las compras realizadas por los usuarios de Amazon provenían de las recomendaciones y hoy en día.

La gran cantidad de información que nos llega por segundo a través de Internet puede resultar abrumadora. En muchas ocasiones, puede suceder que la información recibida no sea precisa o simplemente no nos ofrezca la información necesaria que buscamos. Un ejemplo podría ser cuando realizamos una búsqueda en *Google*, los recomendadores actúan de modo que muestran al usuario los resultados más relevantes de acuerdo con su perfil, descartando aquellos que no resulten destacados en ese momento.

Hemos visto que los recomendadores sirven para muchos ámbitos, pero en este trabajo nos centramos en un dominio de recomendación específico: la música, que está presente en muchos ámbitos de la vida y son las plataformas digitales como *Spotify*, *Amazon Music* o *YouTube* las que nos ofrecen la mayoría de este contenido

A veces nos encontramos con que recibimos siempre el mismo tipo de información, y llevándolo al ámbito musical, al interactuar con el sistema (por ejemplo, YouTube) se nos muestran las mismas canciones sin conocer otras muchas que probablemente sean de nuestro agrado.

1.2. Objetivos

Para desarrollar este proyecto hemos planteado una serie de objetivos y tareas con el fin de implementar progresivamente un algoritmo de recomendación a partir de un conjunto de datos que contiene información de *Spotify*, y de realizar el desarrollo de la aplicación como soporte para mostrar el funcionamiento de nuestro recomendador con las distintas funcionalidades para el usuario final.

Hemos escogido recoger la información de *Spotify* porque es una de las plataformas musicales más grandes que existen y cuenta con millones de canciones (Spotify AB 2024), que nos pueden ser útiles para intentar lograr la máxima completitud de nuestro algoritmo de recomendación. Además, cuenta con la funcionalidad de recomendación de contenido, que nos sirve como referencia para nuestro trabajo.

Estos objetivos y tareas son:

1. **Objetivo 1**. Conocer tecnologías y trabajos similares para ayudar a definir la funcionalidad del sistema.
    a. ***Tarea 1.1***. Lectura de proyectos similares.
    b. ***Tarea 1.2***. Búsqueda de datasets o plataformas que publiquen sus datos.
    c. ***Tarea 1.3***. Diseño de la arquitectura software: estudiar patrones adecuados para la aplicación.
2. **Objetivo 2**. Desarrollar un algoritmo de recomendación.
    a. ***Tarea 2.1***. Estudiar los tipos de recomendadores para determinar el que mejor se adecue a las necesidades detectadas.
    b. ***Tarea 2.2***. Estudiar frameworks existentes y determinar si son aplicables en nuestro diseño.
    c. ***Tarea 2.3***. Conocer los datos proporcionados por el dataset.
    d. ***Tarea 2.4***. Implementar un recomendador sencillo y probarlo en un prototipo con pocos datos para comprobar su funcionamiento.
    e. ***Tarea 2.5***. Probar diferentes tipos de recomendadores en un prototipo con pocos datos.
    f. ***Tarea 2.6***. Escalar el recomendador a datos grandes y/o conexión con datos en tiempo real.
    g. ***Tarea 2.7***. Comprobar que los datos a utilizar y el recomendador funcionan correctamente.
3. **Objetivo 3**. Desarrollar una interfaz web de usuario.
    a. ***Tarea 3.1***. Diseñar los prototipos de la interfaz.
    b. ***Tarea 3.2***. Desarrollar la interfaz web.
    c. ***Tarea 3.3***. Implementar la lógica de la interfaz.
    d. ***Tarea 3.4***. Realizar pruebas de usabilidad para comprobar su correcto funcionamiento.
4. **Objetivo 4**. Sistema completo funcionando.
    a. ***Tarea 4.1***. Integrar las partes: Recomendador - Interfaz.
    b. ***Tarea 4.2***. Resolver la integración entre las partes.

5. **Objetivo 5**. Experimentar con usuarios reales para validar el diseño propuesto.
    a. ***Tarea 5.1***. Crear usuarios de prueba para probar el funcionamiento.
    b. ***Tarea 5.2***. Integrar en el sistema usuarios potenciales de la aplicación.
6. **Objetivo 6**. Escribir y revisar la memoria.
    a. ***Tarea 6.1***. Desarrollo secuencial de la memoria.
    b. ***Tarea 6.2***. Revisión de la completitud y cohesión de la memoria.

1.3. Plan de trabajo

Para establecer el plan de trabajo hemos tenido en cuenta los tiempos de investigación, desarrollo y entrega de cada una de las partes de nuestro proyecto. Durante todo el proceso hemos seguido un desarrollo ágil basado en la metodología Scrum (ver apartado 2.3) con entregas pequeñas en periodos cortos de tiempo (generalmente, dos semanas), adaptando las necesidades que pudieran surgir a estos objetivos de tiempo. También, hemos realizado reuniones periódicas para revisar los progresos, la calidad y poner en común el trabajo a realizar futuro. Para realizar el seguimiento de todo este proceso, hemos usado la herramienta Excel con las diferentes tareas a realizar en los *sprints* y los tiempos de entrega.

La investigación ha sido la primera tarea realizada, con el fin de familiarizarnos con los conceptos y conocer las diferentes herramientas disponibles para la posterior implementación de nuestro proyecto. Estas investigaciones se han realizado durante todo el proceso de desarrollo del proyecto con el fin de mejorar la consistencia y rendimiento del trabajo.

Después de realizar la fase de investigación y alcanzar los conocimientos necesarios, hemos continuado con la fase de desarrollo. En esta fase hemos implementado el algoritmo recomendador y la aplicación, y nos ha servido para poner en práctica todo lo estudiado anteriormente.

Durante la evolución del trabajo, hemos ido alternando las dos fases anteriormente mencionadas, investigación y desarrollo, con la tarea de escritura de la

documentación del proyecto. También, hemos ido corrigiendo los errores o cambios producidos en el transcurso del mismo para que la información aquí presente esté actualizada y conforme con el trabajo realizado.

1.4. Proyecto

Para la gestión y colaboración en el desarrollo de nuestro proyecto, utilizamos *GitHub*, una de las plataformas de control de versiones y alojamiento de código que hemos usado en diferentes asignaturas a lo largo de la carrera (ver enlace).

A continuación, enlazamos unos vídeos de los diferentes flujos de la aplicación:

- Registro de usuario (ver enlace).
- Perfil de usuario y cierre de sesión (ver enlace).
- Descubrir canciones (ver enlace)
- Crear playlist (ver enlace).
- Mis favoritos y mis playlists (ver enlace)

Capítulo 2 - Estado de la cuestión

Para este proyecto nos hemos centrado en el estudio de los sistemas de recomendación, una rama de la inteligencia artificial que ha cobrado gran relevancia en los últimos años y cómo integrarlo en una aplicación web, como hemos introducido en el capítulo anterior (sección 1.1). Por ello hemos estudiado los diferentes tipos de recomendadores para designar cual se adecua mejor a nuestro propósito. Además de investigar las distintas arquitecturas de software.

En este capítulo introducimos los sistemas de recomendación, que son herramientas que proporcionan sugerencias personalizadas a los usuarios (sección 2.1). Existen distintos tipos de recomendadores (sección 2.1.1), entre los que destacan los basados en contenido, los colaborativos y los híbridos. Cada uno de estos tipos tiene sus ventajas y desventajas, y su elección depende del sistema y múltiples factores (tabla 2.2).

Otro aspecto importante en los sistemas de recomendación es la explicabilidad, es decir, la capacidad de proporcionar a los usuarios razones comprensibles sobre por qué se les ha recomendado un determinado ítem (sección 2.1.2). La explicabilidad no solo aumenta la confianza del usuario en el sistema, sino que también puede mejorar su satisfacción y su compromiso con el sistema (tabla 2.3).

Sin embargo, los sistemas de recomendación también se enfrentan a varios desafíos. Entre ellos están los problemas de *cold-start* y *long tail*. El problema de *cold-start* (sección 2.1.3.1) se refiere a la dificultad de hacer recomendaciones para nuevos usuarios o artículos, mientras que el problema de *long tail* se refiere a la tendencia de los sistemas de recomendación a concentrarse en un pequeño número de ítems populares, ignorando la gran cantidad de ítems menos populares (sección 2.1.3.2).

Asimismo, en esta sección también se detallarán los fundamentos de los lenguajes usados (sección 2.4), la arquitectura software (sección 2.2) y la metodología (sección 2.3) seguida para el desarrollo del algoritmo de recomendación, así como los de la aplicación.

2.1. Fundamentos de los sistemas de recomendación.

La idea de los sistemas de recomendación (Ricci et al. 2010) surgió al observar que las personas a menudo confían en recomendaciones de otros para tomar decisiones diarias. Este enfoque inicial, conocido como filtrado colaborativo, es una de las técnicas más usadas en las recomendaciones y en el que se aprovecha los gustos de usuarios similares.

Los sistemas de recomendación se siguen estudiando en la actualidad ya que se trata de algo relativamente reciente, surgiendo como área de investigación en la década de 1990. El campo de investigación aborda diversas técnicas, explicabilidad de recomendaciones (como *XAI* o Inteligencia Artificial Explicable), interacción con los usuarios y algoritmos avanzados para mayor eficiencia y fiabilidad, así también incluye el uso de contenido generado para el usuario y la protección del usuario. Actualmente se celebran conferencias dedicadas al tema de los sistemas de recomendación y se ofrecen cursos para el aprendizaje del desarrollo de estos en instituciones educativas.

Asimismo, con el desarrollo de sitios web de comercio electrónico en los años 2000, surgió la necesidad de filtrar la amplia gama de opciones disponibles. Los sistemas de recomendación ayudan a superar la sobrecarga de información al dirigir al usuario hacia ítems nuevos y relevantes para su tarea actual. Por lo que desempeñan un papel crucial en servicios digitales populares como *Amazon*, *YouTube*, *Netflix*, *Tripadvisor* y muchos más, ya que supone un aumento en las ventas y beneficios para estos sí está bien diseñado el recomendador.

Simplificando la funcionalidad de los sistemas de recomendación, las recomendaciones personalizadas se ofrecen como listas clasificadas de ítems, que son evaluadas y clasificadas basándose en las preferencias del usuario. Las preferencias del usuario pueden ser proporcionadas explícitamente por el usuario o mediante interacciones, por ejemplo, con clasificación del ítem recomendado. Se ha de tener en cuenta que no necesariamente se trata de recomendaciones personalizadas, también existen recomendadores no personalizados.

2.1.1. Tipos de sistemas de recomendación.

Todo modelo de sistema de recomendación opera con dos tipos de datos, y dependiendo de cuál prevalezca, se clasifica como un tipo específico de sistema de recomendación. Un tipo de datos se refiere a las interacciones entre el usuario y el ítem, como es el ejemplo ratings o valoraciones de la recomendación o el historial de compra. El otro tipo corresponde a la información asociada a los usuarios e ítems del sistema, como en el caso de las películas, que es un ítem, un género sería una característica (Aggarwal 2016).

Si el sistema recomendador se basa principalmente en el primer tipo de datos, se clasifica como un sistema de *filtrado colaborativo* (sección 2.1.1.1), el cual es el modelo más comúnmente empleado en los sistemas recomendadores actuales. Este enfoque tiene en cuenta los ratings de todos los usuarios y cuanto más similares sean dos usuarios, más se recomendarán los ítems que el otro usuario ha valorado de manera positiva (Aggarwal 2016).

Por otro lado, si el sistema utiliza principalmente el segundo tipo de datos, se trata de una recomendación *basada en contenidos* (sección 2.1.1.2). En este caso, la recomendación se realiza por similitud de ítems con características asociadas parecidas a los ítems que le gustaron al usuario anteriormente, por ejemplo, si el historial de un usuario es que le gustaron bastante canciones del género pop, la canción a recomendar será pop también. Este tipo de recomendadores busca satisfacer de manera precisa las necesidades del usuario (Aggarwal 2016).

Además, existe la posibilidad de desarrollar *sistemas de recomendación híbridos* (sección 2.1.1.3), que combinan aspectos de distintos modelos de recomendadores. Esto se hace comúnmente para aprovechar las ventajas de un modelo y corregir las desventajas del otro (Ricci et al. 2010). Existen más modelos de recomendadores como los basados en conocimientos, demográficos, basados en grafos, etc. (Aggarwal 2016) (Caro Martínez 2022).

En la Tabla 2.1 se muestran las comparativas de cada tipo.

Enfoque	Descripción	Datos
Filtrado colaborativo	Proporciona recomendaciones basadas en un enfoque colaborativo aprovechando las ratings y acciones del propio usuario como de los demás usuarios del sistema.	ratings del usuario + ratings del resto de usuario
Basado en contenidos	Proporciona recomendaciones basadas en el contenido (atributos) que ha favorecido en ratings y recomendaciones anteriores.	ratings del usuario + atributos de los ítems

Tabla 2-1. Comparación de las características principales de los modelos básicos de sistemas de recomendación (Caro Martínez 2022)

2.1.1.1. Filtrado colaborativo.

El filtrado colaborativo es un modelo fundamental, como bien se trata en los libros de (Aggarwal 2016) (Ricci et al. 2010) (Jannach et al. 2011), en el campo de los sistemas de recomendación, utilizada para predecir ítems que puedan gustar al usuario y prever las preferencias de éste en función de la información acumulada sobre las interacciones y valoraciones o ratings de todos los usuarios del sistema. La premisa básica es que, si dos usuarios tienen historiales de interacciones o ratings similares, es probable que compartan preferencias y, por lo tanto, las recomendaciones de uno pueden ser aplicables al otro.

Esta técnica se basa en la idea de que las comunidades de usuarios comparten similitudes en sus comportamientos y preferencias. A través del análisis de patrones en los ratings o acciones de un grupo de usuario, el sistema puede predecir y sugerir ítems que podrían interesar al usuario actual. Esta información obtenida se ha vuelto esencial en la industria, como en sitios de venta en línea, donde se personaliza el contenido según las necesidades de un cliente específico para mejorar la experiencia del usuario, fomentar la compra de artículos adicionales y aumentar las ventas.

El filtrado colaborativo puede implementarse de distintas maneras ya que hay distintos enfoques para éste. Los métodos más utilizados, según (Aggarwal 2016), son los conocidos métodos basados en memoria y los métodos basados en modelos:

1. **Basados en memoria**. También conocidos como basados en vecinos. Se trata de obtener ratings usuario-ítem en función de sus vecinos. Los vecinos son aquellos usuarios que valoran parecidos entre sí o ítems que son valorados parecidos. Estos vecinos pueden ser acorde al estudio realizado por (Caro Martínez 2022):
    - **Basados en usuarios.** Determina la clasificación de un ítem *A* para un usuario *X* acorde a la clasificación dada por los vecinos del usuario *X* al ítem *A*.
    - **Basados en ítems**. Determina la clasificación de un ítem *A* para un usuario *X* acorde a la clasificación dada por el usuario *X* a los vecinos del ítem *A*.
2. **Basados en modelo.** Utilizan métodos de aprendizaje automático y minería de datos para modelos predictivos. Algunos métodos son *árboles de decisión*, *sistemas basados en reglas*, *Naive Bayes*, *modelos de variable latentes* o *factorización de matrices*, entre otros.

El método de factorización (Ko et al. 2022) de matrices permite comprender las preferencias de los usuarios a través de datos de evaluación, almacenados como vectores en una matriz. Esto facilita la identificación de factores latentes que expresan la información del usuario y sus preferencias. Una forma de usar este método es con la descomposición de valores singulares (*SVD*), que ayuda a predecir los datos a los clientes de manera eficiente, transformando usuarios y elementos en un espacio común de factores latentes. Otra forma es usando las matrices de utilidad, que se encargan de almacenar información sobre las interacciones entre usuarios e ítems, como las calificaciones dadas por los usuarios a los ítems. Para utilizar correctamente estas matrices se usan en modelos de filtrado colaborativo con métodos como las matrices de factorización o modelos basados en memoria (Ho, Le, and Vu 2023).

En resumen, el filtrado colaborativo se basa en la premisa de que la sabiduría colectiva de la comunidad de usuarios puede guiar de manera efectiva las recomendaciones personalizadas. Este enfoque ha evolucionado a lo largo del tiempo,

con desarrollos continuos y adaptaciones, como se evidencia en la investigación y la competencia en el campo de los sistemas de recomendación.

2.1.1.2. Basado en contenidos.

Los sistemas de recomendación basados en contenido, como también se trata en los libros de (Aggarwal 2016) (Ricci et al. 2010) (Jannach et al. 2011), son una categoría de algoritmos de recomendación que utilizan los atributos descriptivos de los ítems para proporcionar recomendaciones personalizadas a los usuarios. El término "contenido" en este contexto se refiere a las características que describen los ítems, como palabras claves, género, actores y otra información relevante.

En los métodos basados en contenido, las recomendaciones se generan analizando las características de los ítems y combinándolas con los ratings o historial del comportamiento del usuario, si hay. El historial de interacciones del usuario con los distintos ítems, como los ratings, se utilizan como datos de entrenamiento para crear un modelo de clasificación o regresión específico del usuario. Además, estos datos de entrenamiento también incluyen las características de los ítems que el usuario haya interactuado en el algún momento. Así se crea un perfil para cada usuario teniendo en cuenta estos ítems y se utiliza para predecir si al usuario las preferencias del usuario, incluso si el ítem no tiene ningún rating.

En cuanto a los aspectos positivos de este modelo de sistema de recomendación es que realiza recomendaciones para nuevos ítems, como una nueva canción que ha sido recién sacada, que pueden no tener suficientes ratings o no tiene valoraciones directamente, ya que este modelo aprovecha los atributos similares de otros ítems ya calificados por el usuario para hacer predicciones.

Otro aspecto positivo es la independencia de unos usuarios con otros del sistema, ya que se trata de un sistema basado en contenido, en el que solo se tiene en cuenta los ratings proporcionados por el usuario que se va a construir el perfil. Esto último hace contraste con el método de filtrado colaborativo, explicado en el apartado anterior. Otra ventaja que ofrecen los modelos de recomendación basados en contenidos es que es más fácil producir explicaciones de las recomendaciones por las características.

Por otro lado, este modelo tiene desventajas, como es el caso de realizar recomendaciones obvias, debido a basarse en palabras claves o atributos, llevando a una reducción en la diversidad de recomendaciones. Además de resultar menos efectivos para nuevos usuarios por la ausencia o escasos ratings o historial de interacción con los ítems del sistema. Por lo que de primeras resulta difícil proporcionar recomendaciones precisas y a esto se le denomina *cold-start*, un problema clásico de los sistemas de recomendación (Gope and Jain 2017) que describiremos en la sección 2.1.3.1. Estos son los motivos por los que este tipo de recomendadores se usan en sistemas híbridos, para aprovechar sus ventajas y reducir sus desventajas con otro modelo.

A este modelo se le puede especificar alguna especificación por parte del usuario. Sin embargo, esto a veces se le conoce como sistemas basados en conocimientos, lo que provoca que haya una línea muy fina entre estos dos modelos. Para calcular la similitud entre los elementos hay distintos métodos (Ko et al. 2022) como los modelos de vectores de espacio, *TF-IDF*, *Naive Bayes* o *SVM*, que se usa cuando se quiere hacer la similitud por lo parecido en texto, por ejemplo, cuando se parece un género del otro, es que paso como en rock y punk-rock, que son parecidos entre sí. Otros métodos son por vecinos próximos, como *K-NN* o *Clustering*.

Clustering (Ko et al. 2022) es un algoritmo utilizado para agrupar datos en categorías o *clusters*, con el fin de describir patrones y similitudes entre ellos. El método más común es el *clustering K-Means*, que asigna datos al *cluster* más cercano basado en la similitud y repite el proceso para determinar el centro del cluster. Este tipo de algoritmo se usa en filtrado colaborativo, pero también en basado en contenido.

En resumen, los sistemas basados en contenidos se enfrentan a desafíos como el análisis de contenido limitado y la sobre especialización. Para mejorar este modelo o se combina con otro modelo, conocido como sistemas de recomendación híbridos, o utilizando conocimientos comunes y específicos del dominio, es decir, pasando a ser sistemas de recomendación basados en conocimiento. Pero este modelo presenta ventajas que hace que sus recomendaciones sean precisas.

2.1.1.3. Sistemas híbridos.

Un sistema de recomendación híbrido (Aggarwal 2016) (Burke 2002) (Jannach et al. 2011) es un enfoque sofisticado que aprovecha las fortalezas de múltiples métodos de recomendación para superar las limitaciones asociadas con modelos individuales, las cuales se encuentran en la Tabla 2.2. Este tipo de recomendadores están diseñados para integrar fuentes de datos y aprovechar el poder algorítmico de varios sistemas de recomendación para obtener recomendaciones más sólidas y precisas.

Hay tres formas principales (Aggarwal 2016) de crear sistemas de recomendación híbridos: diseño en conjunto o *ensemble design*, diseño monolítico o *monolithic design* y sistemas mixto o *mixed systems*.

Enfoque	Ventajas	Desventajas
Filtrado colaborativo	A. Puede identificar clientes B. No es necesario el conocimiento del dominio C. La calidad mejora con el tiempo D. Retroalimentación implícita suficiente	H. Problema en nuevos usuarios I. Problema en incremento de nuevos artículos J. Problema de la "caja negra" K. La calidad de la recomendación depende de un conjunto de datos históricos. L. Problema de estabilidad frente a plasticidad
Basado en contenidos	B, C, D	H, K, L

Tabla 2-2. Comparación de las ventajas y desventajas principales de los modelos básicos de sistemas de recomendación (Burke 2002)

1. **Diseño en conjunto.** Estos sistemas utilizan múltiples algoritmos de recomendación, tratando cada uno como un componente separado. Las recomendaciones de estos sistemas se combinan luego en una salida única. Esta combinación puede ser secuencial, donde la salida de un recomendador se utiliza como entrada para otro, o paralela, donde los recomendadores funcionan de manera independiente y sus predicciones se combinan al final.

2. **Sistemas monolíticos**. Este tipo de sistemas incorporan varios tipos de datos en un solo algoritmo de recomendación. La distinción entre las diversas partes del algoritmo puede no ser clara, y los algoritmos de recomendación que ya existen pueden necesitar ser modificados para su uso dentro del enfoque general.
3. **Sistemas mixtos**. Estos utilizan múltiples algoritmos de recomendación, pero los ítems recomendados por los diversos sistemas se presentan juntos uno al lado del otro. Este enfoque se utiliza frecuentemente cuando la recomendación es una entidad compuesta, donde se pueden recomendar varios ítems como un conjunto relacionado.

Además, los distintos tipos de sistemas de recomendación híbridos pueden emplear diferentes técnicas para combinar los algoritmos que son los siguientes (Aggarwal 2016) (Burke 2002,) (Caro Martínez 2022):

1. **Ponderación** (*weighted*). Puntajes de varios sistemas de recomendación se combinan en un puntaje unificado mediante agregados ponderados. La ponderación puede ser heurística o basada en modelos estadísticos formales.
2. **Conmutación** (*switching*). El algoritmo cambia dinámicamente entre diferentes sistemas de recomendación según las necesidades actuales, adaptándose a situaciones cambiantes o perfiles de usuarios.
3. **Cascada** (*cascade*). Un sistema de recomendación perfecciona las recomendaciones dadas por otro, a menudo involucrando un proceso escalonado donde el segundo recomendador clasifica aún más las sugerencias del primero.
4. **Inclusión de características** (*feature augmentation*). La salida de un sistema de recomendación se utiliza como características de entrada para el siguiente, mejorando el rendimiento del sistema central sin modificarlo.
5. **Combinación de características** (*feature combination*). Se combinan características de diferentes fuentes de datos y se utilizan en el contexto de un solo sistema de recomendación, creando un enfoque monolítico.

6. **Metanivel** (*meta-level*). El modelo generado por un sistema de recomendación sirve como entrada para otro, permitiendo un procesamiento más eficiente de representaciones densas de intereses del usuario.
7. **Mezcla** (*mixed*). Las recomendaciones de varios motores se presentan simultáneamente, especialmente relevantes en dominios de ítems complejos y a menudo utilizados en conjunto con sistemas de recomendación basados en conocimientos.

En resumen, un sistema de recomendación híbrido busca optimizar la calidad de las recomendaciones al combinar estratégicamente varios enfoques de recomendación, ya sea a través de métodos en conjunto, diseños monolíticos o mixtos. La elección del método de hibridación depende de los requisitos específicos y las características del problema de recomendación en cuestión.

2.1.2. Explicabilidad

Los sistemas de recomendación (Ricci et al. 2010) (Jannach et al. 2011) desempeñan un papel crucial en mejorar la experiencia del usuario al proporcionar sugerencias personalizadas, ya sea para películas, música u otros productos como ya hemos visto en los anteriores apartados. El crecimiento de las webs, desde de las de comercio electrónico hasta las de entretenimiento, servicios, contenido y sociales como es *Amazon*, *Twitch*, *Netflix* o *Tinder*, entre otras, y el aumento de datos han llevado a que los sistemas de recomendación estén siendo utilizados por miles de usuarios. Esto puede hacer que haya una discrepancia entre la recomendación del sistema y las necesidades de información del usuario. Las explicaciones pueden cerrar esta brecha justificando las recomendaciones de una manera que el usuario pueda entender. Por lo que ha surgido la investigación en explicaciones, especialmente con el desarrollo de nuevos algoritmos, particularmente para el filtrado colaborativo. Esto requiere técnicas para generar automáticamente explicaciones satisfactorias.

La recomendación explicable implica algoritmos de recomendación personalizados que proporcionan explicaciones junto con recomendaciones para mejorar la transparencia, persuasión y satisfacción del usuario y otros criterios que se

muestran en la Tabla 2.3. Los modelos de recomendación explicables pueden ser intrínsecos al modelo (mecanismo de decisión transparente) o agnósticos al modelo (mecanismo de decisión de caja negra con explicaciones generadas después de la decisión).

Criterio	**Definición**
Transparencia	Explica cómo funciona el sistema
Verificabilidad	Permite al usuario corregir al sistema
Confianza	Aumenta la confianza del usuario con el sistema
Eficacia	Ayuda al usuario a tomar buenas decisiones
Persuasión	Convence al usuario de probar cosas o comprar
Eficiencia	Ayuda al usuario a tomar decisiones más rápido
Satisfacción	Aumenta la facilidad del uso o disfrute del sistema

Tabla 2-3. Criterios de la explicabilidad y su definición (Tintarev and Masthoff 2007) (Ricci et al. 2010)

El alcance de la recomendación explicable se extiende más allá del desarrollo de modelos transparentes para incluir métodos efectivos de entrega de recomendaciones y explicaciones a los usuarios o diseñadores de sistemas. Si bien la transparencia (Ricci et al. 2010) es un aspecto esencial, no es el único objetivo explicativo. Los diferentes algoritmos explicativos se adaptan a diversas preferencias de los usuarios y a la eficiencia del sistema. Por ejemplo, las explicaciones de sistemas basados en contenido pueden ser más satisfactorias para los usuarios, aunque las explicaciones de sistemas de filtrado colaborativo sean más eficientes en ciertos contextos.

Los diferentes algoritmos explicativos se pueden dividir en tres niveles acorde cuán fácil es de generar la explicación para un tipo de usuario. Estos niveles son usuario individual (Nivel 1), contextualización (Nivel 2) y autorrealización (Nivel 3) que fueron propuestos en (Ricci, Rokach, and Shapira 2022). Las explicaciones basadas en contenido son más adecuadas para las explicaciones de usuarios individuales, mientras que las explicaciones de filtrado colaborativo son más adecuadas para la contextualización con otros usuarios y las explicaciones de modelos híbridos son más adecuadas para la autorrealización.

1. **Usuario individual**. A este nivel, las explicaciones se centran en proporcionar información sobre las preferencias y comportamientos individuales del usuario. Este nivel es adecuado para los sistemas de recomendación basados en contenido. En ese tipo de sistemas de recomendación, las explicaciones se basan en las propiedades de los elementos y cómo coinciden con el comportamiento o las preferencias pasadas del usuario. Por ejemplo, si un usuario ha calificado varias películas de acción de manera positiva, el sistema podría recomendar una nueva película de acción basada en características, como es el género de la película, o actores similares.
2. **Contextualización**. A este nivel, las explicaciones contextualizan las preferencias del usuario dentro de un contexto más amplio, como su comunidad o espacio de preferencia. Este nivel es el ideal para las explicaciones de filtrado colaborativo, que pueden centrarse en cómo las preferencias de un usuario se comparan con las de usuarios o grupos similares dentro del sistema.
3. **Autorrealización**. A nivel de autorrealización, las explicaciones apoyan los objetivos centrados en el usuario que fomentan el autodescubrimiento y la exploración. Las explicaciones híbridas pueden combinar elementos de enfoques de filtrado colaborativo y basados en contenido para ayudar a los usuarios a alcanzar sus objetivos de autorrealización, como descubrir nuevos intereses o perspectivas.

Hay muchos enfoques populares de recomendación explicables, métodos basados en modelos, donde las recomendaciones y las explicaciones provienen de un modelo explicable. Los métodos de recomendación explicables basados en modelos están estrechamente relacionados con el aprendizaje automático para los sistemas de recomendación. Algunos de los modelos propuestos por (Zhang and Chen 2020) son:

1. **Modelos de factorización**. El uso de factorización puede ser con matrices y máquinas de factorización para una recomendación explicable. Estos modelos tienen como objetivo abordar el desafío de las dimensiones latentes en la factorización de matrices alineándose con características explícitas extraídas de las reseñas de los usuarios.
2. **Modelado de temas**. Utilizan reseñas textuales en el comercio electrónico que se emplea ampliamente para recomendaciones explicables.
3. **Modelos basados en grafos**. Los modelos basados en grafos se utilizan para recomendaciones explicables, particularmente en configuraciones de redes sociales. Estos métodos demuestran la efectividad de los enfoques basados en gráficos para mejorar la explicabilidad de las recomendaciones.
4. ***Deep Learning***. Estos modelos se utilizan para tareas como la predicción de calificaciones y la recomendación de los N mejores, ofreciendo explicaciones a través de varios enfoques, incluyendo la distinción de palabras importantes de las reseñas, la generación de explicaciones en lenguaje natural y la selección de reseñas de usuarios relevantes.
5. **Minería de datos**. Las técnicas de minería de datos, particularmente la minería de reglas de asociación, juegan un papel crucial en los sistemas de recomendación explicables. Los ejemplos incluyen el aprovechamiento de las reglas de asociación para la recomendación de páginas web, los sistemas de recomendación de tiendas, el cálculo de similitud entre elementos y las recomendaciones basadas en tareas.
6. **Modelos agnósticos y *post hoc***. Los enfoques agnósticos al modelo y *post hoc* para la recomendación explicable implican la generación de explicaciones

después de las recomendaciones de un modelo de caja negra como es el filtrado colaborativo.

Como conclusión, la explicabilidad de los sistemas de recomendación abarca diversos estilos de explicaciones para los distintos sistemas. La elección del estilo de explicación debe alinearse con los objetivos del sistema y la satisfacción del usuario, garantizando una comprensión integral de las recomendaciones proporcionadas. Equilibrar la transparencia, la eficiencia y las preferencias del usuario es crucial para lograr explicaciones efectivas en los sistemas de recomendación.

2.1.3. Problemas

En los sistemas de recomendación existen dos problemas fundamentales que representan desafíos significativos para la eficacia de estas tecnologías: el problema de *cold-start (*sección 2.1.3.1) y el problema de *long tail* (sección 2.1.3.2).

En esta sección, se explorará en estos dos problemas, discutiendo sus causas, implicaciones en los sistemas de recomendación y sus posibles soluciones que se han propuesto por los autores clásicos como (Ricci et al. 2010), (Aggarwal 2016) y (Jannach et al. 2011).

2.1.3.1. Problemas de Cold-Start

El problema de *cold-start* (Ricci et al. 2010) (Aggarwal 2016) (Jannach et al. 2011) (Gope and Jain 2017) es una desafía común en los sistemas de recomendación, donde el sistema tiene dificultades para hacer recomendaciones precisas debido a la falta de información inicial sobre usuario o ítems. Este problema se pronuncia cuando un nuevo usuario se une al sistema o se añade un nuevo ítem al sistema.

En aplicaciones que hacen uso de sistemas de recomendación, las matrices de calificación, que se utilizan para predecir las preferencias del usuario, como ya hemos visto en las secciones 2.1.1.1 y 2.1.1.2, a menudo están dispersas. Esto significa que los usuarios suelen proporcionar ratings sólo para una pequeña fracción de los ítems disponibles. Esta falta de datos dificulta predicciones precisas.

Para abordar este problema, un enfoque es utilizar información adicional sobre los usuarios, como sus datos demográficos o intereses, para ayudar a clasificar a los usuarios y encontrar usuarios similares o ítems que les puedan interesar. Sin embargo, este enfoque se aleja de un sistema puramente colaborativo y plantea un nuevo modelo híbrido combinado con un sistema basado en conocimiento.

También se utilizan mecanismos de *conmutación* (Jannach et al. 2011) para manejar el problema de *cold-start*. Estos mecanismos implican usar un recomendador cuando hay menos datos disponibles y cambiar a otro recomendador cuando hay más datos disponibles, es decir, pasar a un modelo híbrido (sección 2.1.1.3), que ya hemos hablado anteriormente.

Otro mecanismo es la recomendación grupal (Ricci et al. 2010), que cuando un usuario es nuevo en el sistema, se proporcionan recomendaciones que mantendrían contento a todo grupo de usuarios existentes. Gradualmente, a medida que el sistema aprende sobre los gustos del nuevo usuario, el peso asociado al nuevo usuario aumenta, y los pesos asociados a usuarios existentes cuyos gustos difieren del nuevo usuario disminuyen.

Hay más técnicas para solucionar el problema como recopilar información faltante, ratings por defecto y, como hemos dicho, enfoques híbridos.

En resumen, el problema de *cold-start* es un desafío significativo en los sistemas de recomendación, y se han propuesto diversas estrategias para abordarlo. Estas estrategias buscan recopilar información faltante, ya sea de manera explícita o implícita, y utilizar esta información para hacer recomendaciones precisas y relevantes. También buscan reducir el sesgo, garantizar la adaptabilidad y mantener la diversidad en las recomendaciones (Gope and Jain 2017).

2.1.3.2. Problemas de Long Tail

En muchos escenarios del mundo real, la distribución de ratings o interacciones con ítems sigue una distribución de *Long Tail* (Ricci et al. 2010) (Aggarwal 2016) (Park and Tuzhilin 2008). Esto significa que un pequeño número de ítems son muy populares y tienen

muchos ratings, mientras que un gran número de ítems son menos populares y tienen pocos ratings.

El problema surge al intentar hacer recomendaciones basadas en estos ítems menos populares. Debido a que tienen menos ratings, es más difícil hacer predicciones precisas sobre si a un usuario le gustará o no. Esto puede llevar a una disminución en la calidad de las recomendaciones.

Un enfoque para abordar este problema es utilizar métodos de filtrado colaborativo, que hacen recomendaciones basadas en el comportamiento de usuario similares (ver sección 2.1.1.1). Sin embargo, estos métodos pueden verse influenciados por los ratings de los ítems populares y pueden ignorar los ítems que reciben muy pocos ratings.

Otro enfoque es construir modelos (Park and Tuzhilin 2008) individuales de estimación de ratings para cada ítem. Sin embargo, este método tiende a aumentar las tasas de error para los ítems de baja clasificación en la cola del conjunto de ítems, lo que lleva nuevamente al problema de recomendación *Long Tail*. Para resolver este problema, una estrategia es el clustering o agrupamiento de ítems para que los modelos predictivos aprendan utilizando más datos, disminuyendo así las tasas de error para los ítems menos populares. Este método, conocido como *Total Clustering* (TC) o agrupamiento total, ha demostrado superar al método *Each Item* (cada ítem), especialmente para ítems en la cola de la distribución. Sin embargo, el método de TC puede ser computacionalmente costoso y puede no escalar bien a problemas de recomendación grandes.

Una solución alternativa es dividir el conjunto de ítems en la cabeza y la cola y realizar el agrupamiento solo en la cola. Este enfoque puede producir tasas de error más pequeñas que TC en algunos casos, al tiempo que logra resultados de rendimiento razonables.

En conclusión, el problema de *Long Tail* es un desafío significativo en los sistemas de recomendación, pero se pueden emplear diversas estrategias para abordarlo y mejorar la calidad de las recomendaciones para ítems menos populares. Estas

estrategias implican equilibrar la necesidad de recomendaciones precisas con las demandas computacionales de los métodos utilizados.

En las secciones subsiguientes de este capítulo, abordaremos los aspectos restantes del trabajo. En la sección 2.2, profundizaremos en la arquitectura del software. Posteriormente, en la sección 2.3, discutiremos las metodologías empleadas para el desarrollo de la aplicación web. Finalmente, en la sección 2.4, exploramos las tecnologías web que se utilizarán en el proyecto.

2.2. Arquitectura software

Para poner en contexto el concepto de arquitectura del software, nos tenemos que remontar a la década de los 70 donde aún no se contaba con una etapa para el diseño del sistema, y fue a partir de entonces cuando se empieza a distinguir entre pequeños o grandes programas. Llegando ya a la década de los 90, se empieza a construir el concepto de Arquitectura del Software en la Universidad de *Carnegie-Mellon* y el Instituto de Ingeniería del Software, que es el principio que conocemos actualmente.

La arquitectura software es el diseño y la especificación de alto nivel de un sistema software, considerando aspectos estructurales, funcionales y relacionados con el rendimiento para garantizar la eficacia y el éxito del sistema. (Garlan and Shaw 1994). Por lo tanto, la arquitectura de software es fundamental para el buen funcionamiento y resultado del proyecto ya que nos ayuda a planificar el desarrollo y nos permite elegir las herramientas adecuadas para llevarlo a cabo.

Utilizamos los patrones de arquitectura (Castro, 2012) como soluciones generales para problemas frecuentes en el diseño y desarrollo de aplicaciones. Algunos de los patrones más comunes son:

1. **Modelo Vista Controlador** (*MVC*). Dividen la aplicación en tres componentes (modelo, vista y controlador). Separa los datos y la lógica de su representación y del módulo de gestión.

2. **Capas**. Divide la aplicación en capas lógicas. Esto permite descomponer las tareas en grupos de subtareas en la que las capas inferiores proporcionan servicios a las capas superiores.
3. **Cliente Servidor**. Consta de dos partes (un servidor y múltiples clientes). El servidor proporciona servicios a los clientes y los clientes solicitan servicios a dicho servidor.
4. **Microservicios**. Consta de varias aplicaciones pequeñas e independientes que uniéndose forman el programa principal.

Estos son algunos de los patrones más usados, pero existen muchos más. Dependiendo de los requisitos de cada proyecto, se usarán unos u otros, incluso se pueden combinar (Jiménez Torres, Tello Borja, and Ríos Patiño 2014).

Para el desarrollo de nuestro proyecto hemos decidido utilizar el patrón Modelo Vista Controlador (MDN contributors 2023), que es un patrón en el diseño de software comúnmente utilizado para implementar interfaces de usuario, datos y lógica de control, y se centra en separar la lógica de la aplicación con la visualización de esta misma (Fernández Romero and Díaz González 2012).

El concepto de *MVC* fue introducido por Trygve Reenskaug en los años 70 (Reenskaug 2003) y se propuso como forma de desarrollo de interfaces de usuario en aplicaciones de escritorio. Actualmente, este concepto sigue vigente y su implementación existe en muchos tipos de sistemas y lenguajes.

En esta arquitectura (Voorhees 2020) se realiza una separación de la aplicación en tres componentes de diseño:

1. **Modelo**. Responsable de gestionar los datos, implementa la lógica para crear, leer, actualizar y eliminar los datos de la aplicación.
2. **Vista**. Proporciona una interfaz para las interacciones del usuario.
3. **Controlador**. Responsable de la lógica de dominio. Básicamente el controlador comunica los componentes de vista y modelo.

Para clarificar, el flujo de control de este patrón queda reflejado en la Figura 2.1.

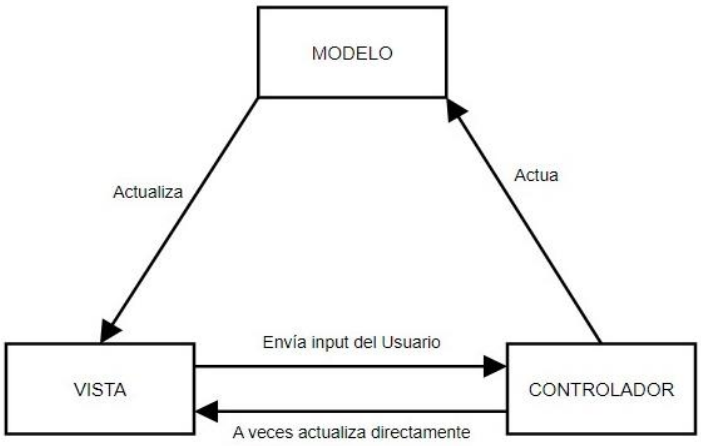

Figura 2-1. Diagrama de MVC inspirado en (MDN contributors 2023)

Las ventajas (Voorhees 2020) que ofrece este modelo es mantener la interfaz de usuario separada de la gestión de datos, lo que provoca que cualquier cambio en alguna de estas no afecte a la otra y así, también, permite mejor mantenimiento.

2.3. Metodología

La evolución de los desarrollos software (Esteban Gabriel, 2015) y el aumento de la complejidad de las tareas hacen que a finales de los 60 surja la necesidad de controlar los avances de los proyectos, estructurar de manera clara los equipos de trabajo y crear fases diferenciables para poder realizar verificaciones intermedias. Así como, la necesidad de documentar estos procesos y mayores estándares. (Zumba Gamboa and León Arreaga, 2018)

Actualmente, y después de todos los modelos propuestos en el tiempo, podemos definir una metodología de desarrollo software como un enfoque estructurado para el desarrollo de software que incluye modelos de sistemas, notaciones, reglas y guías de procesos (Zumba Gamboa and León Arreaga, 2018).

Las metodologías de desarrollo de software nos permiten controlar, estructurar y planificar todo el proceso de desarrollo del software para lograr que sea eficiente y

efectivo en todas sus fases, y permitir una correcta comunicación entre los equipos desarrolladores.

La elección de la metodología es clave para el éxito del desarrollo y no existe una metodología correcta, dependiendo de los requisitos y los objetivos podremos aplicar una u otras. Las metodologías se dividen en tradicionales y ágiles (Esteban Gabriel, 2015).

Las metodologías tradicionales se basan en realizar la planificación al inicio del proyecto, es decir, tener documentada la especificación de requisitos, modelado y plan de trabajo en la fase inicial. Este método plantea un enfoque lineal de las etapas del desarrollo, una etapa debe ser completada antes de pasar a la siguiente. Estas metodologías no se adaptan a cambios, por lo que su uso se debe realizar cuando los requisitos pueden predecirse y no dan lugar a variaciones.

Las metodologías ágiles (Canós et al. 2003) nacen como una solución a los problemas derivados de las metodologías tradicionales, se basan en la flexibilidad, y pueden ser modificadas para ajustarse a los cambios que puedan ocurrir durante el desarrollo del software. El proyecto se divide en subproyectos más pequeños y cada uno se trata independiente y desarrolla características en un corto periodo (Navarro Cadavid et al.). 2013)

En la Tabla 2.4 podemos encontrar las diferencias más relevantes entre las metodologías ágiles y las tradicionales.

Metodologías Ágiles	Metodologías Tradicionales
Basadas en heurísticas provenientes de prácticas de producción de código	Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo
Especialmente preparados para cambios durante el proyecto	Cierta resistencia a los cambios
Impuestas internamente (por el equipo)	Impuestas externamente
Proceso menos controlado, con pocos principios	Proceso mucho más controlado, con numerosas políticas/normas
No existe contrato tradicional o al menos es bastante flexible	Existe un contrato prefijado
El cliente es parte del equipo de desarrollo	El cliente interactúa con el equipo de desarrollo mediante reuniones
Grupos pequeños (<10 integrantes) y trabajando en el mismo sitio	Grupos grandes y posiblemente distribuidos
Pocos artefactos	Más artefactos
Pocos roles	Más roles
Menos énfasis en la arquitectura del software	La arquitectura del software es esencial y se expresa mediante modelos

Tabla 2-4. Diferencias entre metodología ágiles y no ágiles (Canós, Letelier, and Penadés 2003)

En este proyecto, nos vamos a centrar en las metodologías ágiles, en concreto en una de las más representativas: *Scrum* (Schwaber and Sutherland 2020). *Scrum* nace en el año 1986 gracias a un artículo publicado por Takeuchi y Nonaka en el que se define un nuevo enfoque en el desarrollo de productos, centrado en mejorar la agilidad y flexibilidad de los procesos. Está diseñado para lograr la colaboración eficaz de equipos. Dentro de un equipo se atribuyen diferentes roles: *Scrum Master* (encargado de comprobar que el modelo y la metodología funcionan), *Product Owner* (encargado de tomar las decisiones durante el proceso) y equipo de desarrollo (compuesto por un equipo pequeño de 5-9 personas que organizan y toman decisiones para conseguir el objetivo).

Esta metodología tiene como base la entrega del proyecto en pequeñas iteraciones llamadas "*Sprints*". Cada *Sprint* corresponde a un periodo de tiempo en el que se realiza una parte del desarrollo del producto, y su duración máxima es de un mes. Un Sprint se compone de los diferentes elementos (Schwaber and Sutherland 2020):

1. **Planificación del *Sprint***. Se establece el trabajo que se realizará en ese *Sprint* y es realizado por todo el equipo.

2. ***Daily Scrum***. Lo realiza el equipo de desarrollo y tiene el propósito de poner en común el trabajo realizado hasta el momento y el progreso a realizar hasta el objetivo.
3. **Revisión del *Sprint***. Se revisa el resultado del *Sprint* y se discute el progreso hacia el objetivo del proyecto.
4. **Retrospectiva del *Sprint***. Se analiza los aspectos como la comunicación, el proceso y las herramientas del *Sprint* terminado para determinar posibles mejoras para el siguiente.

Esta metodología se realiza en varias fases (Trigas Gallego 2012):

1. **Preparación del proyecto** (*sprint* 0). Es la fase inicial en la que se define el proyecto, el Backlog inicial, los entregables y se constituye el equipo.
2. **Planificación del Sprint**. En esta fase se realiza una reunión en la que participan todos los miembros del proyecto y tiene como finalidad seleccionar las funcionalidades sobre las que se va a trabajar en ese *Sprint*.
3. ***Desarrollo del Sprint***. en esta fase se trabaja para cumplir los objetivos propuestos en ese *Sprint*.

2.4. Tecnologías

En esta sección se detalla el conjunto de tecnologías que se han empleado en el desarrollo de este proyecto. La elección de estas tecnologías no ha sido aleatoria, sino que se ha basado en su relevancia en la industria actual, su robustez y la eficiencia que proporcionan en el desarrollo del software.

En la sección 2.4.1 se describen los lenguajes de programación utilizados que es la base de cualquier desarrollo de software.

En la sección 2.4.3 se detallan los frameworks empleados, es decir, los conjuntos de librerías que facilitan el desarrollo de software.

En la sección 2.4.4 se detallan las herramientas que se van a utilizar para facilitar el desarrollo del proyecto. Estas herramientas abarcan desde entornos de desarrollo

integrados (*IDEs*), como *Visual Studio Code*, hasta sistemas de control de versiones, como es *GitHub*.

Cada una de estas tecnologías ha jugado un papel importante en el desarrollo del proyecto, permitiendo un desarrollo más eficiente y robusto.

2.4.1. Lenguajes

Un lenguaje de programación es un sistema artificial que permite expresar algoritmos (Gabbrielli and Martini 2010). Hay distintos tipos de lenguajes de programación, algunos que propone (Ben-Ari, 1996) son los lenguajes orientados a objetos (*C++*, *Python*), funcionales (*Haskell*) y lógico (*Prolog*). Nosotras nos centraremos en *Python*, que es el lenguaje de programación de alto nivel y orientado a objetos utilizado para desarrollar el modelo y controlador del proyecto (sección 2.2).

Asimismo, se ha utilizado las tecnologías web, como *JavaScript* que es un lenguaje de programación de alto nivel responsable de definir el comportamiento de la web. *HTML* y *CSS* son tecnologías que complementan a *JavaScript* para definir, presentar y procesar el contenido y diseño de la aplicación (Sinha et al. 2020), respectivamente.

2.4.2. Librerías

Las librerías son conjuntos de archivos de código usados para el desarrollo del mismo, que facilitan la programación mediante funcionalidades comunes previamente resueltas. Permiten a los desarrolladores una mayor agilidad y reducción de costes en el proceso (Equipo de datos.gob.es).

Dado que *Python* es un lenguaje ampliamente usado, podemos encontrar multitud de librerías para simplificar el código de programación.

A continuación, explicaremos las librerías que hemos utilizado para el desarrollo de este proyecto.

1. ***Numpy***. Es una librería de Python que proporciona un objeto de matriz multidimensional y gran variedad de funciones matemáticas con operaciones rápidas. (NumPy Developers)
2. ***Pandas***. Es una librería que proporciona estructuras de datos y herramientas para el análisis de datos. ("pandas documentation — pandas 2.2.1 documentation")
3. ***Sklearn***. Es una librería de aprendizaje automático que proporciona herramientas de ajuste de modelos, preprocesamiento de datos, selección y evaluación de modelos, entre otras utilidades. (scikit-learn developers)
4. ***Scipy***. Es una librería que proporciona algoritmos de optimización e integración para modelar y resolver problemas científicos. (Virtanen et al. 2020), (SciPy Steering Council).

2.4.3. Frameworks

Los *Frameworks* son una técnica de reutilización orientada a objetos con el objetivo de facilitar el desarrollo y la implementación de una aplicación mediante la estructuración y normalización del código. (Johnson 1997)

Con el uso de *Frameworks* podemos programar un código de manera más eficaz y robusta, ya que ofrecen una estructura que puede ser modificada por el programador según sus necesidades. (Martínez Villalobos et al. 2010) (Pantoja and Pardo 2016)

Uno de los *Frameworks* más utilizados en la programación con *Python* es *Django*, es gratuito y de código abierto. Este marco de trabajo fue desarrollado entre 2003 y 2005, y proporciona un F*ramework* de alto nivel muy útil para la implementación de aplicaciones web debido a su rápido y fácil desarrollo.

Django utiliza el patrón de diseño *MTV* (*Model-Template-View*), que es una modificación del *MVC* anteriormente descrito, en el que separa los componentes de la aplicación en tres:

1. **Modelo**. Lógica de la aplicación.

2. **Datos**. Interfaz de usuario.

3. **Controlador**. Sirve como mediador entre el modelo y la vista.

Para poder usar este F*ramework* es necesario instalarlo y generar un proyecto *Django*, que ya proporciona ficheros automáticos para comenzar el desarrollo. (Vidal-Silva et al., 2021) (Forcier et al., 2009)

2.4.4. Herramientas

A continuación, vamos a describir las herramientas utilizadas para la elaboración de este proyecto.

Hemos usado una base de datos sirve para almacenar y estructurar el conjunto de datos con lo que vamos a trabajar. (Marqués Andrés, 2011). En nuestro caso, hemos optado por *Postgresql* (The PostgreSQL Global Development Group, 2024), un sistema de base de datos relacional orientada a objetos compatible con SQL y proporciona características avanzadas a este estándar (Gilbert Ginestà and Pérez Mora 2014).

Para editar el código hemos decidido usar *Visual Studio Code* ("Documentation for Visual Studio Code", 2024) que tiene soporte integrado para *JavaScript*, *TypeScript* y *Node.js*, y tiene extensiones para múltiples lenguajes, entre ellos, *Python*.

Y para el control de versiones, hemos decidido usar *Github* ("Documentación de GitHub" 2017), que es una plataforma que permite a los usuarios almacenar su código y compartirlo para que otros usuarios puedan revisarlo o colaborar en el proyecto.

Capítulo 3 - Implementación

3.1. Metodología

Nos hemos inspirado en la metodología *Scrum* para gestionar el desarrollo de la aplicación y del recomendador, llevando a cabo reuniones frecuentes tanto con la directora del TFG como entre nosotras para resolver dudas y revisar los avances. Nos hemos planificado siguiendo en cierta medida los principios de *Scrum*, una metodología ágil que se centra en la flexibilidad y la adaptabilidad antes los cambios durante el desarrollo del proyecto como detallamos en la sección 2.3.

En *Scrum*, el proyecto se divide en iteraciones llamadas *Sprints*, donde cada *Sprint* tiene una duración definida y se enfoca en la entrega de un conjunto de funcionalidades, como hemos visto en el capítulo anterior en la sección 2.3. En nuestro caso, las reuniones con la directora de duración parecida eran la fecha límite para ciertas tareas y así hemos podido avanzar en el trabajo de forma adecuada y constante. Además, estas reuniones nos han ayudado a revisar y mejorar el trabajo.

Las iteraciones de nuestro proyecto en cuanto al desarrollo del recomendador ha sido un número total de 4 iteraciones (sección 3.4) de una duración cada una de un mes y medio, siendo las dos primeras más simples debido a que en los primeros 3 meses se centró más en la investigación de sistemas de recomendación y de probar con los distintos métodos.

Esta aproximación nos ha permitido gestionar de manera eficiente nuestro proyecto, adaptándonos a los contratiempos que han surgido y maximizando la calidad del proyecto.

3.2. Descripción del conjunto de datos

Los conjuntos de datos empleados para la implementación del recomendador han variado conforme se desarrollaba. Para la última iteración del recomendador hemos usado un conjunto de datos referente a la información de las canciones (sección 3.2.1) (Joshi, Parolkar, and Das 2023), ajustado a los requerimientos específicos de

nuestra aplicación (sección 3.2.1). Y, por otro lado, tenemos un conjunto de datos que se refiere a los ratings de los usuarios, este dataset (sección 3.2.2), ha sido generado de forma sintética a través de un algoritmo, que ha consistido en la generación aleatoria de valoraciones de 610 usuarios, valoradas en 0 o 1.

3.2.1. Dataset canciones

Para la versión final del proyecto se ha utilizado un conjunto de datos o dataset, que contiene 10000 canciones con su correspondiente información. En la tabla 3.1 se muestra la información que contiene cada canción, representada en forma de columna con sus respectivos valores.

Columna	**Descripción**	**Valores**
SongId	Identificador único de la canción	[0, 10000]
Artist_name	Nombre del artista	Cadena de caracteres
Track_name	Título de la canción	Cadena de caracteres
Track_id	Identificador único de la canción en Spotify	Cadena de caracteres
Popularity	Cuán popular es la canción. Un número alto indica que es más popular	[0, 100]
Year	Año de lanzamiento de la canción	Año
Género	Género de la canción	Cadena de caracteres
Danceability	Describe lo adecuada que es una pista para bailar basándose en una combinación de elementos musicales como el tempo, la estabilidad del ritmo, la fuerza del compás y la regularidad general. Un valor de 0.0 es el menos bailable y 1.0 el más bailable	[0.0, 1.0]

Energy	Medida que representa una medida perceptiva de intensidad y actividad	[0.0, 1.0]
Key	Tonalidad de la pista. Los números enteros se asignan a tonos utilizando la notación estándar Pitch Class. Por ejemplo, 0 = C, 1 = C♯/D♭, 2 = D, y así sucesivamente. Si no se detectó ninguna clave, el valor es -1	[-1, 10]
Loudness	Los valores de sonoridad se promedian en toda la pista y son útiles para comparar la sonoridad relativa de las pistas. La sonoridad es la cualidad de un sonido que es el principal correlato psicológico de la fuerza física (amplitud). Los valores suelen oscilar entre -60 y 0 dB	[-60, 0] dB
Mode	Indica la modalidad (mayor o menor) de una pista, el tipo de escala del que se deriva su contenido melódico. Mayor se representa con 1 y menor con 0.	0 o 1
Speechiness	Detecta la presencia de palabras habladas en una pista. Cuanto más exclusivamente hablada sea la grabación (por ejemplo, programa de entrevistas, audiolibro, poesía), más se acercará a 1.0 el valor del atributo. Los valores superiores a 0,66 describen pistas que probablemente estén compuestas en su totalidad por palabras habladas. Los valores entre 0.33 y 0.66 describen pistas que pueden contener tanto música como voz, ya sea en secciones o en capas, incluyendo casos como la música rap. Los valores inferiores a 0.33 representan probablemente música y otras pistas no habladas.	[0.00,1.00]
Acousticness	Medida de confianza de 0.0 a 1.0 de si la pista es acústica. 1,0 representa una confianza alta en que la pista es acústica	[0.0,1.0]
Instrumentalness	Predice si una pista no contiene voces. Los sonidos "ooh" y "aah" se consideran instrumentales en este contexto. Las pistas de rap o *spoken word* son claramente "vocales". Cuanto más se acerque el valor de instrumental a 1.0, mayor será la probabilidad de que la pista no contenga voces. Los valores superiores a 0.5 representan pistas instrumentales, pero la confianza es mayor a medida que el valor se acerca a 1.0	[0.0,1.0]
Liveness	Detecta la presencia de público en la grabación. Los valores de *liveness* más altos representan una mayor	[0.0, 1.0]

	probabilidad de que la pista se haya interpretado en directo. Un valor superior a 0.8 indica una gran probabilidad de que la pista se haya grabado en directo	
Valance	Medida de 0.0 a 1.0 que describe la positividad musical que transmite una pista. Las pistas con valencia alta suenan más positivas (por ejemplo, felices, alegres, eufóricas), mientras que las pistas con valencia baja suenan más negativas (por ejemplo, tristes, deprimidas, enfadadas)	[0.0, 1.0]
Tempo	En pulsaciones por minuto (BPM). En terminología musical, el tempo es la velocidad o el ritmo de una pieza determinada y se deriva directamente de la duración media del compás	Valor mayor o igual a 0
Duration_ms	Duración de la pista en milisegundos.	Valor mayor o igual a 0
Time_signature	Convención notacional para especificar cuántos tiempos hay en cada compás	[3,7]
Genres	Lista de géneros de una canción generados de forma aleatoria entre los 75 géneros totales del dataset	Lista de cadena de caracteres

Tabla 3-1. Explicación del dataset de las canciones del sistema (Joshi, Parolkar, and Das 2023)

3.2.2. Dataset rating

En el dataset que contiene los ratings de los usuarios para las diferentes canciones se especifica el valor que un usuario ha atribuido a una canción concreta como se ve en la tabla 3.2.

Columna	**Descripción**	**Valores**
UserId	Identificador único del usuario	[1, valor indefinido]
SongId	Identificador único de la canción	[0, 10000]

Rating	indica si al usuario le ha gustado la canción o no. 1 indica que le ha gustado la canción, 0 indica que no le ha gustado.	0 o 1
Timestamp	Fecha en la que el usuario dio el rating	Milisegundos

Tabla 3-2. Explicación del dataset de los ratings del sistema

3.3. Diseño de la interfaz de usuario

Los prototipos de interfaces son elementos muy importantes para crear un diseño conceptual de la página web y sirven de guía para el desarrollo del diseño de la aplicación.

Usando la herramienta *Figma*, hemos desarrollado una serie de prototipos interactivos adaptados a nuestros gustos y necesidades, para poder probar y mejorar el modelo y las funcionalidades, así como, reducir los errores de diseño.

Hemos diseñado una web simple y fácil de usar que cubra todas las necesidades que un usuario de la aplicación pueda requerir, esta interfaz contiene diferentes funcionalidades:

1. Crear cuenta de usuario (figuras 3.2.1 y 3.2.2).
2. Inicio de sesión (figura 3.1).
3. Recomendación por canciones que le gustan al usuario y atributos preferidos (Género, año, duración y popularidad) (figuras 3.8.1 y 3.8.2).
4. Crear *playlist* a partir de las recomendaciones generadas en función a las canciones incluidas en esa lista (figura 3.6).
5. Consultar canciones que han gustado al usuario ("Mis favoritos") (figura 3.4).
6. Consultar las *playlists* del usuario ("Mis playlists") (figura 3.5).
7. Consultar las canciones que contiene cada *playlist* (figura 3.6).
8. Gestionar *playlists* (crear, visualizar, eliminar).
9. Consultar el perfil.

Con estos bocetos se ha buscado definir de forma parcial todos los elementos necesarios para la implementación y son un reflejo de lo que queríamos implementar en el resultado final.

En la figura 3.1 hemos diseñado la ventana de inicio de sesión en la que el usuario deberá introducir el correo electrónico o nombre de usuario y la contraseña (existe la posibilidad de hacer visible este campo). En el caso de no tener una cuenta en la aplicación, se da la opción de registrarse en la parte inferior del cuadro.

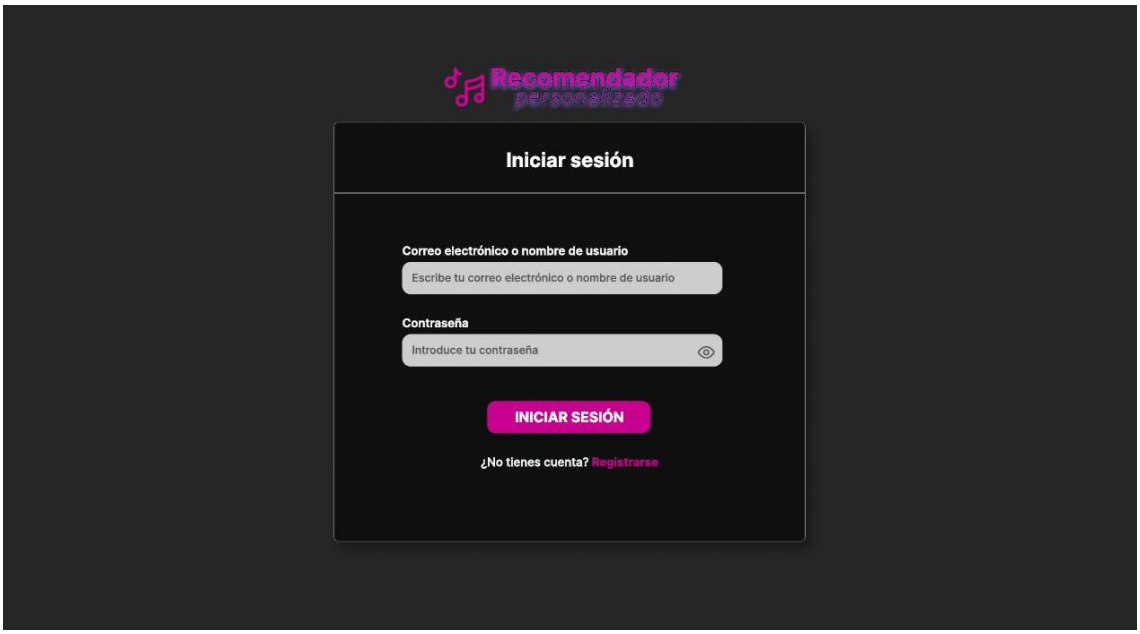

Figura 3-1. Diseño de inicio de sesión

En la Figura 3.2.1 podemos encontrar la ventana de crear cuenta (registro de usuario) en la que el usuario debe introducir el correo electrónico, nombre de usuario y contraseña (existe la posibilidad de hacer visible este campo). Una vez completados los campos anteriores, el usuario tendrá que marcar las casillas con las canciones que más se identifiquen con sus gustos para generar un perfil inicial.

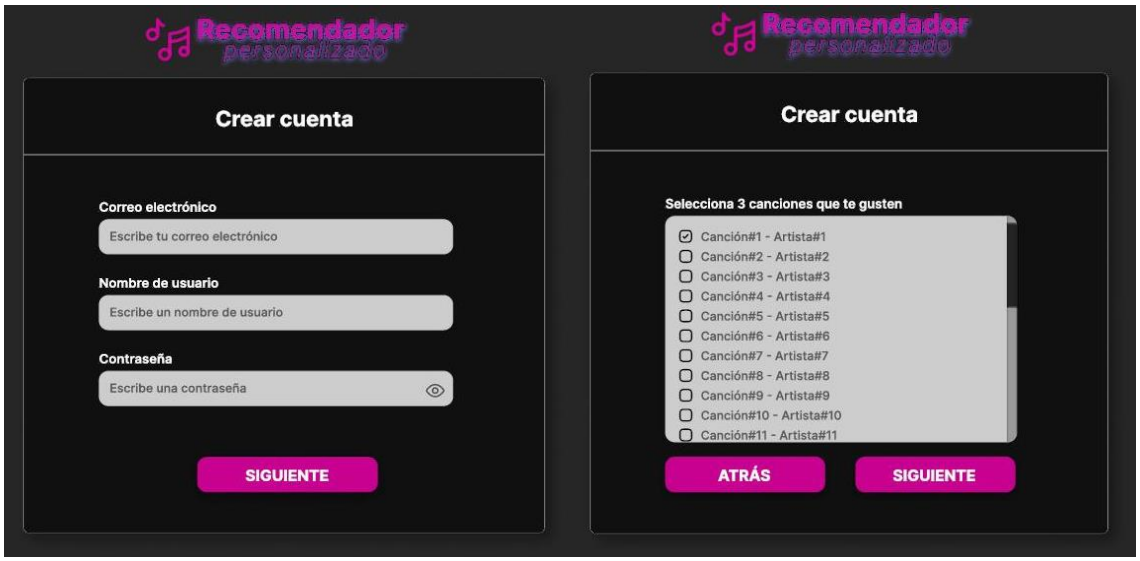

Figura 3-2-1. Diseño de crear cuenta

La Figura 3.2.2 es la continuación de la Figura 3.2.1, y como en el boceto anterior, el usuario deberá marcar las casillas de los artistas y géneros acordes a sus gustos y preferencias.

Figura 3-2-2. Diseño de crear cuenta

La Figura 3.3 muestra el boceto de la página principal de la aplicación web.

En esta pantalla hay un encabezado con el icono para el perfil del usuario, una barra de búsqueda para encontrar canciones a partir de la que recomendar y el icono de la casa para volver a la página de inicio.

En la parte central de la ventana encontramos los elementos para acceder a las funcionalidades principales de la aplicación: recomendación diaria (muestra canciones basadas en los gustos de cada usuario) y recomendación personalizada (genera una *playlist* a partir del perfil de cada usuario). Además, en la parte inferior, se muestra una barra de reproducción de canciones.

También, tenemos una barra lateral que contiene la lista de canciones favoritas y la lista de *playlists* del usuario.

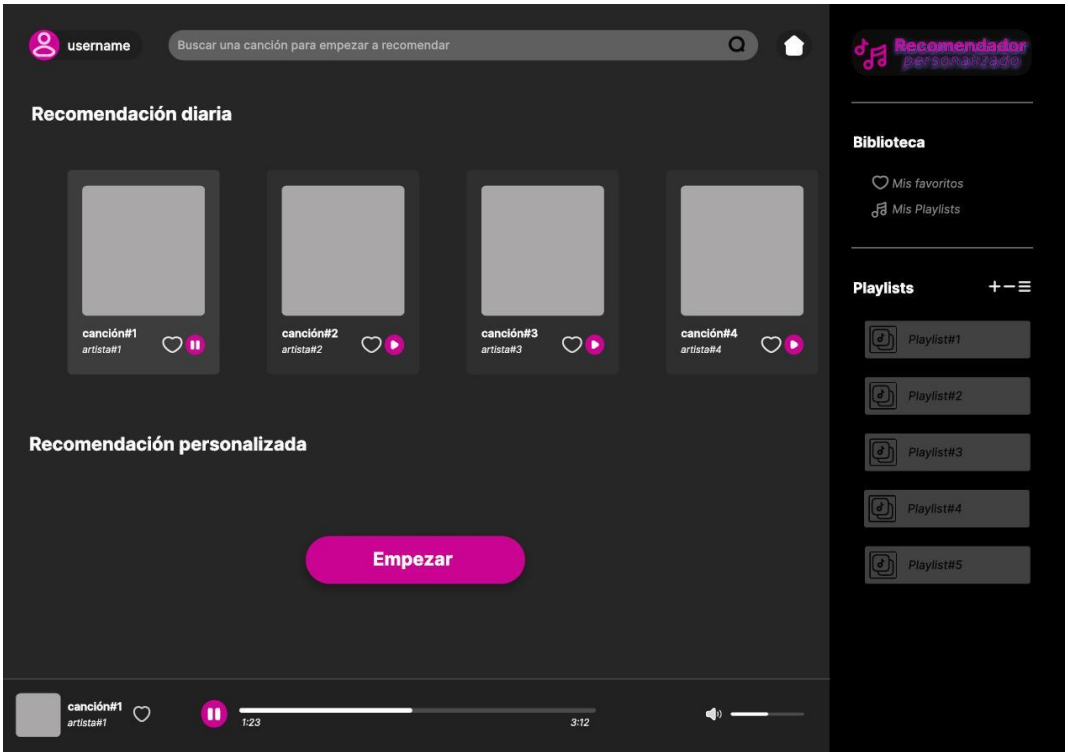

Figura 3-3. Diseño de inicio de la aplicación web

En la Figura 3.4 vemos la ventana de "Mis favoritos" que muestra la lista con la información principal (nombre, artista y duración) de todas las canciones que le han gustado al usuario. Tenemos la posibilidad de reproducir la canción y quitarla de la lista.

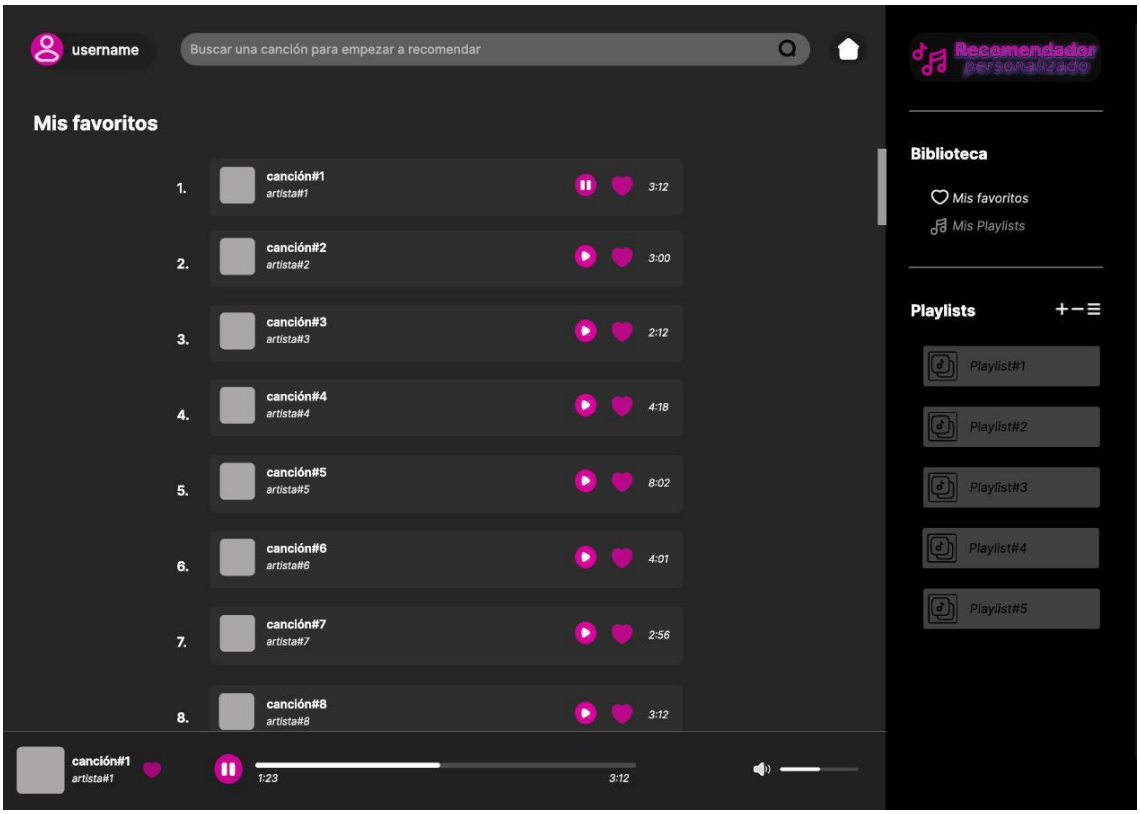

Figura 3-4. Diseño de canciones favoritas

En la Figura 3.5 vemos la ventana de "Mis playlists" que muestra la lista con la información principal (nombre y duración) de todas las *playlists* del usuario. Tenemos la posibilidad de reproducir la *playlist* y quitarla de la lista.

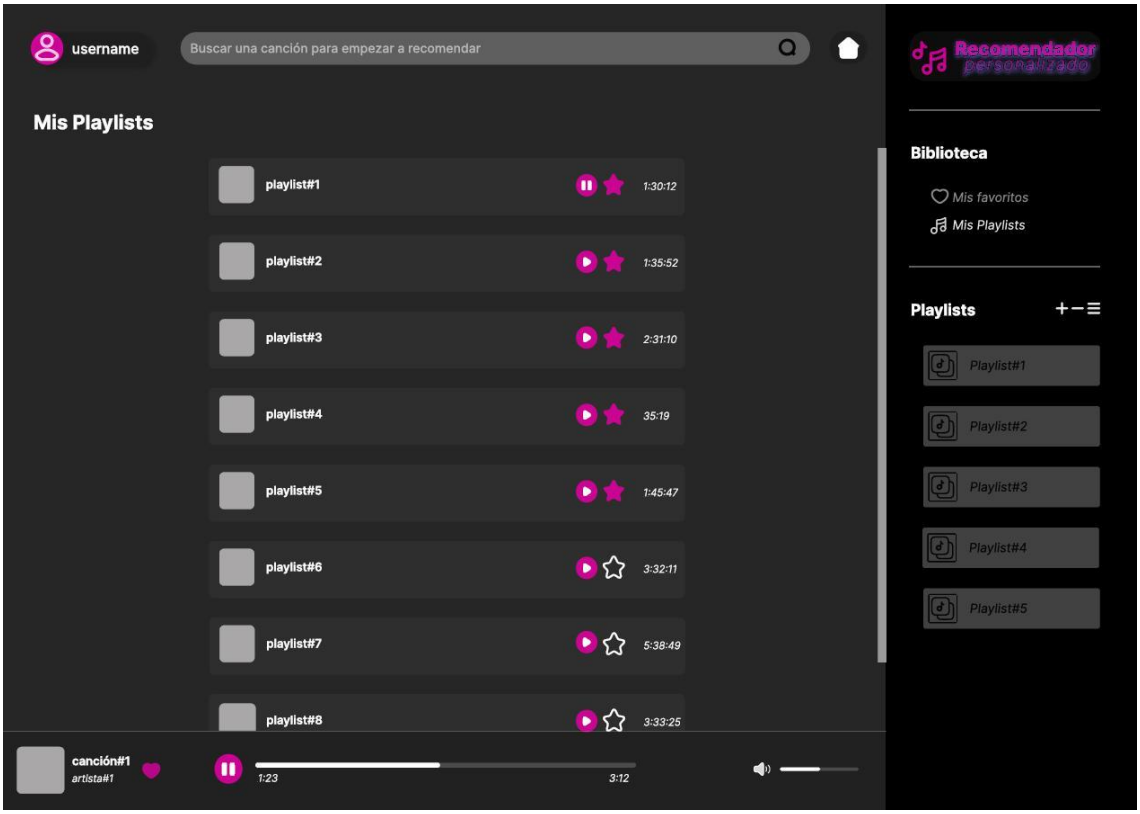

Figura 3-5. Diseño de playlist creadas

En la Figura 3.6 encontramos la primera funcionalidad principal en la que muestra la lista de canciones de una *playlist* en concreto. Además, contamos con la opción de recomendar otra *playlist* a partir de las canciones de esta *playlist*.

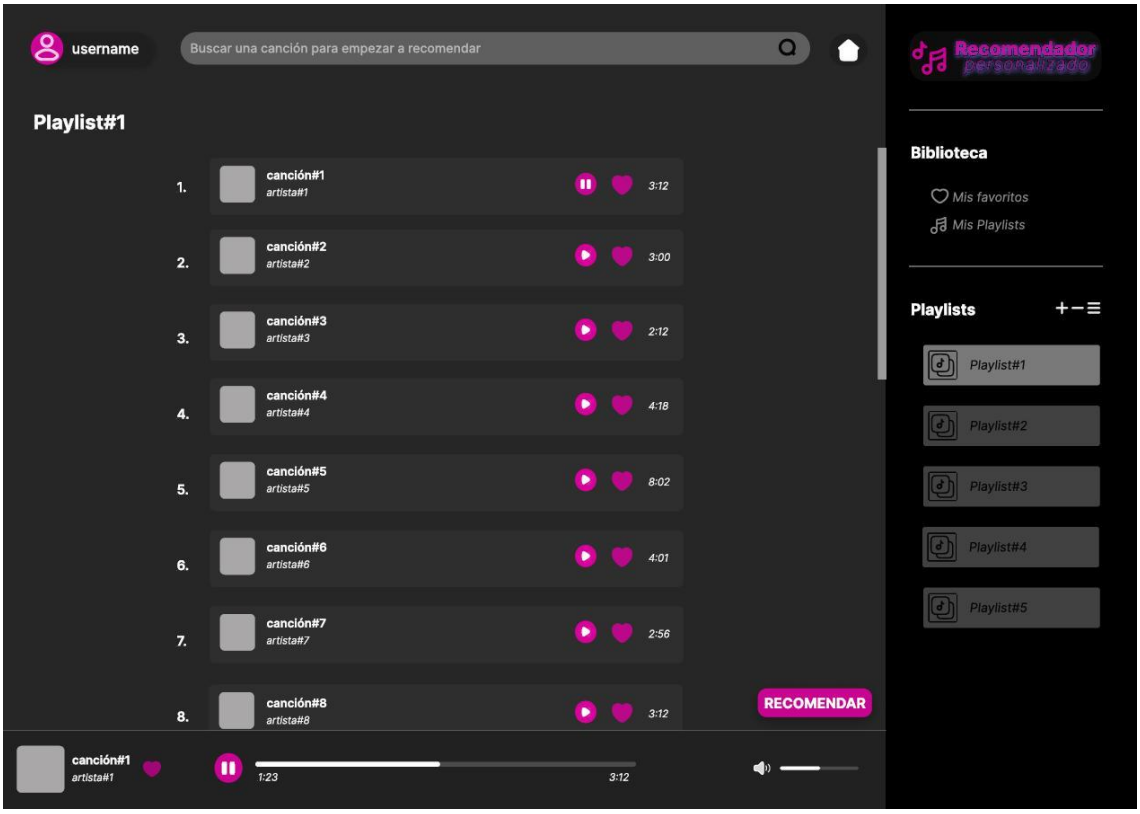

Figura 3-6. Diseño de playlist seleccionada

En la Figura 3.7 encontramos la segunda funcionalidad principal en la que va mostrando canciones en función al perfil del usuario para que éste vaya creando su *playlist* personalizada.

Cada vez que se muestra una canción con su información principal, tenemos varias opciones para tratar esa canción:

1. **Corazón**. El corazón se encuentra lleno si la canción está en la lista de favoritos del usuario, en caso contrario, se encontraría vacío.
2. **Cruz**. Rechaza la canción actual y mostraría otra en función a los gustos del usuario.

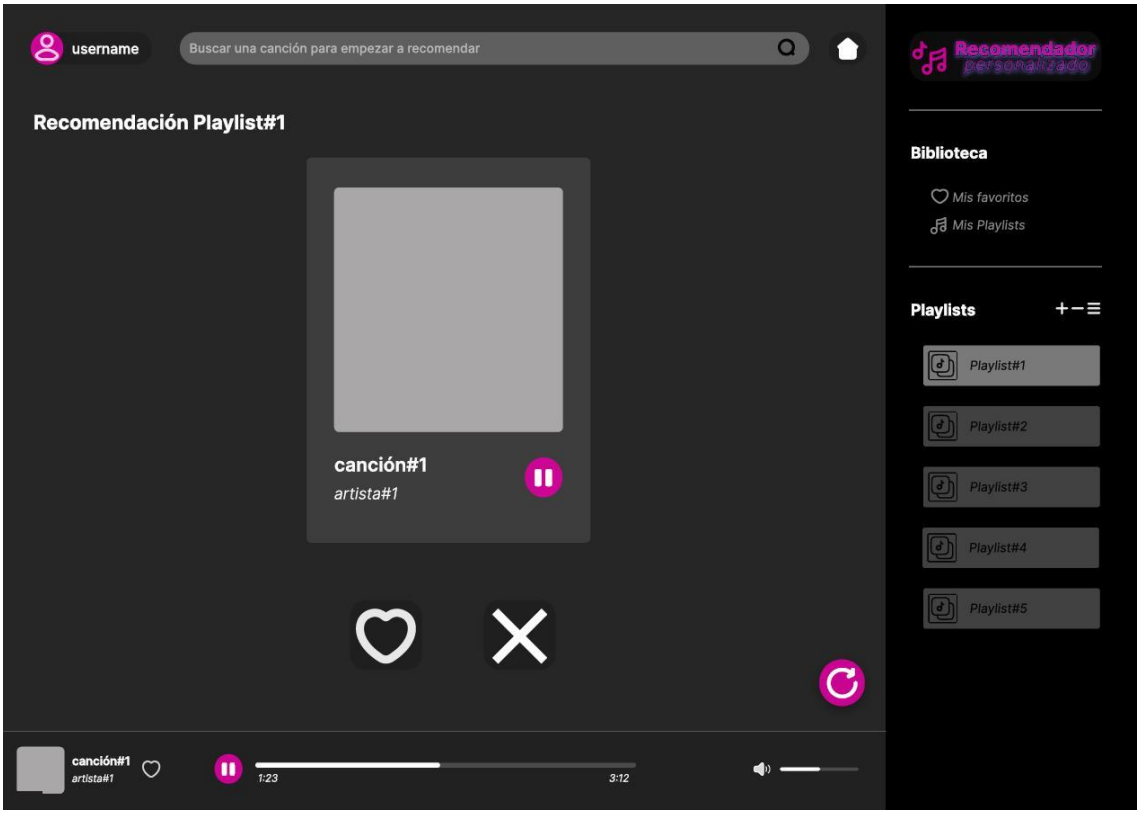

Figura 3-7. Diseño de recomendación

En la Figuras 3.8.1 y 3.8.2 encontramos la tercera funcionalidad principal no mostrada en la página principal en la cual el usuario elige una o varias canciones, artistas y géneros, a partir de los cuáles recomienda (esta funcionalidad no se basa en el perfil del usuario).

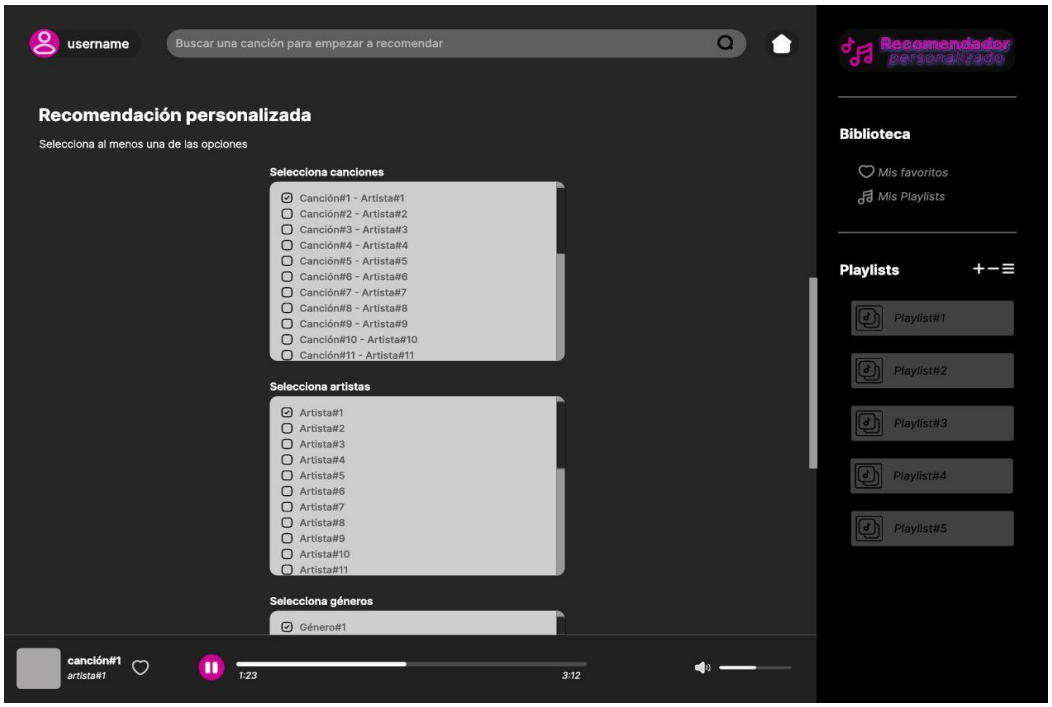

Figura 3-8-1. Diseño de recomendación personalizada

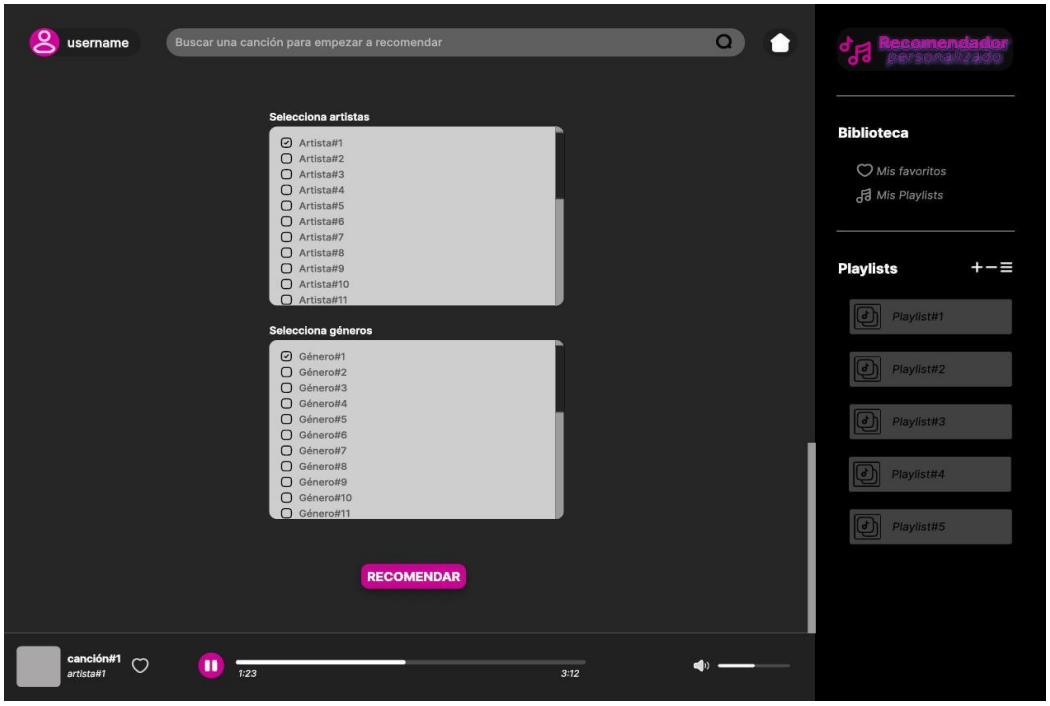

Figura 3-8-2. Diseño de recomendación personalizada

3.4. Desarrollo del algoritmo de recomendación

Durante el transcurso del trabajo hemos desarrollado diferentes recomendadores acorde con las iteraciones o *Sprints* de la metodología *Scrum* (secciones 3.1 y 2.3), mejorando y aprendiendo más acerca de su funcionamiento y aplicación. En total hemos realizado cuatro iteraciones para el recomendador que corresponden a las secciones 3.4.1, 3.4.2, 3.4.3 y 3.4.4, respectivamente.

3.4.1. Primera versión

En la primera versión hemos realizado numerosas iteraciones (sección 3.1), en la que hemos desarrollado un primer recomendador que se desarrolló usamos el F*ramework* (sección 2.4.3) de "*Surprise*" pero se usó mal ya que hicimos que recomendara por géneros parecidos y para ello pasamos los distintos géneros que había en el *dataset* (sección 3.2.1) a números enteros, lo cual fue un error porque los números cercanos no tenían por qué tener relación alguna entre estos. Pese a ello con esta primera iteración pudimos aprender el funcionamiento de las librerías de *Surprise* para hacer un sistema recomendador. Además, en esta primera iteración conseguimos pasar el dataset de Pandas a *Surprise*, lo cual nos será útil para iteraciones posteriores. El dataset empleado para esta iteración fue el que hemos detallado en la sección 3.2.1, con la excepción de que la columna de "Genres" no existía, fue agregada más tarde.

En una segunda iteración estuvimos estudiando cómo trabajar con los datos en el recomendador, para ello nos decantamos por la función de similitud del coseno (figura 3.9) tras investigar los distintos tipos de medidas para la función de similitud (Hug 2015). Ésta en concreto consiste en una medida matemática que se utiliza para evaluar la similitud entre dos ítems, a menudo representadas como vectores. Estos vectores podrían denotar diversas características o atributos de los objetos, como es en el caso de las canciones, el género o la popularidad o la puntuación. La similitud del coseno nos va a permitir evaluar cuán similares son esos vectores entre sí. En la figura 3.9 se muestra la fórmula de la similitud del coseno, donde u y v representan dos ítems o dos usuarios y r_{ui} y r_{vi} son los vectores que corresponden al género o popularidad de los ítems o las calificaciones dadas a un conjunto de ítems por los usuarios (Sondur, Chigadani,

and Nayak 2016). La similitud entre dos ítems puede variar entre 0 y 1, donde 1 representa la máxima similitud y se alcanza cuando ambos ítems son idénticos. (Mana and T.Sasipraba 2021).

$$\text{cosine\_sim}(u,v) = \frac{\sum_{i \in I_{uv}} r_{ui} \cdot r_{vi}}{\sqrt{\sum_{i \in I_{uv}} r_{ui}^2} \cdot \sqrt{\sum_{i \in I_{uv}} r_{vi}^2}}$$

Figura 3-9. Función de similitud del coseno (Hug 2015)

Además, configuramos la función de similitud de nuestro primer recomendador usando la métrica de coseno y el recomendador basado en contenidos (sección 2.1.1.2).

En una tercera iteración usamos la técnica de *Validación cruzada* para comprobar cuán bueno era nuestro modelo en 7 iteraciones (establecido con cv=7). Esta técnica consiste en dividir el conjunto de datos en 7 subconjuntos de manera aleatoria, de los cuales se entrenan 6 y el restante es el encargado de validar con el resto (Berrar 2018). Y como resultado, se muestra la medida de errores de validación en cada iteración, que es nuestro caso, tal como se muestra en la Figura 3.10, fueron números altos lo cual indica que nuestro modelo no era demasiado bueno.

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5	Fold 6	Fold 7	Mean	Std
RMSE (testset)	9.5911	10.9372	10.0591	11.2969	13.7098	9.2653	7.9671	10.4038	1.6914
MAE (testset)	6.9565	7.7298	7.2382	8.2620	9.2726	7.0868	6.4433	7.5699	0.8777
Fit time	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Test time	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Figura 3-10. Resultado de la validación cruzada.

Finalmente, en una cuarta iteración para comprobar el correcto funcionamiento del algoritmo, creamos una *playlist* con 10 canciones de forma aleatoria. Se genera un resultado bueno si hay suficientes canciones del mismo género porque o sino recomienda por otros géneros que no tienen ninguna relación ya que hicimos mal la correlación entre distintos géneros, como hemos dicho al principio de esta sección.

Como conclusión podemos decir que cada iteración nos ha ayudado a entender las librerías de *Surprise* para un sistema de recomendación y cómo funcionan los recomendadores pese a haber usado mal la correlación de los datos. Cada iteración ha tenido una duración de dos semanas.

3.4.2. Segunda versión

En esta segunda versión, hicimos un recomendador muy simple y manual sin usar ninguna librería de *Surprise* para poder empezar a desarrollar la página web e ir poco a poco.

Para esta versión simple pensamos en recomendar las 10 canciones más populares de un artista, sin incluir la canción pasada en el caso de que se haya seleccionado alguna en vez de seleccionar exclusivamente un artista.

Para concluir, esta versión del recomendador es bastante limitada y simple que solo nos ha servido para dar formar al *front-end* de la aplicación con un sistema de recomendación más cierto que la primera versión (sección 3.4.1) pero muy impreciso y que no será útil un dataset de mayor tamaño. Esta versión en el recomendador solo fue una iteración (sección 3.1), el resto fue en el desarrollo de la aplicación (sección 3.5).

3.4.3. Tercera versión

En una tercera versión del recomendador desarrollamos un algoritmo que simula un sistema de recomendación híbrido (sección 2.1.1.3) que combinaba las características de un modelo basado en contenido (sección 2.1.1.2) y filtrado colaborativo (sección 2.1.1.1). A partir de esta iteración, se utilizaron los dos conjuntos de datos descritos en las secciones 3.2.1 y 3.2.2.

Para llevar a cabo esta nueva versión del sistema de recomendación se hizo simulando dos fases, la primera fase sería el basado en contenido correspondiente con la primera iteración (sección 3.1) de la tercera versión del recomendador, en la que el usuario selecciona las opciones para el contenido (popularidad, año, género y duración de la canción), que son las opciones que hemos considerado más relevantes para una

recomendación personalizada que otras como bailable, energía o tempo, entre otras variables descritas en la sección 3.2.1. La segunda fase sería el filtrado colaborativo. Y finalmente, se mezclan las recomendaciones resultantes en una cuarta y última iteración (sección 3.1).

Para llevar a cabo la implementación de la primera fase o basado en contenido (sección 2.1.1.2), empleamos las *matrices de similitud de coseno* en una primera iteración (sección 3.1) de la tercera versión, técnica que se ha explicado previamente en la sección 3.4.1. Como en esta fase se hace por característica individual y no por conjuntos de características, se hace una matriz de similitud de dos canciones solo por popularidad o por género o por otra opción. Así que luego de obtener las matrices de las opciones seleccionadas se hacía una media aritmética que esto se hizo en una segunda iteración (sección 3.1), lo cual hacía que fuera poco eficiente y tardaba mucho tiempo en calcular el resultado.

Para llevar a cabo la implementación de la fase de filtrado colaborativo (sección 2.1.1.1), usamos *matrices de dispersión o de utilidad* junto con el algoritmo de *k-NN* en una tercera iteración (sección 3.1). Las *matrices de utilidad* ayudan a obtener la dispersión de las valoraciones de los usuarios respecto a los ítems, y como vimos en el *cold-start* (sección 2.1.3.1), cuanto más disperso menos preciso es recomendar una canción de forma precisa. El algoritmo de *k-NN* consiste en comparar elementos y dependiendo del parecido en atributos, son esos *k* elementos más parecidos los que se escogen (Rodríguez Rodríguez, Rojas Blanco, and Franco Camacho 2007). Esta fase también tardaba bastante por lo que en la siguiente versión del recomendador (sección 3.4.4) lo mejoramos.

Conclusiones, esta versión del recomendador daba unos resultados aceptados y más preciso que las anteriores iteraciones (secciones 3.4.1 y 3.4.2) pero el tiempo de espera y el consumo de recursos era muy elevado y cuanto mayor era el tamaño de los datasets, tanto canciones como ratings, era imposible saber si el resultado era el esperado debido a la poca eficiencia. Esta versión se realizó en cuatro iteraciones de una duración de dos semanas.

3.4.4. Última versión

En esta última versión del recomendador, finalizamos con la implementación del recomendador y teniendo en cuenta todas las versiones anteriores (secciones 3.4.1 y 3.4.2), mejoramos la tercera versión (sección 3.4.3) del recomendador manteniendo aún las dos fases del recomendador. Las mejoras incluyen: cambio de algoritmo tanto para el recomendador basado en contenido (primera iteración) y filtrado colaborativo (segunda iteración), añadida una tercera fase (tercera y quinta iteración) e implementación de explicabilidad (cuarta iteración).

El cambio del algoritmo de la primera fase o recomendador basado en contenido, que se realizó en la primera iteración de la última versión del recomendador, fue de usar matrices de *similitud de coseno* a *clustering*. El *clustering* es una técnica que agrupa, en nuestro caso, las canciones similares del sistema basándose en las opciones que el usuario selecciona (Son and Kim 2017). Este algoritmo mejoró la precisión de las predicciones, así como aceleró los cálculos de las opciones de popularidad, año y duración de la canción. Para género se dejó *matriz de similitud* ya que era más rápido e igual de preciso que el *clustering*.

El cambio del algoritmo para el recomendador de filtrado colaborativo (sección 2.1.1.1) fue de usar *matriz de similitud* a *matriz de factorización SVD* se realizó en una segunda iteración (sección 3.1), manteniendo el algoritmo de *k-NN*. La *matriz de factorización SVD* aborda los problemas de cobertura limitada y dispersión al proyectar usuarios e ítems en un espacio latente reducido (Ricci, Rokach, and Shapira 2022), es decir, que permite comparar usuarios e ítems en un espacio denso esto permite mayor velocidad y robustez en el aprendizaje, manejo de nuevos usuarios sin necesidad de reentrenamiento y más precisión en las predicciones.

La tercera fase se trata de un recomendador basado en contenido (sección 2.1.1.2) teniendo en cuenta todas las características de las canciones, menos género, y esta tercera fase sólo ocurre en caso de que no se pueda aplicar el filtrado colaborativo (sección 2.1.1.1) ya que no haya ratings suficientes en el sistema y, además, el usuario no haya elegido ninguna opción para una recomendación basada contenido (primera

fase), esto se realizó en una tercera iteración (sección 3.1). Durante las sesiones de evaluación del sistema (sección 4.5) una de las autoras del TFG se dio cuenta que se aplicaba el filtrado colaborativo (sección 2.1.1.1) cuando el usuario no había realizado ninguna valoración y no había seleccionado ninguna opción, por lo que se realizó una quinta iteración (sección 3.1) en la que se permitiera aplicar esta tercera fase en ese caso sino solo se centrara en aplicar el recomendador basado en contenido (sección 2.1.1.2) con las opciones seleccionadas.

Para finalizar, la explicabilidad (sección 2.1.2) ha sido desarrollada en una cuarta iteración de manera que se explique si es por contenido, las características de la canción que se ha basado y de la canción recomendada y que el usuario pueda comparar. En la de filtrado colaborativo, al ser un algoritmo caja negra (sección 2.1.1.1), la explicación se basa en explicar que es por gustos de usuarios similares. En la de basado en contenido la explicación consiste en poner, tanto en la canción basada como la recomendada, los atributos elegidos con sus valores, si se han elegido, para que el usuario pueda contrastar y asegurarse de que sean canciones similares por ello. En la figura 3.31 se puede ver un ejemplo de la explicabilidad en la página.

En conclusión, esta versión del recomendador es bastante eficiente y precisa respecto a las anteriores y el tiempo de espera es muy corto. Además, con todas las versiones hemos aprendido del funcionamiento interno de los recomendadores y hemos acabado implementando uno sencillo de música. En esta versión se ha realizado un total de cinco iteraciones de una duración de entre 1-2 semanas.

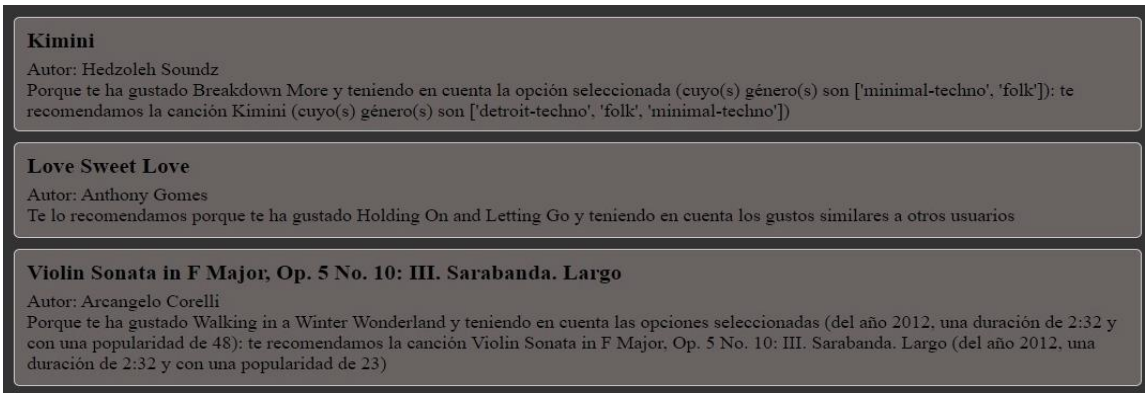

Figura 3-31. Ejemplos de explicabilidad

3.5. Desarrollo de la aplicación web.

Además de la implementación del algoritmo de recomendación (sección 2.1), en este proyecto también se ha desarrollado una aplicación integrada con el recomendador para permitir que las recomendaciones generadas sean accesibles al usuario final a la que hemos nombrado como *Music4U* para que sea reconocible.

El primer desarrollo de la aplicación fue una interfaz sencilla en la que pedíamos al usuario que seleccionara entre diez canciones sacadas aleatoriamente del primer dataset usado como podemos observar en la Figura 3.11. Con esto pudimos realizar la primera integración del recomendador con la web y comprobar su funcionamiento.

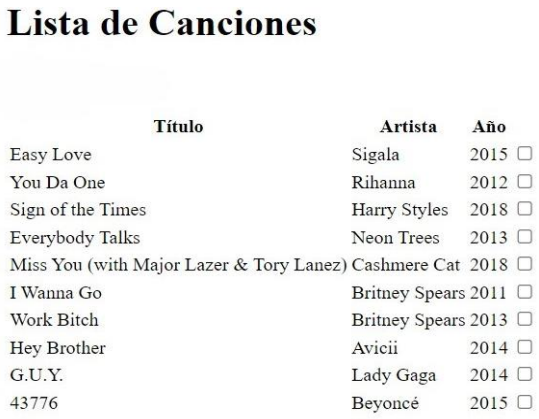

Figura 3-11. Primera interfaz simple

Después de implementar esta funcionalidad y ver que se recomendaba correctamente, añadimos la posibilidad de seleccionar atributos a los que el usuario les quiera dar más peso en la recomendación. La Figura 3.12 muestra la pantalla en la que están los diferentes atributos (artista, género, año, duración y popularidad) y posteriormente mostrará la lista de diez canciones aleatorias como en la Figura 3.11.

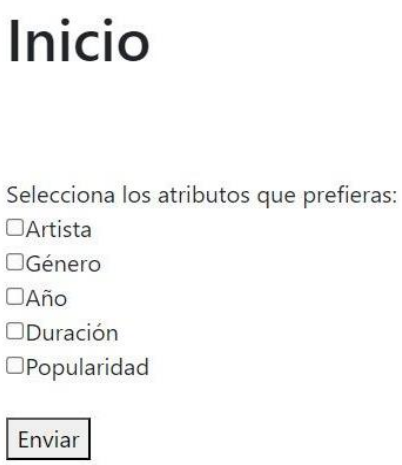

Figura 3-12. Primera interfaz simple: atributos

Lo explicado anteriormente supondría la primera iteración de nuestra página web siguiendo con la metodología descrita en el apartado 3.1.

Se puede resaltar que se logró la conexión con la *API* de *Spotify* mediante la obtención de las credenciales necesarias. Aunque finalmente, no se ha llegado a utilizar.

Una vez tenemos la pequeña prueba de nuestra aplicación con el recomendador integrado y con la ayuda del diseño de las interfaces de usuario como guía de diseño para la implementación de la aplicación web pasamos a codificar nuestro proyecto teniendo en cuenta los dos componentes principales de la una aplicación web: el *back-end* y el *front-end*.

Al principio, empezamos creando la base de datos y cambiando las configuraciones por defecto del *framework Django* para que se adaptaran a nuestras necesidades. Luego, continuamos desarrollando las funcionalidades básicas de toda página web: inicio de sesión y registro, y probamos estas funcionalidades.

Una vez ya teníamos las funcionalidades anteriormente mencionadas, realizamos la página principal. En ella, empezamos a añadir los componentes siguiendo el modelo de los prototipos: *header*, barra lateral y contenido principal.

La construcción de la vista de la página principal se fue alternando con la implementación de las dos funcionalidades principales del proyecto: descubrir nuevas canciones y crear *playlist*.

Cabe destacar que durante todo el proceso de desarrollo de la aplicación web se han ido realizando cambios de diseño y utilidad para hacer la página web más funcional y adaptada a los objetivos establecidos.

3.5.1. Back-End

El *back-End* se encarga de la manipulación de datos y de manejar la lógica de negocio, las operaciones del lado del servidor y la comunicación con la base de datos. (Pérez Ibarra et al. 2021) (Ramón Sanchis 2023)

Para iniciar el *desarrollo* hemos decidido apoyarnos en el Framework *Django* (ver 2.4.3) basado en *Python* (ver 2.4.1) por su rapidez, seguridad y facilidad de uso. Además, este F*ramework* utiliza una arquitectura semejante al *MVC* (ver 2.2), estudiada y aplicada en una de las asignaturas del grado por lo que resulta familiar y sencilla de utilizar.

En cuanto a la base de datos, hemos decidido cambiar *SQLite*, que se configura en *Django* por defecto, a una base de datos *PostgreSQL* (ver 2.4.4) debido a las limitaciones que suponía la anterior base de datos en cuanto a los tipos de datos soportados y el rendimiento ofrecido. Aunque el uso de nuestra base de datos es muy simple, es bueno que el sistema pueda gestionar la aplicación si se introdujeran cambios más potentes en ella o mayor cantidad de datos.

La elección de la base de datos se afianzó conforme íbamos desarrollando las tablas debido a que, en un primer momento, no se tuvieron en cuenta todos los campos de cada una de las tablas y se tuvieron que cambiar para satisfacer las necesidades de la aplicación.

En la Figura 3.13 podemos observar el diagrama entidad-relación realizado con *Canva* muestra las distintas tablas y sus relaciones implementadas en la base de datos.

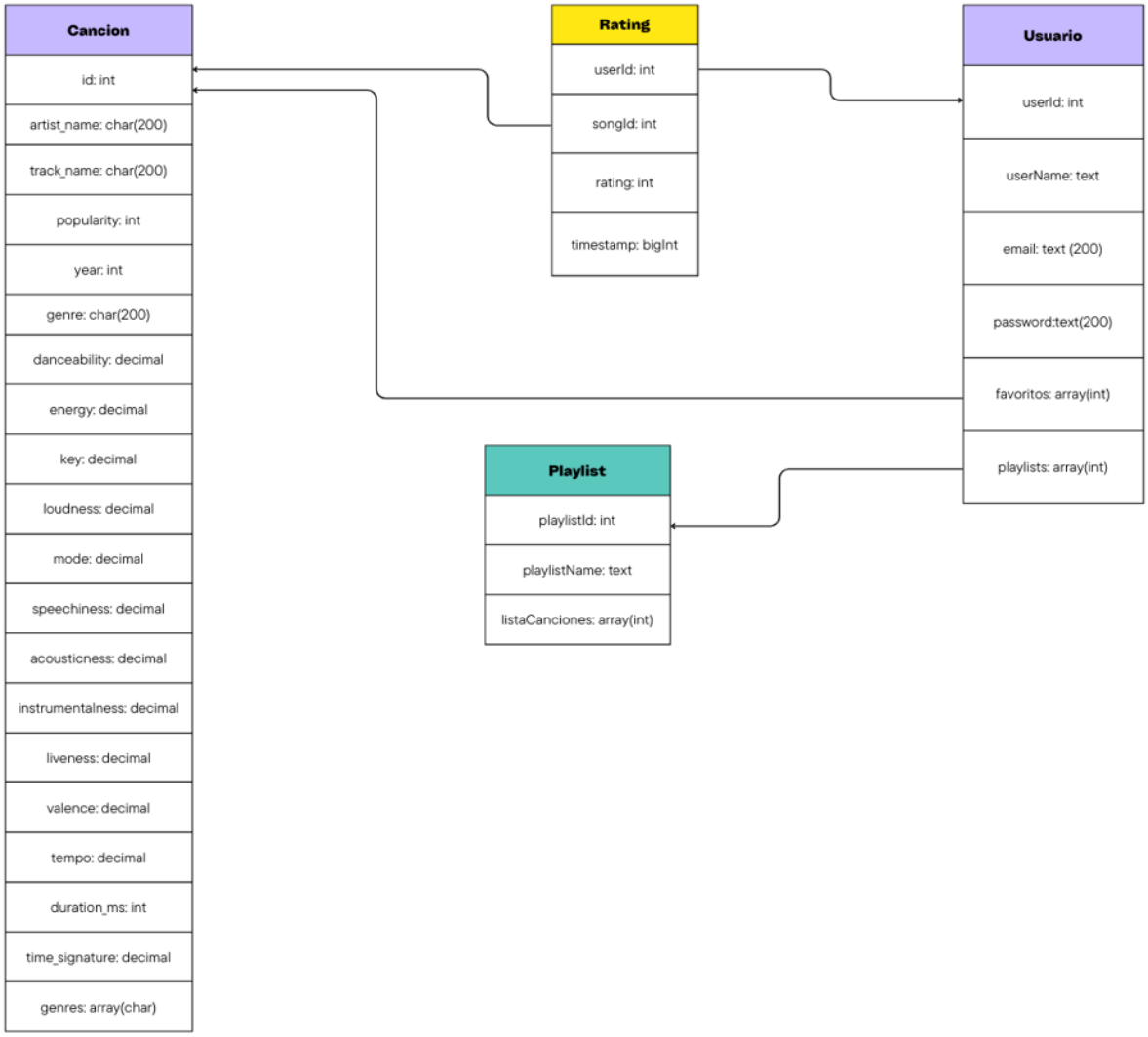

Figura 3-13. Diagrama entidad-relación de la base de datos

A continuación, vamos a realizar una explicación detallada de las tablas de nuestra base de datos:

3.5.1.1. Tabla "Cancion".

La tabla "Cancion" contiene todas las canciones que van a ser usadas en nuestra aplicación. Todas las canciones contienen "id" que es la clave primaria y el resto de los atributos de las canciones fueron descritas en detalle anteriormente (ver 3.2.1)

En las Figuras 3.14.1, 3.14.2 y 3.15 podemos ver un ejemplo de lo que contiene la tabla "Cancion".

id [PK] bigint	artist_name character varying (200)	track_name character varying (200)	track_id character varying (200)	popularity integer	year integer	genre character varying (200)	danceability numeric (50,6)	energy numeric (50,6)	key numeric (50,6)
0	Jason Mraz	I Won't Give Up	53QF56cjZA9RTuuMZDrSA6	68	2012	acoustic	0.483000	0.303000	4.000000
1	Jason Mraz	93 Million Miles	1s8tP3jP4GZcyHDsjvw218	50	2012	acoustic	0.572000	0.454000	3.000000
2	Joshua Hyslop	Do Not Let Me Go	7BRCa8MPiyuvr2VU3O9W0F	57	2012	acoustic	0.409000	0.234000	3.000000
3	Boyce Avenue	Fast Car	63wsZUhUZLlh1OsyrZq7sz	58	2012	acoustic	0.392000	0.251000	10.000000
4	Andrew Belle	Sky's Still Blue	6nXIYClvJAfi6ujLiKqEq8	54	2012	acoustic	0.430000	0.791000	6.000000
5	Chris Smither	What They Say	24NvptbNKGs6sPy1Vh1O0v	48	2012	acoustic	0.566000	0.570000	2.000000
6	Matt Wertz	Walking in a Winter Wonderland	0BP7hSvLAG3URGrEvNNbGM	48	2012	acoustic	0.575000	0.606000	9.000000
7	Green River Ordinance	Dancing Shoes	3Y6BuzQCg9p4yH347Nn8OW	45	2012	acoustic	0.586000	0.423000	7.000000
8	Jason Mraz	Living in the Moment	3ce7k1L4EkZppZPz1EJWTS	44	2012	acoustic	0.650000	0.628000	7.000000
9	Boyce Avenue	Heaven	2EKxmYmUdAVXlaHCnnW13o	58	2012	acoustic	0.619000	0.280000	8.000000

Figura 3-14-1. Tabla "Cancion". Parte 1

loudness numeric (50,6)	mode numeric (50,6)	speechiness numeric (50,6)	acousticness numeric (50,6)	instrumentalness numeric (50,6)	liveness numeric (50,6)	valence numeric (50,6)	tempo numeric (50,6)	duration_ms integer	time_signature numeric (50,6)
-10.058000	1.000000	0.042900	0.694000	0.000000	0.115000	0.139000	133.406000	240166	3.000000
-10.286000	1.000000	0.025800	0.477000	0.000014	0.097400	0.515000	140.182000	216387	4.000000
-13.711000	1.000000	0.032300	0.338000	0.000050	0.089500	0.145000	139.832000	158960	4.000000
-9.845000	1.000000	0.036300	0.807000	0.000000	0.079700	0.508000	204.961000	304293	4.000000
-5.419000	0.000000	0.030200	0.072600	0.019300	0.110000	0.217000	171.864000	244320	4.000000
-6.420000	1.000000	0.032900	0.688000	0.000002	0.094300	0.960000	83.403000	166240	4.000000
-8.197000	1.000000	0.030000	0.011900	0.000000	0.067500	0.364000	121.083000	152307	4.000000
-7.459000	1.000000	0.026100	0.252000	0.000006	0.097600	0.318000	138.133000	232373	4.000000
-7.160000	1.000000	0.023200	0.048300	0.000000	0.119000	0.700000	84.141000	235080	4.000000
-10.238000	0.000000	0.031700	0.730000	0.000000	0.103000	0.292000	129.948000	250063	4.000000

Figura 3-14-2. Tabla "Cancion". Parte 2

genres character varying[]
{progressive-house,samba,sad,rock-n-roll,dubstep}
{rock}
{rock-n-roll}
{german,classical}
{ambient,sad}
{pop-film,sleep,emo,indian}
{electro,show-tunes,minimal-techno,hardcore}
{ska,emo,minimal-techno}
{chicago-house,sad,acoustic,indie-pop}
{hard-rock,salsa,goth,hardcore,sad}

Figura 3-15. Tabla "Cancion". Parte 3

3.5.1.2. Tabla Rating.

La tabla "Rating" contiene todas las valoraciones realizadas por los usuarios sobre las canciones (les ha gustado o no la canción). Todos los ratings contienen "id" que es la clave primaria y el resto de los atributos de los ratings fueron descritas en detalle anteriormente (ver 3.2.2)

En la Figura 3.16 podemos ver un ejemplo de lo que contiene la tabla "Rating".

id [PK] bigint	userId integer	songId integer	rating integer	timestamp bigint
1	520	8928	1	1705967781
2	23	6730	0	1705960446
3	328	1045	0	1705953990
4	340	905	1	1705972539
5	19	791	0	1705957124

Figura 3-16. Tabla "Rating".

3.5.1.3. Tabla Playlist.

La tabla "Playlist" contiene todas las *playlist* de todos los usuarios de la página web. Todas las playlists contienen "playlistId" que es la clave primaria, "playlistName" (nombre de la playlist) y "listaCanciones" (lista de *ids* de las canciones que contiene esa playlist).

En la Figura 3.17 podemos ver un ejemplo de lo que contiene la tabla "Playlist".

id [PK] bigint	playlistId integer	playlistName text	listaCanciones integer[]
96	1	playlist1	{1,2,3,4,5,6,7,8,9,10,11}
97	2	playlist2	{0,641,6402,3712,4481,2049,9347,133,1792,1922}
98	3	playlist3	{5637,2825,7562,3083,4245,9622,6301,8351,9380,8616}
116	4	playlist4	{8323,7556,648,2825,5385,522,7562,7697,7449,4889}
117	5	playlist5	{1952,4672,4366}
118	6	playlist6	{1780,1376}

Figura 3-17. Tabla "Playlist".

3.5.1.4. Tabla Usuario.

La tabla "Usuario" contiene la información de todos los usuarios registrados de la página web. Todos los usuarios contienen "userId" que es la clave primaria, "userName" (nombre de usuario único), "email" (correo electrónico del usuario que no se puede repetir para varias cuentas), "password" (contraseña de la cuenta), "favoritos" (lista de ids de las canciones que le han gustado al usuario) y "playlists" (lista de ids de las playlists que tiene cada usuario).

No hemos hecho diferencia de roles en los usuarios porque al tratarse de una página web simple, la diferencia entre un administrador y un usuario común, no habría tenido mucha relevancia y no habría aportado ningún valor añadido.

En la Figura 3.18 podemos ver un ejemplo de lo que contiene la tabla "Usuario".

userId integer	userName text	email text	password text	favoritos integer[]	playlists integer[]
1	Leire	leire@ucm.es	1234	{22,23,25,26,29,33,17,1770,2687,34,2550,32,8326,4452,6925,4889,21}	{1,2,3,4}
2	guille02	guillermo@gmail.com	1	{1, 8, 10, 673}	{10,11}
5	mario	mario@gmail.com	mario	{2724,7556,6797}	{16,17,18}
6	javi	javi@ucm.es	hola1234	{8712}	{19,20}

Figura 3-18. Tabla "Usuario".

3.5.2. Front-End

El *Front-End* se refiere a la parte de una aplicación con la que interactúa el usuario, la que crea la experiencia. Aquí incluimos la interfaz gráfica y las funcionalidades de la página web.

Como hemos explicado anteriormente, antes de comenzar a codificar la página web, realizamos una serie de diseños con las funcionalidades principales. Con el transcurso del proyecto se han ido cambiando progresivamente para cubrir las necesidades incipientes y poder solventar los problemas que iban surgiendo a medida que avanzamos.

Para realizar el Front-End de la aplicación nos hemos apoyado en *HTML*, *CSS* y *JavaScript*, con el apoyo de *jQuery* (ver 2.4.1).

Las funcionalidades implementadas tratan de ser intuitivas para que el usuario no experimente confusión al usarla. También, se ha puesto el foco en hacer la página lo más simple posible y sin utilizar elementos que puedan distraer al usuario de su función principal.

A continuación, describiremos las pantallas finales de la aplicación.

1. **Inicio de sesión**. En esta página el usuario iniciará sesión en su cuenta introduciendo su correo electrónico y su contraseña. En el caso de no tener cuenta en la aplicación, se da la opción de registro.

Figura 3-19. Inicio de sesión

2. **Registro**. En esta página permite al usuario registrarse en la aplicación introduciendo su correo electrónico, nombre de usuario y contraseña.

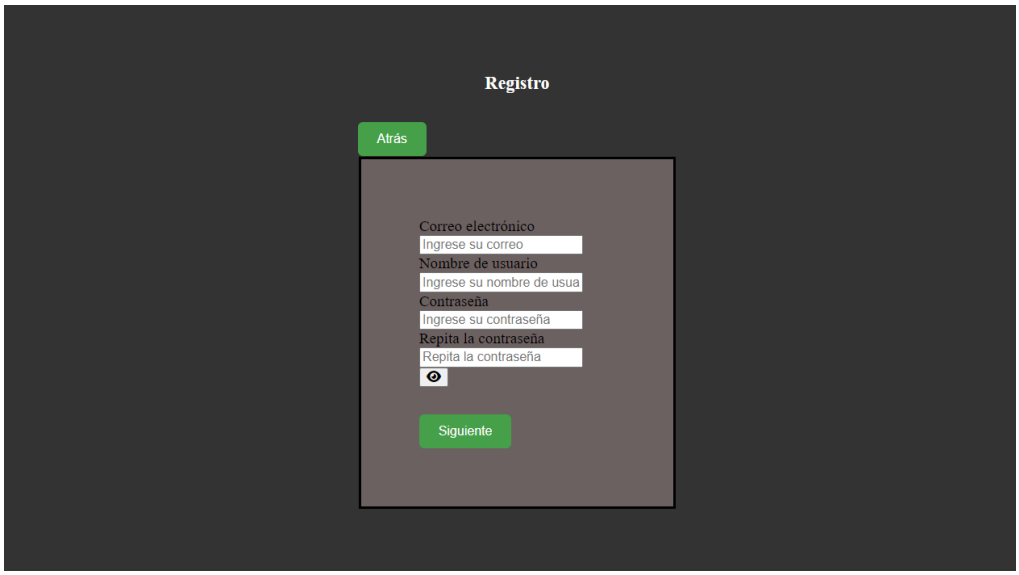

Figura 3-20. Registro

3. **Página principal**. Esta página encontramos la cabecera con el icono de la casa, que sirve para volver a la página principal; y el icono de usuario, que lleva a la sección de "Mi perfil" con la información de usuario.

   En la barra lateral de la derecha encontramos dos clickables para acceder a la biblioteca de "Mis favoritos" y la de "Mis playlists". También encontramos la lista de las playlists del usuario con sus nombres.

   En la parte central, encontramos dos cajas con las funcionalidades que ofrece nuestra página web: descubrir nuevas canciones y crear *playlist*.

   En la figura 3.21 podemos observar la página principal con un usuario que ha iniciado sesión, en este caso, el usuario "Leire".

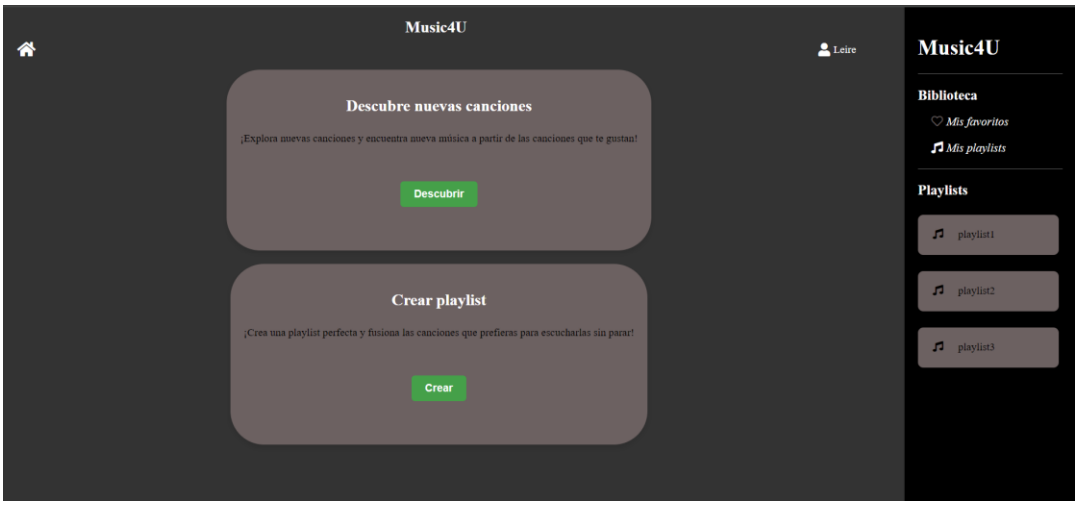

Figura 3-21. Página principal con usuario registrado.

En el caso de que el usuario no haya iniciado sesión, los botones de las funcionalidades están desactivadas y en la parte en la que se mostrarían las playlists del usuario, indica que se debe iniciar sesión para poder visualizarlas. La vista la podemos encontrar en la Figura 3.22.

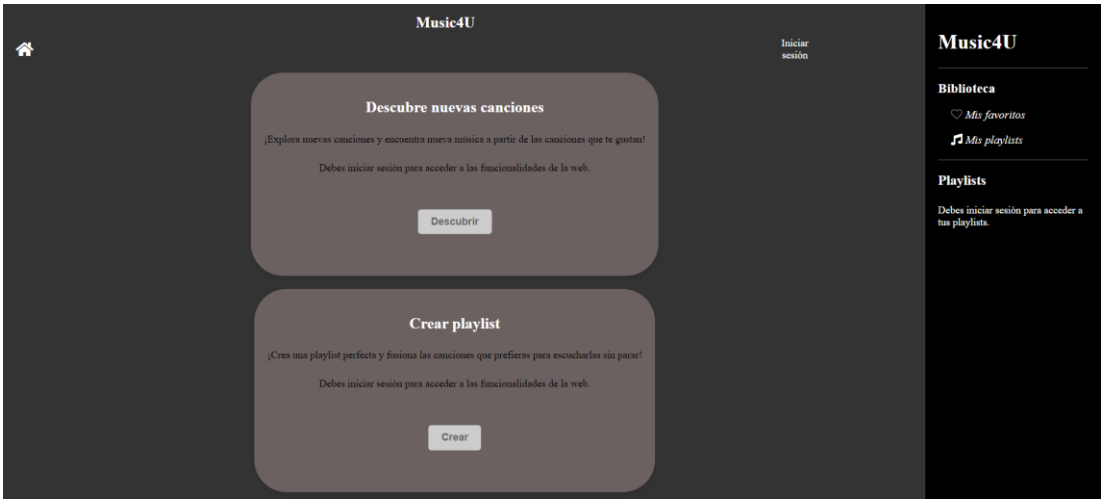

Figura 3-22. Página principal con usuario no registrado.

4. **Descubrir canciones**. En esta primera funcionalidad el usuario obtendrá una lista de canciones recomendadas (canciones que el usuario no tiene en su lista de favoritos) a partir de unos atributos y canciones que le gustan al usuario.

El usuario tendrá la opción de marcar ninguno, uno o varios atributos (género, año, duración y popularidad) según la relevancia que le quiera dar a la hora de obtener una recomendación como refleja la Figura 3.23.

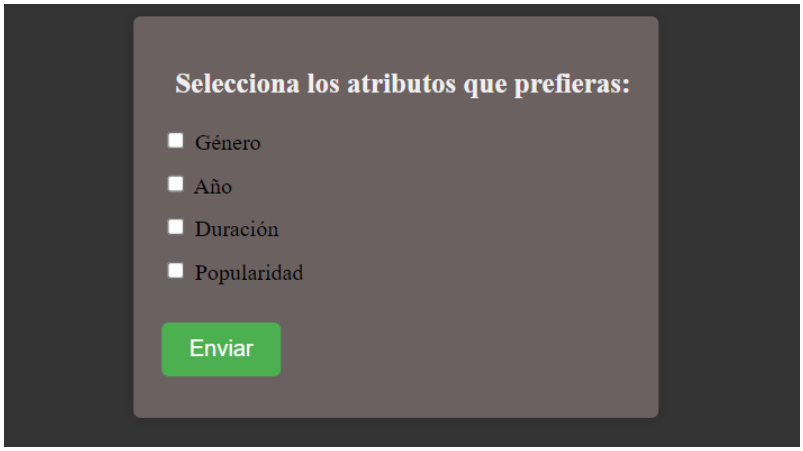

Figura 3-23. Crear playlist: elección de atributos

Una vez seleccionados los atributos, se mostrará una lista con veinte canciones aleatorias de la lista de favoritos del usuario como se puede ver en la Figura 3.24. En el caso de que al usuario en ese momento no le hayan gustado al menos veinte canciones, se mostrarán las canciones favoritas hasta el momento. También se puede dar que el usuario no haya añadido ninguna canción a favoritos, por lo tanto, se mostrarán veinte canciones populares.

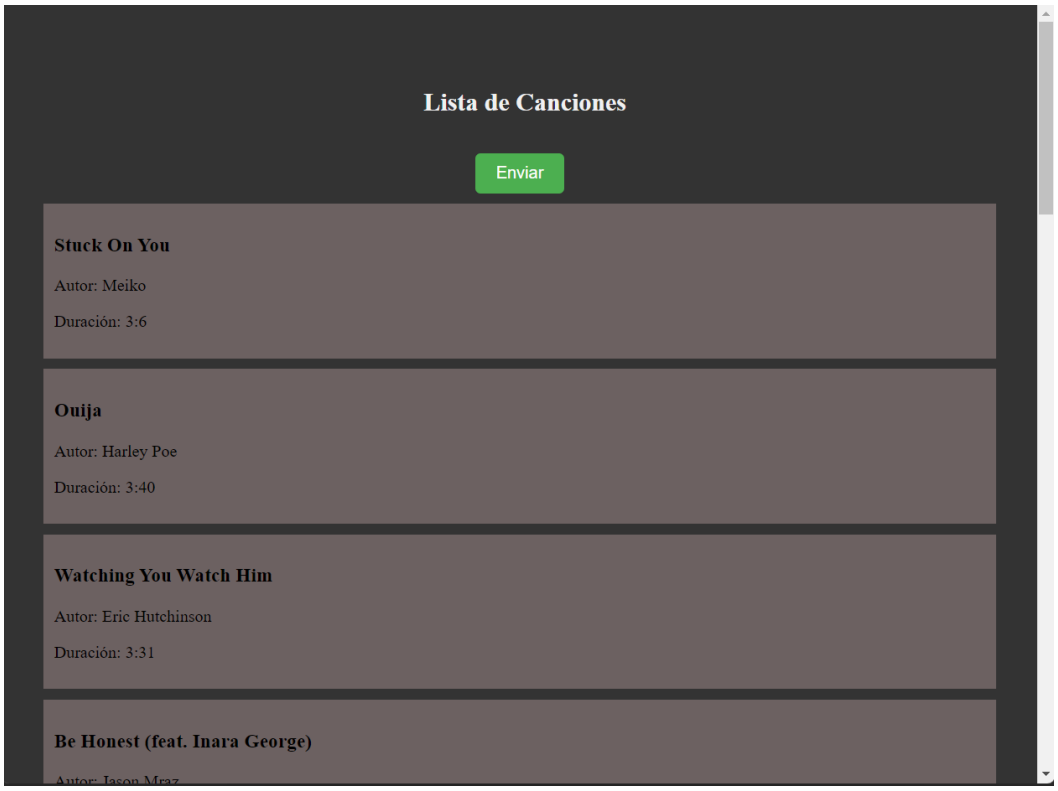

Figura 3-24. Crear playlist: selección de canciones

Tras realizar los pasos anteriores, se mostrará una lista de canciones y dará la opción al usuario de guardar la playlist en la biblioteca (ver Figura 3.25).

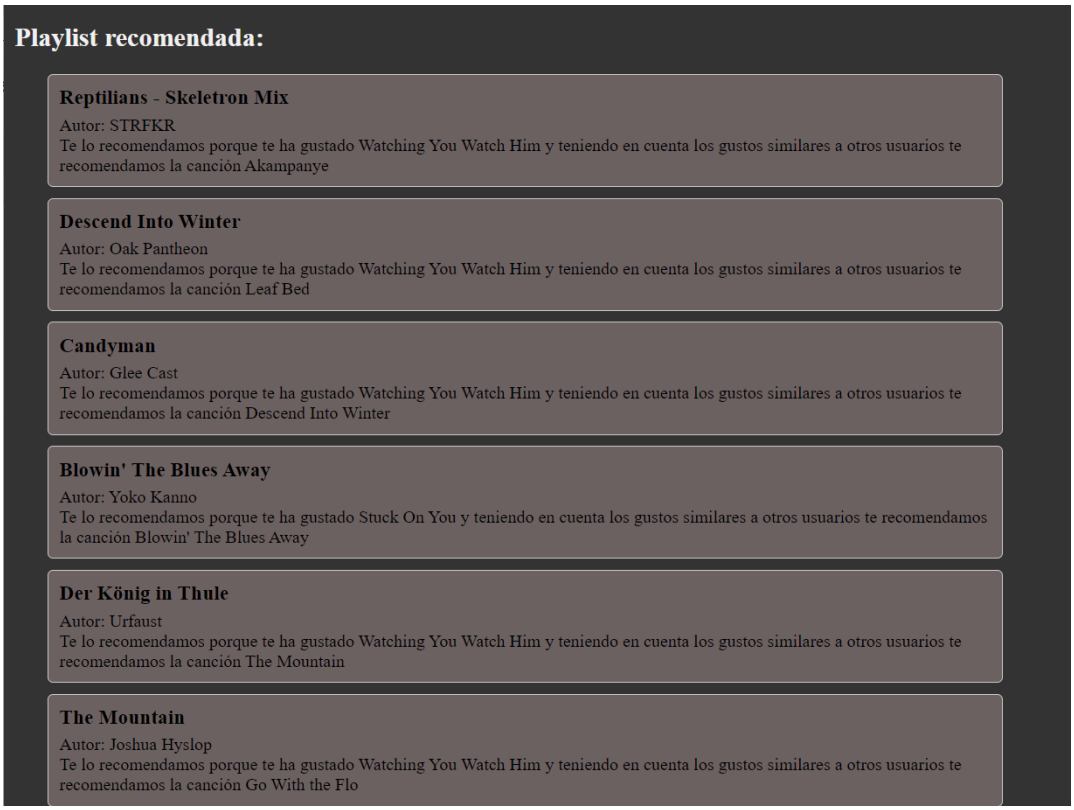

Figura 3-25. Crear playlist: selección de canciones

5. **Crear *playlist***. En esta segunda funcionalidad el usuario podrá generar una *playlist* totalmente personalizada en la que podrá introducir las canciones que desee.

   En un principio el usuario deberá introducir el nombre que le quiere asignar a la nueva *playlist* que va a crear y la aplicación le irá mostrando una canción para que el usuario decida si le interesa meterla en la *playlist* o no.

   Primero te muestra una canción de la lista de favoritos del usuario (si no tiene canciones en la lista de favoritos, se muestran directamente las más populares), y en el caso de que no le interese meter esta canción en la *playlist*, empieza a mostrar las canciones más populares para que el usuario pueda tener más variedad a la hora de recomendar y no sólo centrarse en las canciones que le han gustado.

El usuario puede elegir para cada canción si quiere meterla en la *playlist*, pasar la canción (no meterla en la *playlist* y mostrar otra canción) y cuenta con un botón de favoritos para que el usuario pueda quitar o poner la canción en su lista de favoritos. En todo momento el usuario tendrá la posibilidad de crear la *playlist* y añadirla a la biblioteca de playlists (ver Figura 3.26)

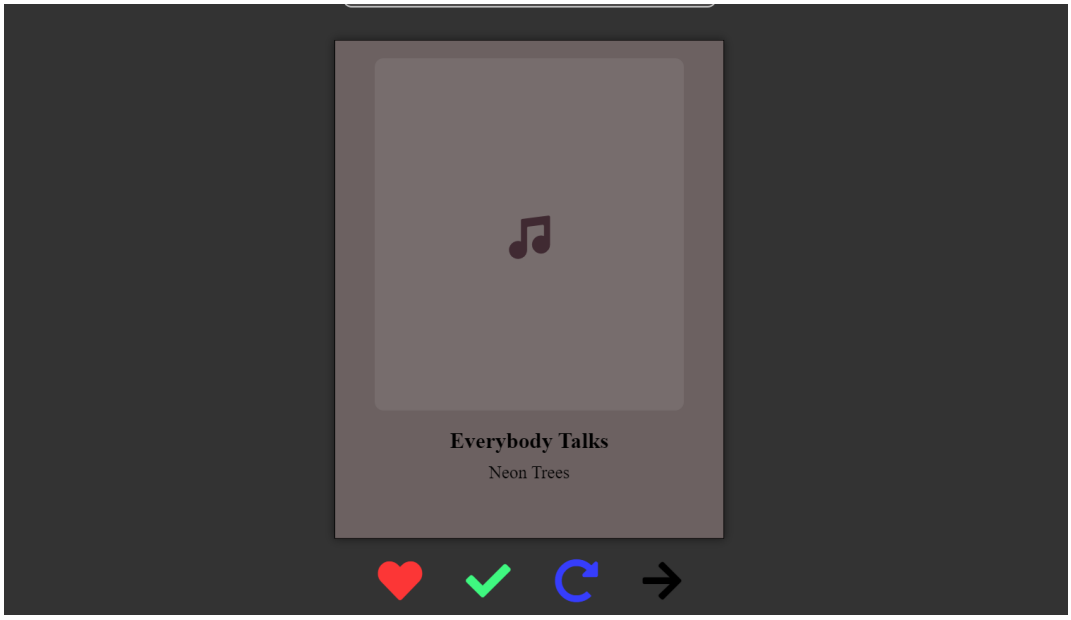

Figura 3-26. Crear playlist

En la Figura 3.27 podemos ver un diagrama del patrón que sigue la funcionalidad en función de si el usuario decide o no meter la canción mostrada en la *playlist*.

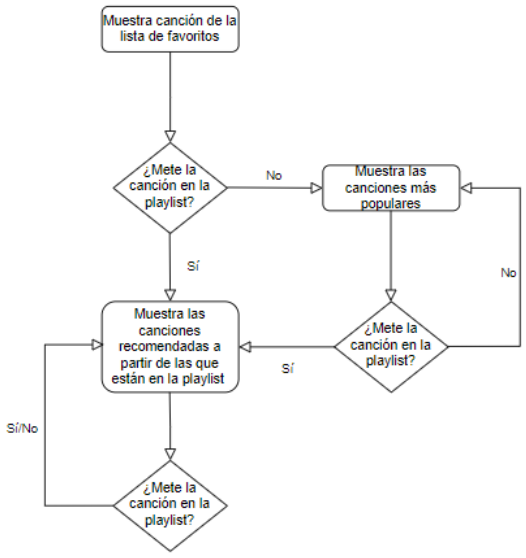

Figura 3-27. Crear playlist: diagrama de flujo

6. **Mis favoritos**. En esta página encontramos una lista con las canciones que le han gustado al usuario. Tiene la posibilidad de quitar canciones de esta lista en el icono del corazón de la derecha (ver Figura 3.28)

Figura 3-28. Mis favoritos

7. **Mis playlists**. En esta página encontramos una lista con las playlists del usuario (ver Figura 3.29). Tiene la posibilidad de ver las canciones que tiene cada *playlist* (ver Figura 3.30)

Figura 3-29. Mis playlists

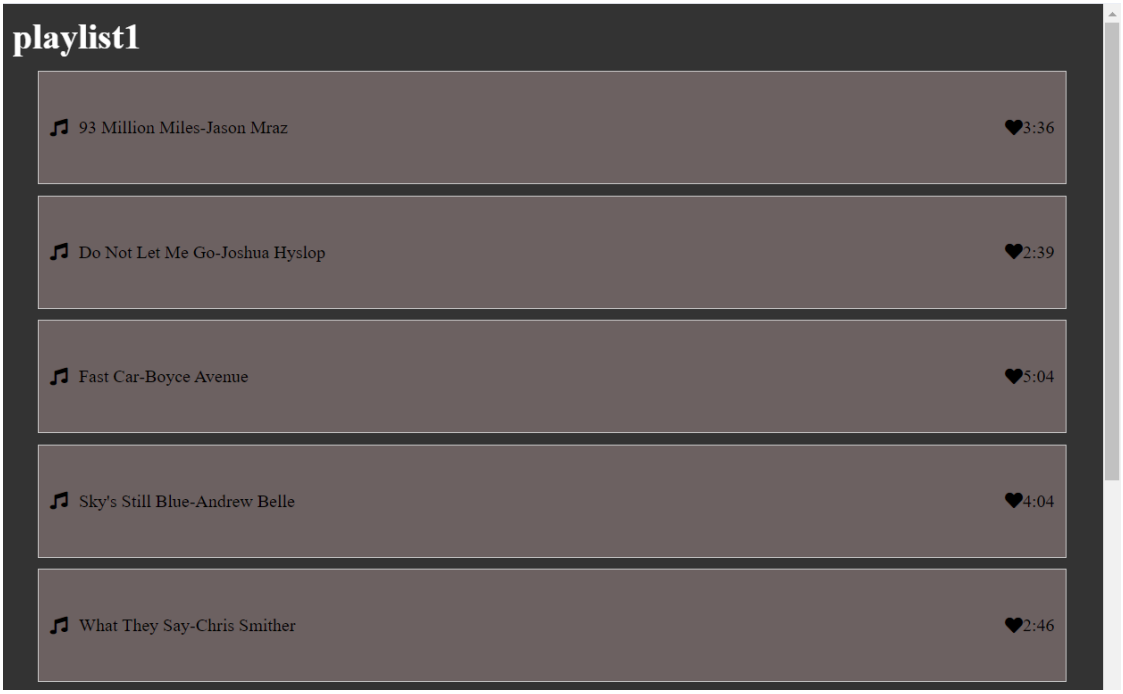

Figura 3-30. Mis playlists: lista de canciones de "playlist1"

Se ha decidido realizar los diseños en *CSS* sin ninguna herramienta porque nos parece interesante poder aplicar los conocimientos aprendidos en asignaturas como

"Aplicaciones Web". Así como, ser capaces de desarrollar nuestro código y poder cambiarlo para adaptarlo a nuevas necesidades.

Dentro del proyecto tenemos:

1. **Modelo**. se encarga de interactuar con la base de datos *PostgreSQL*.
2. **Vista**. se encuentra en el fichero *views.py* y se encargan de recibir y manejar las solicitudes del cliente según sea necesario. En nuestro proyecto algunas de esas funciones se comunicaban con el modelo.

   También tenemos el fichero *urls.py* que asocia un enlace con la función de la vista correspondiente.

3. **Plantilla**. utilizamos *HTML, CSS* y *JavaScript* para cada vista de nuestra aplicación.

   Hemos usado *JavaScript* para aportar un comportamiento dinámico a la página web. Además, utilizamos *AJAX* para enviar datos a la vista y poder tratarlos ahí.

Capítulo 4 - Evaluación con usuarios

La evaluación con usuarios finales resulta de mucha utilidad para el desarrollo y la mejora de aplicaciones web debido a que se puede recabar información importante sobre la experiencia de usuario y la satisfacción del cliente sobre tu página.

A continuación, se detalla la planificación y el análisis de los resultados obtenidos sobre la evaluación realizada a varios usuarios con el objetivo de encontrar fallos y áreas de mejora en el proyecto.

Esta evaluación con usuarios ha sido planteada con los conocimientos obtenidos en una asignatura "Desarrollo de Sistemas Interactivos" impartida en el Grado de Ingeniería Informática (Francisco Gilmartín 2023)

4.1. Preparación del plan de evaluación

El propósito de esta evaluación con usuarios es, debido a la finalización de la aplicación, asegurar la utilidad de la aplicación y realizar cambios según la retroalimentación de los usuarios que vayan a probar esta.

Asimismo, el objetivo es comprobar que la interfaz y el recomendador se adecúa a las expectativas de los usuarios y que puedan usar la aplicación de forma correcta por sí mismos.

Para llevar a cabo esto, hemos establecido un plan que consiste en seleccionar a los participantes de esta evaluación, aquellos usuarios que usen aplicaciones para escuchar música como *Spotify*, *YouTube* o *Apple Music*. Seguido de esto hemos desarrollado una lista de tareas a realizar en unas pruebas guiadas, donde las desarrolladoras de la aplicación actuaremos de moderadoras y anotaremos los problemas que vayan surgiendo. Además, estas pruebas se llevarán a cabo en el ordenador de una de las desarrolladoras.

Las tareas a realizar por los participantes de la evaluación con usuarios:

1. Registrarse.

2. Cerrar sesión e iniciar sesión.
3. Usar la funcionalidad "Descubrir nuevas canciones" sin opciones.
4. Usar la funcionalidad "Descubrir nuevas canciones" seleccionando alguna opción.
5. Usar la funcionalidad "Crear playlist".
6. Mirar "Mis playlists", acceder a las canciones que tiene dicha *playlist* y comprobar que son las que se han añadido.
7. Mirar "Mis favoritos" y quitar una canción.
8. Ir a perfil y volver a la página principal.
9. Cerrar sesión.

Las anotaciones realizadas por las moderadoras se centrarán sobre todo en aquellas tareas que hayan resultado más tediosas y, por tanto, serán las funcionalidades a mejorar con mayor prioridad.

4.2. Preparación del entorno de evaluación

La prueba se realizará de forma presencial en el ordenador de una de las desarrolladoras, previamente todo preparado para el inicio de esta. Se realizará en un entorno tranquilo, en el que estarán presente las dos o una de las moderadoras junto con el participante para mayor tranquilidad.

4.3. Encontrar y seleccionar a los usuarios

Para esta prueba hemos contactado con 6 participantes que usan aplicaciones para escuchar música como Spotify o Apple Music.

Antes de realizar la prueba, hemos realizado una entrevista corta con cada uno de los participantes para comprobar si su perfil encaja con el buscado. En dicha prueba hemos realizado una serie de preguntas:

1. ¿Escuchas música con frecuencia?
2. ¿Utilizas plataformas o aplicaciones para descubrir nueva música?

3. ¿Cuándo escuchas música, te gusta que la aplicación te recomiende o prefieres elegir manualmente las canciones?

4.4. Preparación de los materiales para la evaluación

Para asegurar el éxito de la prueba se ha realizado una preparación previa que va a consistir en:

1. Una introducción a la utilidad de aplicación, es decir, de qué trata nuestra aplicación.
2. Realizaremos una entrevista muy corta antes de las pruebas para reclutar a los participantes.
3. Se anotará el comportamiento de cada participante con la aplicación.
4. Una lista de tareas a realizar, descritas en la sección 4.1.
5. Y para finalizar con la prueba se distribuirá un cuestionario para calificar la sesión y la aplicación según lo observado durante la ejecución de las tareas.

4.5. Sesiones de evaluación

Durante las sesiones de evaluación con cada usuario hemos seguido el plan de evaluación donde el usuario ha seguido las tareas descritas. Todo este proceso ha sido supervisado y guiado por moderadores, en este caso, las desarrolladoras del proyecto para conseguir mejores resultados.

Hemos utilizado la técnica *Think-aloud*, estudiada en la asignatura de "Desarrollo de Sistemas Interactivos" (Francisco Gilmartín 2023) en la se pide a los participantes que expresen en voz alta lo que sienten o piensen en cada prueba. De esta manera, el moderador puede recoger toda esta información para su posterior análisis.

En las tablas 4.1, 4.2, 4.3 y 4.4, se muestran las opiniones de los participantes de las sesiones de evaluación antes de realizar cambios.

Tarea	Opinión
1	El proceso ha sido gratificante.
2	El proceso ha sido sencillo.
3	Pregunta si es aleatorio la lista de canciones a seleccionar. Dice que no es intuitivo porque no hay *checkbox* pese a los cambios de colores. Debajo del texto "Lista de canciones" debería haber información de que hay que hacer, si seleccionar o qué sucede a continuación. Debajo del texto de "Playlist recomendada" se necesita información de que son esas canciones recomendadas. Hay que destacar la canción de algún modo en el texto de la explicación de la recomendación y el color de las tarjetas de recomendación debería ser más destacable. Poner en explicabilidad en algunas opciones similares, mejorar la explicación de filtrado colaborativo.
4	Hay que explicar que hace las opciones, por ejemplo: "Las opciones seleccionadas se van a tener en cuenta en la recomendación". Dejar el *header* fijo de lista de canciones y solo bajar las canciones.
5	Los botones con información, no se entienden. Quiere saber las canciones que va añadiendo.
6	Debería redireccionar las playlists laterales y no sólo en mis playlists. Separar corazón más de duración.
7	Si no tienes favoritos, que ponga un mensaje informando.
8	Centrar el botón de cerrar sesión.
9	No ha surgido ningún problema.

Tabla 4-1. Usuario 1 (22 años)

Tarea	Opinión
1	Si el usuario se equivoca de contraseña, tiene que rellenar todo de nuevo. Debería permanecer el correo y usuario.

2	Le ha parecido fácil.
3	La explicación está mal escrita. Le ha gustado la funcionalidad. Los colores le parecen tristes.
4	Quiere un "*refresh*" para seleccionar nuevas canciones, y no sólo 20.
5	No entiende los símbolos de los botones, quiere información. No entiende el símbolo de la flecha azul. No le transmite confianza al crear *playlist* por el pop-up.
6	Sale alguna canción repetida, pero en general le parece bien. No le gusta que se abran las ventanas.
7	No se pueden añadir favoritos desde mis playlists. Quitar corazón de *playlist* si no es para añadir en favoritos.
8	No ha surgido ningún problema.
9	No ha surgido ningún problema.
Extra	En la página principal tiene que indicar que inicie sesión y no sólo inhabilitar los botones.

Tabla 4-2. Usuario 2 (21 años)

Tarea	**Opinión**
1	Debería haber un botón de registrarse en la página principal, no solo iniciar sesión.
2	Debería dar la opción de iniciar sesión con el nombre de usuario. El icono de usuario ya que no permite foto debería ser más original. El fondo le parece deprimente.
3	Falta Información en lista de canciones, quiere saber si son esas canciones tipo populares o de tu *playlist*. Falta más canciones, necesita ver más. Más información en la tarjeta de canciones: autor, duración, género y año. Falta información en esta sección. Acortar la explicabilidad: porque te ha gustado

	tal y a otros usuarios similares. Al final, en la *playlist* recomendada, se necesita información como: creemos que estas canciones te gustarán.
4	Lo mismo que en la tarea 3. Cambiar la explicación a un lenguaje más humano.
5	Explicar los botones y añadir un botón que ponga terminar en lugar de la flecha negra. Cambiar el botón de refrescar, no se entiende. El texto de la explicación en rojo parece un error. La ventana de confirmación de crear *playlist* debería ser igual que el resto en formato.
6	Ir a *playlist* sin darle a la flecha (darle a la tarjeta de la *playlist*) La flecha es muy pequeña y no resalta cuando pasa por encima.
7	Más grandes los botones de corazón y separados.
8	Le ha parecido bien.
9	Le ha parecido bien.
Extra	En la página principal poner un botón de inicio en la barra lateral, arriba de "Mis favoritos" para ir al inicio de la aplicación.

Tabla 4-3. Usuario 3 (22 años)

Tarea	**Opinión**
1	Se siente cómodo e incluido.
2	Le ha parecido muy simple y bien.
3	Los colores le parecen tristes, además de mucho espacio desaprovechado. Le falta información arriba de que tienes que hacer en cada paso.
4	Le falta información para los atributos a seleccionar.
5	No le parece que sea intuitivo los botones. Poco *feedback* visual.

6	Le parece bien, el corazón no tiene sentido en las playlists.
7	Le ha parecido bien.
8	Le ha parecido bien.
9	Le ha parecido bien.

Tabla 4-4. Usuario 4 (21 años)

En la tabla 4.5, se muestran los resultados de la sesión de evaluación una vez realizado algunos de los cambios acorde con las críticas constructivas de los anteriores participantes. Los cambios realizados sobre la aplicación fueron:

1. Colocación de un botón para registrarse en la página principal sin necesidad de meterse en "Iniciar sesión".
2. Explicación de las diferentes partes de las funcionalidades.
3. Añadir información de los botones en la funcionalidad "Crear playlist".
4. Cambio de botón de repetición en "Crear playlist".

Tarea	**Opinión**
1	Opina que el botón de registrarse debería estar al lado o arriba y los cuadrados cree que más grandes y el espacio más amplio. El color marrón no le gusta. El botón de atrás está mal.
2	Opina que debería haber un botón de atrás, pero le resulta bien el resto.
3	Le gustaría buscar canciones en lugar de seleccionar unas canciones que aparecen predeterminadas.
4	Le falta canciones para seleccionar aparte de las que te gustan.
5	Le parecen poco intuitivos los botones, sobre todo de siguiente y terminar playlists. Para él, el botón de siguientes sería más una cruz y el de terminar playlists sería más un botón de guardar.

6	Le parece muy bien, sólo el color de la interfaz le falla.
7	Le parece bien, funciona perfectamente.
8	Le parece intuitivo pero el icono le gustaría más grande.
9	Le parece bien.

Tabla 4-5. Usuario 5 (22 años)

Además de esta evaluación con un quinto usuario, realizamos una sesión de evaluación más reducida en la que obtuvimos un *feedback* acerca de las funcionalidades principales de la aplicación que consistía principalmente en añadir una opción de filtro para conseguir una recomendación más personalizada. Además de que debía tener la página una mejor consistencia al mostrar una lista de canciones.

4.6. Debriefing con los participantes y los observadores

Tras las sesiones de evaluación realizamos un pequeño resumen (*debriefing*) de las impresiones del usuario para completar la información recogida en el transcurso de la entrevista.

Después del *debriefing* compartiremos con todos los participantes un cuestionario en el que tendrán que expresar su grado de conformidad con las expresiones propuestas. El cuestionario tendrá preguntas generales, de la funcionalidad del sistema y de evaluación de la interfaz, y de manera opcional el usuario podrá dar su opinión de manera abierta sobre las partes que le han parecido más interesantes y las que menos.

Los resultados del cuestionario se ven reflejados en el gráfico de las figuras 4.1, 4.2 y 4.3.

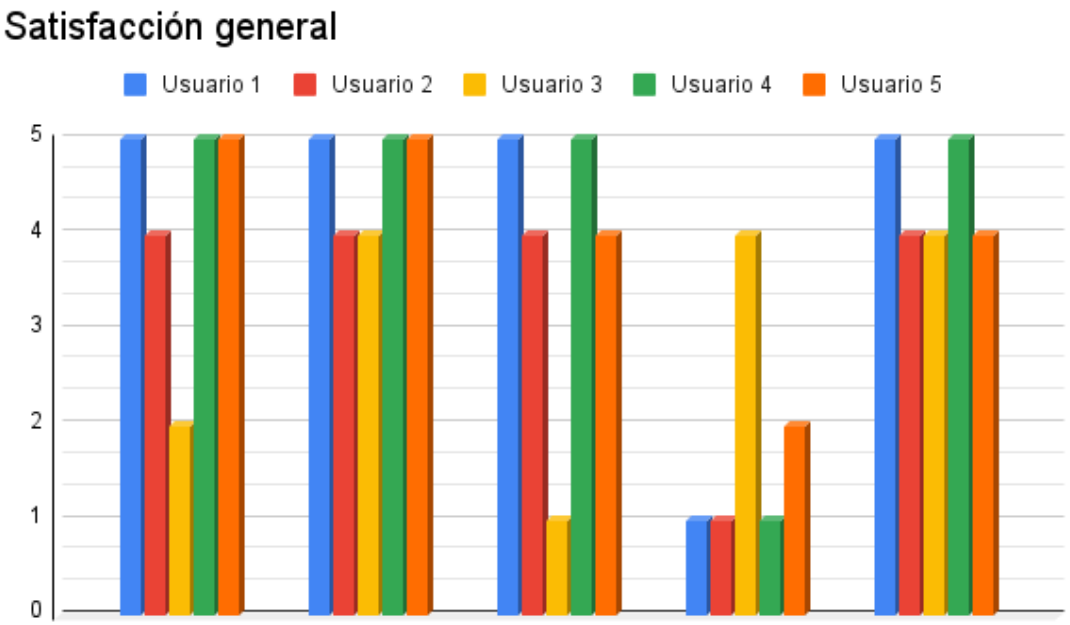

Figura 4-1. Resultados del cuestionario. Satisfacción general.

En esta sección se realizan preguntas acerca de aspectos generales de la aplicación. Las preguntas son:

1. En general, estoy satisfecho/a con la facilidad para utilizar el sistema.
2. Pude completar las tareas rápidamente.
3. Me sentí cómodo/a usando el sistema.
4. Necesitas saber bastantes cosas antes de poder empezar a usar la aplicación.
5. Considero que las funciones de la aplicación están bien integradas.

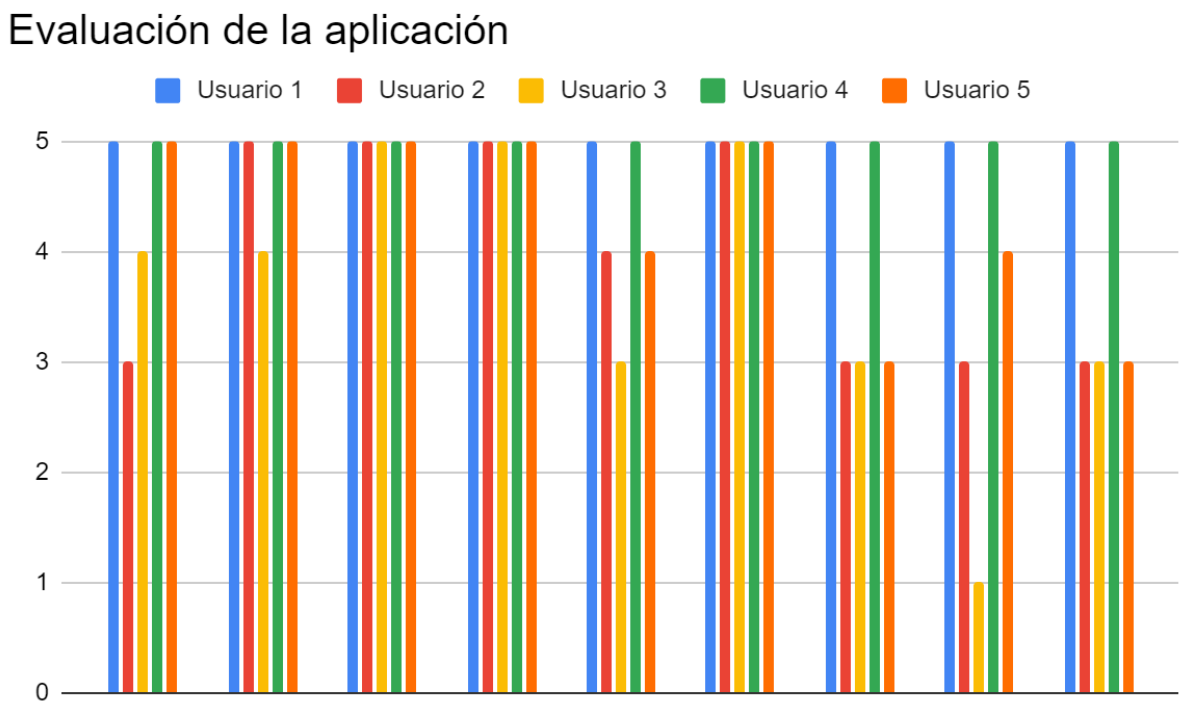

Figura 4-2. Resultados del cuestionario. Evaluación de la aplicación.

En esta sección se realizan preguntas acerca de las funcionalidades del sistema y del sistema de recomendación. Las preguntas son:

1. Me resulta útil la funcionalidad "Descubre nuevas canciones".
2. Me resulta útil la funcionalidad "Crear playlist".
3. Me resulta intuitiva la forma de acceder a "Mis favoritos".
4. Me resulta intuitiva la forma de acceder a "Mis playlists".
5. Me resulta fácil de usar "Descubre nuevas canciones".
6. Me resulta fácil de usar "Crear playlist".
7. Me resulta útil la información que muestra "Mi perfil".
8. Me resulta útil la información acerca de la recomendación.
9. Estoy de acuerdo con la recomendación dada según los criterios seleccionados.

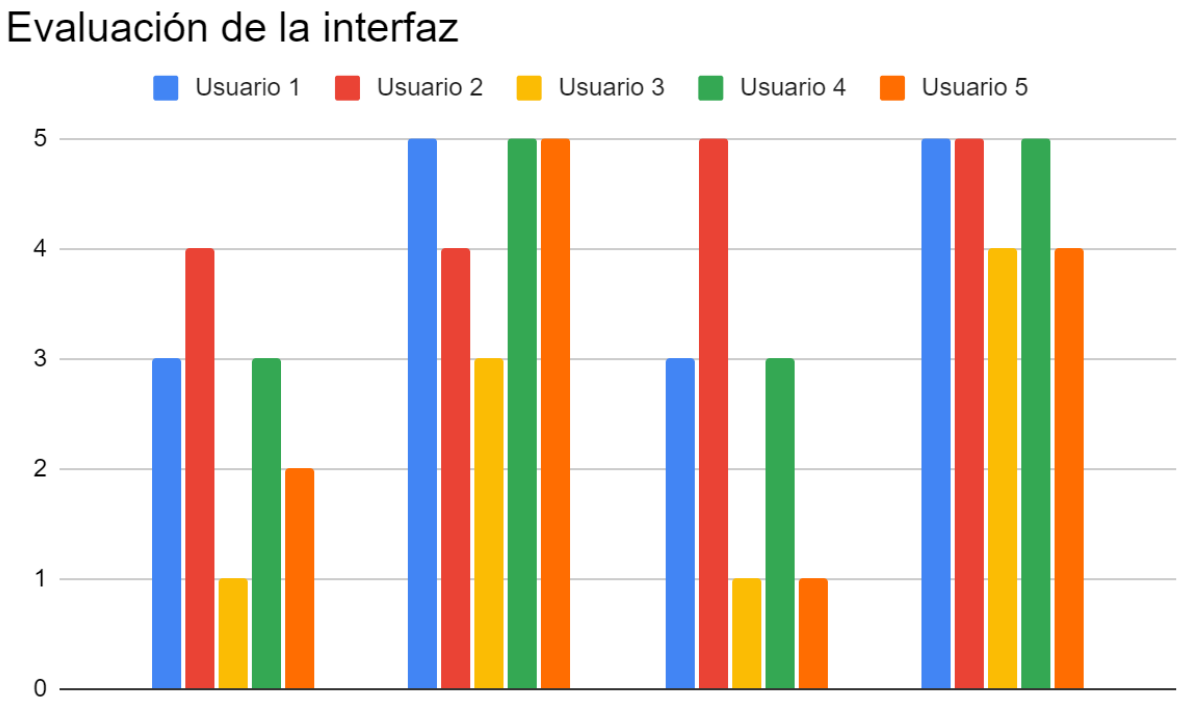

Figura 4-3. Resultados del cuestionario. Evaluación de la interfaz.

En esta sección se realizan preguntas acerca de la apariencia estética de la interfaz. Las preguntas son:

1. La forma de la interfaz me resulta agradable.
2. Me resulta fácil navegar por la interfaz.
3. Considero que los colores y el diseño son bonitos.
4. Considero que los colores permiten una buena visibilidad de la interfaz.

4.7. Análisis de los datos y las observaciones.

Una vez tenemos toda la información de las sesiones de evaluación y los cuestionarios completados pasamos a analizarlos con el fin de añadir todas las observaciones y resultados de los datos.

Respecto a las primeras sesiones de evaluación (sección 4.5) hemos observado que los usuarios han encontrado los siguientes problemas:

1. Falta de información y claridad en varias secciones de la interfaz, como en la funcionalidad de "Descubrir nuevas canciones", que en cada paso faltaba información acerca de que se pedía.
2. Falta de claridad en las explicaciones, han observado un mal formato en las explicaciones dadas en las canciones recomendadas.
3. Inconsistencia en los botones de las recomendaciones de "Crear playlist", confusión con algunos de ellos. Además, de faltar información de estos.
4. Falta de opciones de personalización y control, como la capacidad de refrescar la selección de canciones para recomendar.
5. Falta de funcionalidades básicas, como la posibilidad de registrar un nuevo usuario desde la página principal y la opción de iniciar sesión con el nombre usuario.
6. Selección de colores tristes.

Respecto al cuestionario pasado al final de la sesión de la evaluación se ha obtenido los siguientes datos:

1. Satisfacción general bastante positiva.
2. Evaluación de la aplicación en general bastante positiva a excepción de la explicación acerca de la recomendación.
3. Evaluación de la interfaz, resultados muy variados. Positivos acerca de la facilidad y visibilidad de los colores de la interfaz. Neutros acerca de los colores, como se nos comentó en las sesiones de evaluación por la tristeza que transmitía. Y negativos por la forma de la interfaz, cuyos fallos se han descrito en esta misma sección anteriormente.

En las evaluaciones que hicimos tras los cambios, que hemos comentado en la sección 4.5, no hemos detectado ningún fallo mayor, a excepción de nivel estético y la falta de personalización a la hora de elegir canciones, sobre todo en la funcionalidad de "Descubrir canciones".

4.8. Informe de hallazgos y recomendaciones.

Acorde a las tareas realizadas (sección 4.1) en las sesiones de evaluación (sección 4.5) y los problemas encontrados (sección 4.7), hemos concluido lo siguiente:

1. La falta de información y de claridad en las explicaciones puede afectar negativamente a la experiencia del usuario y puede llevar a malas decisiones de uso.
2. La inconsistencia en los botones de la recomendación puede generar confusión y frustración en los usuarios, y en el formato de la lista de canciones.
3. La falta de opciones de personalización y control (punto 4 de la sección 4.7) limita la utilidad y la satisfacción del usuario con el sistema.
4. La falta de funcionalidades básicas (punto 5 de la sección 4.7) dificulta el acceso por parte de nuevos usuarios.
5. La selección de colores tristes puede incomodar al usuario.
6. La falta de personalización a la hora de elegir una canción, a partir de la cual se recomendará.

Estos problemas son los que hemos tratado de resolver, aunque algunos se han incluido como trabajo futuro.

Capítulo 5 - Conclusiones y trabajo futuro

En este capítulo se presentan las conclusiones derivadas de los objetivos planteados, así como, las dificultades encontradas y las posibles mejoras del trabajo.

5.1. Conclusiones.

En este trabajo se ha desarrollado una aplicación con un sistema de recomendación híbrido personalizado con los objetivos planteados en la sección 1.2:

1. Objetivo 1: Conocer tecnologías y trabajos similares para ayudar a definir la funcionalidad del sistema.
2. Objetivo 2: Desarrollar un algoritmo de recomendación.
3. Objetivo 3: Desarrollar una interfaz web de usuario.
4. Objetivo 4: Sistema completo funcionando.
5. Objetivo 5: Experimentar con usuarios reales para validar el diseño propuesto.
6. Objetivo 6: Escribir y revisar la memoria.

Aunque se haya cumplido con la mayoría de los objetivos, se ha tenido cierta complicación con algunos de ellos. Las complicaciones han sido sobre todo en el desarrollo del sistema de recomendación en el que, bien se han estudiado *frameworks,* pero no se han aplicado, ya que hemos optado por implementarlo sin estos nos ha permitido un mayor entendimiento del funcionamiento de los sistemas de recomendación. Otro problema encontrado ha sido los datasets que no se pudo encontrar uno completo así que se tuvo que modificar, como se ha comentado en la sección 3.2, ya que faltaba una columna con la lista de varios géneros de la canción. Otro punto que no se ha podido cumplir ha sido el de usar datos muy grandes por falta de tiempo, así como el uso de datos en tiempo real como es la *API* de *Spotify*, esto se detalla más adelante en la sección 5.2.

En el desarrollo de la interfaz han surgido contratiempos con la base de datos, teniendo que haberse cambiado varias veces hasta la final, que se indica en la sección

2.4.4. Y tampoco se ha podido subir a un servidor online para ser probado públicamente, más detallado en la sección 5.2.

El resto de los objetivos y tareas se han podido realizar correctamente y sin problemas, por lo que nos han permitido aprender bastante sobre los sistemas de recomendación, las aplicaciones web y las formas de integrarlas, como son con los modelos MVC.

Cabe destacar que durante los primeros meses del desarrollo del proyecto las tareas se basaron principalmente en investigar e ir probando, sobre todo la parte del recomendador. En los siguientes meses ya se mejoró el recomendador desarrollado, como se puede ver en la sección 3.4, y se pudo poner en desarrollo la aplicación web con una interfaz mucho más visual e interactiva (sección 3.5).

5.2. Trabajo futuro.

Aunque el trabajo cumple con los objetivos planteados (sección 1.2), el proyecto admite mejoras para desarrollarlas en el futuro que le aportarían valor adicional.

A continuación, se exponen las propuestas de mejora:

1. Uso de librerías como "*Surprise*" para el recomendador.

   Sería interesante que, una vez conocido el funcionamiento de un algoritmo de recomendación, se pudieran introducir herramientas y algoritmos ya implementados para mejorar el rendimiento del recomendador.

2. Uso de la *API* de *Spotify*.

   El uso de un dataset limita la información con la que se puede operar. Sería conveniente poder conectar nuestra aplicación con la *API* de *Spotify* ya que contaremos con millones de canciones y el servicio ofrecido sería mucho más completo y actual.

3. Desplegar la aplicación en un entorno accesible al público.

   Por falta de tiempo no hemos podido subir la aplicación a un entorno de producción para que los usuarios puedan utilizar la web en sus propios

dispositivos. Los usuarios finales deberán tener accesibilidad a nuestra aplicación y esta sería una tarea fundamental para realizar en un futuro.

4. Añadir nuevas funcionalidades y mejorar la interfaz.

   El uso de la aplicación sólo en plataformas web limita mucho la utilidad de la aplicación, para permitir a los usuarios disfrutar de las recomendaciones se podría adaptar la web para el uso de dispositivos móviles.

   En cuanto a las funcionalidades, se puede agregar una barra de búsqueda para que el usuario busque directamente el nombre de la canción o el artista a partir del cual quiere recomendaciones. Así como la posibilidad de reproducir las canciones en la propia aplicación.

   También, sería importante mejorar la interfaz de usuario y hacerla más adaptativa para adaptarla a diferentes dispositivos.

Para concluir, se ha cumplido con todos los objetivos que se propusieron al principio del proyecto (sección 1.2) y hemos podido aprender acerca del funcionamiento interno de los sistemas de recomendación (sección 2.1), los diferentes tipos (sección 2.1.1) y algunos de los problemas que surgen en estos sistemas (secciones 2.1.2 y 2.1.3) y pese a haber desarrollado un algoritmo sin librerías ha resultado eficiente y fiable. Además, hemos podido aplicar los conocimientos adquiridos a lo largo de la carrera sobre las interfaces de usuario (sección 3.3), las aplicaciones web (sección 3.5) y aplicar metodologías software y patrones de diseño para el desarrollo de un proyecto (secciones 2.2 y 2.3). Y acorde con los resultados obtenidos en la fase de pruebas con usuarios (sección 4.7) se ha confirmado que se ha cumplido con los objetivos pese a algunos fallos detectados (sección 4.5).

Introduction

Motivation

Recommendation has existed as a natural process between people. For example: we have two friends who go to a restaurant and one of them is indecisive about the dish to order, the other will try to advise him based on his tastes and preferences, in order to make the final decision satisfy him. This is a very repeated action in everyday life that, with the arrival of the Internet, has been automated and is essential to solve the problem of information overload in the digital world.

Recommendation systems (Ricci et al. 2010) are tools that offer suggestions for useful items for the user, such as what to buy, what music to listen to or what book to read next, etc. These systems focus on an element or item and accordingly adapt the design of these, the interface and the recommendation technique or techniques to use to provide effective and useful suggestions. This tool is designed for people who do not have enough experience or time to evaluate the large number of options that are presented to them, serving as an aid to improve this experience.

It is a great challenge to be able to offer the user a complete musical catalog based on their preferences, and at the same time to find new musical records to complete the user's experience, that is why the idea of recommendation systems arose. Although this idea already exists and current recommendation systems are very precise, we find it very interesting to be able to replicate and learn more about the operation and implementation of these tools (Spotify AB 2024).

Today there are many platforms, such as *Netflix*, *Instagram,* and *Google*, that have implemented these systems. There are many reports, such as the one by the McKinsey consulting firm (Criado González 2018) which states that in 2018 35% of the purchases made by Amazon users came from recommendations and today.

The large amount of information that reaches us every second through the Internet can be overwhelming. On many occasions, it may happen that the information

received is not accurate or simply does not offer us the necessary information we are looking for. An example could be when we search in *Google*, the recommenders act in such a way that they show the user the most relevant results according to their profile, discarding those that are not outstanding at that time.

We have seen that recommenders are used for many areas, but in this work we focus on a specific recommendation domain: music, which is present in many areas of life and it is digital platforms such as Spotify, Amazon Music or YouTube that offer us most of this content.

On many occasions we have found that we always receive the same type of information, and taking it to the musical field, when interacting with the system (for example, *YouTube*) we are shown the same songs without giving us the opportunity to know many others that are probably to our liking.

Goals

To develop this project, we have proposed a series of objectives and tasks in order to progressively implement a recommendation system based on the data offered by Spotify, and to develop the application as a support to show the operation of our recommender with the different functionalities for the final user.

We have chosen to collect the information from Spotify because it is one of the largest music platforms that exist and has millions of songs (Spotify AB 2024), which can be useful for us to try to achieve maximum completeness of our recommendation algorithm. In addition, it has the content recommendation functionality, which serves as a reference for our work.

These objectives and tasks are:

1. **Objective 1**. Understand technologies and similar work to help define the functionality of the system.
    a. ***Task 1.1***. Reading of similar projects.
    b. **Task 1.2**. Search for datasets or platforms that publish their data.

   c. **Task 1.3.** Design of the software architecture: study suitable patterns for the application.
2. **Objective 2**. Develop a recommendation algorithm.
   a. ***Task 2.1***. Study the types of recommenders to determine the one that best suits the detected needs.
   b. ***Task 2.2***. Study existing frameworks and determine if they are applicable in our design.
   c. ***Task 2.3***. Understand the data provided by the dataset.
   d. ***Task 2.4***. Implement a simple recommender and test it in a prototype with little data to check its operation.
   e. ***Task 2.5***. Test different types of recommenders in a prototype with little data.
   f. ***Task 2.6***. Scale the recommender to large data and/or connection with real-time data.
   g. ***Task 2.7***. Verify that the data to be used and the recommender work correctly.
3. **Objective 3**. Develop a user web interface.
   a. ***Task 3.1***. Design the interface prototypes.
   b. ***Task 3.2***. Develop the web interface.
   c. ***Task 3.3***. Implement the interface logic.
   d. ***Task 3.4***. Perform usability tests to verify its correct operation.
4. **Objective 4**. Complete system working.
   a. ***Task 4.1***. Integrate the parts: Recommender - Interface.
   b. ***Task 4.2***. Resolve the integration between the parts.
5. **Objective 5**. Experiment with real users to validate the proposed design.
   a. ***Task 5.1***. Create test users to test the operation.
   b. ***Task 5.2***. Integrate potential users of the application into the system.
6. **Objective 6**. Write and review the memory.
   a. ***Task 6.1***. Sequential development of the memory.
   b. ***Task 6.2***. Review of the completeness and cohesion of the memory.

Work plan

To establish the work plan, we have considered the research, development, and delivery times for each part of our project. Throughout the process, we have followed an agile development based on the Scrum methodology (see section 2.3) with small deliveries in short periods of time (generally, two weeks), adapting the needs that may arise to these time objectives. We have also held regular meetings to review progress, quality, and share the work to be done in the future. To track this entire process, we have used the Excel tool with the different tasks to be performed in the sprints and the delivery times.

Research was the first task carried out, in order to familiarize ourselves with the concepts and know the different tools available for the subsequent implementation of our project. This research has been carried out throughout the project development process in order to improve the consistency and performance of the work.

After carrying out the research phase and acquiring the necessary knowledge, we have continued with the development phase. In this phase, we have implemented the recommendation algorithm and the web application, and it has served us to put into practice everything studied previously.

During the evolution of the work, we have alternated the two previously mentioned phases, research and development, with the task of writing the project documentation. We have also been correcting the errors or changes that have occurred during the same so that the information presented here is updated and in accordance with the work carried out.

Conclusions and future work

"Conclusiones y trabajo futuro" section translated to English.

Conclusions

In this work, an application with a personalized hybrid recommendation system has been developed with the objectives outlined in section 1.2:

1. Objective 1: Understand technologies and similar work to help define the functionality of the system.
2. Objective 2: Develop a recommendation algorithm.
3. Objective 3: Develop a user web interface.
4. Objective 4: Complete system working.
5. Objective 5: Experiment with real users to validate the proposed design.
6. Objective 6: Write and review the memory.

Although most of the objectives have been met, there have been some complications with some of them. The complications have been mainly in the development of the recommendation system in which, although frameworks have been studied, they have not been applied, since we have chosen to implement it without them, which has allowed us to better understand the operation of recommendation systems. Another problem encountered has been the datasets, since a complete one could not be found, so it had to be modified, as commented in section 3.2, since a column with the list of various genres of the song was missing. Another point that could not be fulfilled has been the use of very large data due to lack of time, as well as the use of real-time data such as the Spotify API, which is detailed later in section 5.2.

In the development of the interface, there have been setbacks with the database, having to have been changed several times until the final one, which is indicated in section 2.4.4. And it has also not been possible to upload to an online server to be publicly tested, more detailed in section 5.2.

The rest of the objectives and tasks have been carried out correctly and without problems, which has allowed us to learn a lot about recommendation systems, web applications and ways to integrate them, such as with MVC models.

It should be noted that during the first months of project development, the tasks were mainly based on research and testing, especially the recommender part. In the following months, the developed recommender was improved, as can be seen in section 3.4, and the web application could be developed with a much more visual and interactive interface (section 3.5).

Future Work

Although the work meets the objectives set out (section 1.2), the project admits improvements to be developed in the future that would provide it with additional value.

The following are the improvement proposals:

1. **Use of libraries like "Surprise" for the recommender.**

   It would be interesting that, once the operation of a recommendation algorithm is known, it would be possible to introduce already implemented tools and algorithms to improve the performance of the recommender.

2. **Use of the Spotify API.**

   The use of a dataset limits the information with which you can operate. It would be convenient to be able to connect our application with the Spotify API since we would have millions of songs and the service offered would be much more complete and up-to-date.

3. **Deploy the application in a public environment.**

   Due to lack of time, we have not been able to upload the application to a production environment so that users can use the web on their own devices. End users should have accessibility to our application, and this would be a fundamental task to carry out in the future.

4. **Add new features and improve the interface.**

The use of the application only on web platforms greatly limits the usefulness of the application. To allow users to enjoy the recommendations, the web could be adapted for use on mobile devices.

As for the functionalities, a search bar can be added so that the user can directly search for the name of the song or the artist from which they want recommendations. As well as the possibility of playing the songs in the application itself. Also, it would be important to improve the user interface and make it more adaptive to adapt it to different devices.

In conclusion, all the objectives that were proposed at the beginning of the project have been met (section 1.2) and we have been able to learn about the internal workings of recommendation systems (section 2.1), the different types (section 2.1.1) and some of the problems that arise in these systems (sections 2.1.2 and 2.1.3) and despite having developed an algorithm without libraries, it has proven efficient and reliable. In addition, we have been able to apply the knowledge acquired throughout the career on user interfaces (section 3.3), web applications (section 3.5) and apply software methodologies and design patterns for the development of a project (sections 2.2 and 2.3). And according to the results obtained in the user testing phase (section 4.7), it has been confirmed that the objectives have been met despite some detected failures (section 4.5).

CONTRIBUCIONES PERSONALES

En este capítulo se explicará en detalle el trabajo realizado por cada uno de los miembros durante el desarrollo del proyecto. Estas tareas se han realizado tanto de forma individual como en grupo para conseguir el objetivo propuesto.

Leire Jiménez González.

Al inicio de este TFG, he estudiado la viabilidad del proyecto junto a mi compañera, buscando información sobre sistemas de recomendación, definiendo los objetivos del proyecto, investigando las diferentes herramientas disponibles para usar y aportando ideas de diseño e implementación para el posterior desarrollo.

Una vez comprobada la viabilidad del proyecto, resultando positivo, diseñamos en grupo un boceto de la interfaz de la aplicación mediante la herramienta Figma. Durante este proceso de diseño, aporté ideas de estilo y diseño general, así como definir el logo.

Al contar con dos módulos de implementación en el proyecto (ver secciones 3.4 y 3.5), me he centrado en el desarrollo de la aplicación. Me he apoyado en el Framework Django para realizar la aplicación porque consideré que era la opción más sencilla para la posterior integración del algoritmo de recomendación programado en Python realizado por mi compañera. A continuación, enumeraré las aportaciones individuales dentro de la parte de implementación de la aplicación:

1. Creación de un proyecto Django y modificación de ficheros para adaptarlos a nuestras necesidades.
2. Importación de librerías y módulos necesarios para el funcionamiento del proyecto.
3. Definir la base de datos en SQLite y posterior cambio a PostgreSQL. Así como, crear las tablas en el fichero models.py, introducir la información necesaria y comprobar su correcto funcionamiento.
4. Desarrollo de la aplicación mediante HTML, CSS, JavaScript y Python.

5. Creación de dos módulos dentro del proyecto: "paginaPrincipal" y "usuarios", para diferenciar entre las vistas generales y las de usuario.
6. Implementación de las funcionalidades de "Descubrir canciones" y "Crear Playlist".
7. Implementación de las funcionalidades básicas de la aplicación en el módulo "usuarios": inicio de sesión, registro y perfil de usuario siguiendo los bocetos propuestos.
8. Implementación de funcionalidades concretas para nuestro proyecto en el módulo "paginaPrincipal": "Mis playlists" y "Mis favoritos".
9. Realizar la conexión con la base de datos para recoger los datos necesarios en las funcionalidades que lo requerían. Esto lo he realizado en el fichero views.py del módulo correspondiente.
10. Integración del algoritmo de recomendación con la aplicación.
11. Comprobación unitaria del funcionamiento de las diferentes funcionalidades.

En cuanto a la evaluación de usuarios, en colaboración con mi compañera preparé el plan de evaluación y participé en las diferentes sesiones de evaluación guiando a los participantes durante todo el proceso y analizando los datos posteriormente recabados durante las entrevistas y los cuestionarios.

Con respecto a la memoria, he contribuido a la redacción de diferentes secciones: el Capítulo 1, donde se realiza una introducción sobre nuestro proyecto, junto a la motivación, objetivos propuestos y plan de trabajo; el Capítulo 2, donde expliqué la arquitectura software, la metodología y las tecnologías usadas; el Capítulo 3, realicé la explicación del diseño de la interfaz de usuario y el desarrollo de la aplicación; el Capítulo 4, donde desarrollé parte de la explicación de la evaluación con usuarios; y el Capítulo 5, donde se detallan las conclusiones y trabajo futuro.

Laura Martínez Tomás.

A lo largo del desarrollo del TFG he contribuido a diferentes partes del proyecto. Lo primero a lo que contribuí fue en el boceto inicial del diseño de la interfaz mediante la herramienta de Figma.

Respecto a la parte de la implementación del proyecto (ver secciones 3.4 y 3.5), he desarrollado el algoritmo para un sistema de recomendación híbrido (sección 3.4.4), tras haber realizado distintas versiones. Para poder implementar bien esta parte he investigado acerca de los distintos recomendadores, finalmente decantándome por un modelo híbrido que combina las ventajas de los sistemas de recomendación basado en contenido (sección 2.1.1.2) y de filtrado colaborativo (sección 2.1.1.1). Este algoritmo lo desarrollé en Python sin ningún uso de librerías hechas para implementar sistemas de recomendación como *Surprise*. Además, para poder ver el funcionamiento del algoritmo necesitaba un conjunto de datos o dataset, que tras una larga búsqueda encontré uno, que luego tuve que modificar con un algoritmo que yo misma creé en C++ para añadir una columna de la lista de géneros aleatorios para cada canción. Asimismo, tuve que reducir el dataset con otro algoritmo para dejarlo en 10000 canciones, ya que el original contenía un millón de canciones y no me permitía observar bien el funcionamiento del recomendador.

Cuando tuvimos una versión funcional tanto para recomendador como página planteamos una evaluación de usuarios que junto con mi compañera preparé el plan de evaluación y participé en las diferentes sesiones de evaluación tomando notas acerca de los comentarios y comportamientos de los participantes durante todo el proceso y analizando los datos posteriormente tanto de las cuestiones como de las sesiones de evaluación y realicé la conclusión acerca de los fallos más comunes.

Con respecto a la memoria, he contribuido a la redacción de diferentes secciones: el Capítulo 1, parte de la introducción sobre nuestro proyecto junto a parte de la motivación y he contribuido al desarrollo del plan de trabajo; el Capítulo 2, donde introduje el capítulo y expliqué toda la sección de los fundamentos de los sistemas de recomendación, además de los lenguajes en la sección de tecnologías. En el Capítulo 3 realicé la explicación de la metodología, la descripción de los conjuntos de datos y la descripción de mi parte de la implementación, el desarrollo del algoritmo de recomendación. En el Capítulo 4 desarrollé junto a mi compañera la preparación del plan de evaluación y luego describí los siguientes cuatro apartados. Como mi rol en las sesiones de evaluación era recolectar la información de los participantes, lo escribí en

las tablas. Y finalmente, en el Capítulo 4, realicé el análisis de los datos y el informe de hallazgos. En el Capítulo 5, detallé las conclusiones.

BIBLIOGRAFÍA

Aggarwal, C. C. (2016). Recommender Systems: The Textbook. Springer International Publishing.

Attardi, J. (2020). Modern CSS: Master the Key Concepts of CSS for Modern Web Development. Apress.

Ben-Ari, M. (1996). Understanding Programming Languages. Wiley.

Berrar, D. (2018). Cross-validation. Encyclopedia of Bioinformatics and Computational Biology, 1, 542-545. https://doi.org/10.1016/B978-0-12-809633-8.20349-X

Burke, R. (2002, Enero). Hybrid Recommender Systems: Survey and Experiments. User Modeling and User-Adapted Interaction, 12(4), 331-370. 10.1023/A:1021240730564

Canós, J. H., Letelier, P., & Penadés, M. C. (2003). Métodologías Ágiles en el Desarrollo de Software [Taller realizado en el marco de las VIII Jornadas de Ingeniería del Software y Bases de Datos, JISBD 2003]. Universidad Politécnica de Valencia. https://issi.dsic.upv.es/archives/f-1069167248521/actas.pdf

Caro Martínez, M. (2022). Sistemas de recomendación y explicaciones basados en grafos de interacción [Tesis inédita de la Universidad Complutense de Madrid, Facultad de Informática, Departamento de Ingeniería de Software e Inteligencia Artificial]. Universidad Complutense de Madrid Facultad de Informática. https://docta.ucm.es/entities/publication/b565c024-4fe7-4d2e-9038-735ebe1bcfd3

Castro, L. M. (2012). Arquitectura Software [Apuntes]. Mads Group. https://upload.wikimedia.org/wikipedia/commons/b/be/AS1-arquitecturas-no-distribuidas.pdf

Criado González, M. (2018). Análisis e implementación de un sistema de recomendación para la lista de la compra [Trabajo Fin de Grado]. https://core.ac.uk/download/pdf/288501906.pdf

Documentación de GitHub. (2017, November 9). Documentación de GitHub. Retrieved April 2, 2024, from https://docs.github.com/es/get-started/start-your-journey/about-github-and-git

Documentation for Visual Studio Code. (2024). Visual Studio Code. Retrieved April 2, 2024, from https://code.visualstudio.com/docs

Elliott, E. (2014). Programming JavaScript Applications. O'Reilly.

Equipo de datos.gob.es. (2022, May 3). 11 librerías para crear visualizaciones de datos | datos.gob.es. Datos.gob.es. Retrieved March 31, 2024, from https://datos.gob.es/es/blog/11-librerias-para-crear-visualizaciones-de-datos

Esteban Gabriel, M. (2015). Metodologías de desarrollo de software [Apuntes]. Facultad De Química E Ingeniería "Fray Rogelio Bacon". https://repositorio.uca.edu.ar/bitstream/123456789/522/1/metodologias-desarrollo-software.pdf

Felfernig, A., & Burke, R. (2008). Constraint-based Recommender Systems: Technologies and Research Issues. In Proceedings of the 10th International Conference on Electronic Commerce: 2008, Innsbruck, Austria, August 19-22, 2008. ACM Press.

Fernández Romero, Y., & Díaz González, Y. (2012, Enero-abril Enero-abril). Patrón Modelo-Vista-Controlador. Revista Digital de las Tecnologías de la Información y las Comunicaciones, 11(1), 47-57. ISSN 1729-3804

Forcier, J., Bissex, P., & Chun, W. (2009). Python Web Development with Django. Addison-Wesley.

Francisco Gilmartín, V. (2023). Tema 9: Evaluación [Apuntes de la asignatura Desarrollo de Sistemas Interactivos].

Gabbrielli, M., & Martini, S. (2010). Programming Languages: Principles and Paradigms. Springer.

Garlan, D., & Shaw, M. (1994). An Introduction to Software Architecture. 10.1184/R1/6603365.v1

Gilbert Ginestà, M., & Pérez Mora, O. (2014). Bases de datos en PostgreSQL. P06/M2109/02152

Gope, J., & Jain, S. K. (2017). A survey on solving cold start problem in recommender systems. 2017 International Conference on Computing, Communication and Automation (ICCCA), 133-138. 10.1109/CCAA.2017.8229786

Ho, T.-L., Le, A.-C., & Vu, D.-H. (2023). Multiview Fusion Using Transformer Model for Recommender Systems: Integrating the Utility Matrix and Textual Sources. Applied Sciences, 13(10). 10.3390/app13106324

Hug, N. (2015). similarities module — Surprise 1 documentation. Surprise' documentation! Retrieved March 10, 2024, from https://surprise.readthedocs.io/en/stable/similarities.html

Jannach, D., Zanker, M., Felfernig, A., & Friedrich, G. (2011). Recommender Systems: An Introduction. Cambridge University Press.

Jiménez Torres, V. H., Tello Borja, W., & Ríos Patiño, J. I. (2014). Lenguajes de Patrones de Arquitectura de Software: Una Aproximación Al Estado del Arte. Scientia et Technica, 19(4), 371-376. 10.22517/23447214.8595

Johnson, R. E. (1997). Frameworks= (Components+Patterns). Communications of the ACM, 40(10), 39-42.

Joshi, A., Parolkar, A., & Das, V. (2023). Spotify 1 million tracks dataset. Kaggle. Retrieved March 10, 2024, from https://www.kaggle.com/datasets/amitanshjoshi/spotify-1million-tracks

Jupyter Team. (2015). Jupyter. Project Jupyter Documentation — Jupyter Documentation 4.1.1 alpha documentation. Retrieved April 2, 2024, from https://docs.jupyter.org/en/latest/

Ko, H., Lee, S., Park, Y., & Choi, A. (2022). A Survey of Recommendation Systems: Recommendation Models, Techniques, and Application Fields. Electronics, 11(1), 141. 10.3390/electronics11010141

Mana, S. C., & T.Sasipraba. (2021). Research on Cosine Similarity and Pearson Correlation Based Recommendation Models. Journal of Physics: Conference Series, 1770(1), 012014. 10.1088/1742-6596/1770/1/012014

Marqués Andrés, M. (2011). Bases de datos. Publicacions de la Universitat Jaume I.

Martínez Villalobos, G., Camacho Sánchez, G. D., & Biancha Gutiérrez, D. A. (2010). Diseño de Framework web para el desarrollo dinámico de aplicaciones. Scientia Et Technica, 16(44), 178-183. 0122-1701

MDN contributors. (2023, November 13). MVC - Glosario de MDN Web Docs: Definiciones de términos relacionados con la Web | MDN. MDN Web Docs. Retrieved April 2, 2024, from https://developer.mozilla.org/es/docs/Glossary/MVC

Musciano, C., & Kennedy, B. (2002). HTML and XHTML, the definitive guide. O'Reilly Media.

Navarro Cadavid, A., Fernández Martínez, J. D., & Morales Vélez, J. (2013). Revisión de metodologías ágiles para el desarrollo de software. PROSPECTIVA, 11(2), 30-39. ISSN 1692-8261

NumPy Developers. (2022). What is NumPy? — NumPy v1.26 Manual. NumPy -. Retrieved March 31, 2024, from https://numpy.org/doc/stable/user/whatisnumpy.html

Núñez Gómez, S. O. (2022, Enero 18). Sistemas de recomendación de canciones. Universitat Politècnica de Catalunya Barcelonatech Facultat d'Informàtica de Barcelona.
https://upcommons.upc.edu/bitstream/handle/2117/362448/164343.pdf?sequence=1

pandas documentation — pandas 2.2.1 documentation. (2024, February 23). Pandas. Retrieved March 31, 2024, from https://pandas.pydata.org/docs/

Pantoja, L., & Pardo, C. (2016). Evaluando la facilidad de aprendizaje de frameworks MVC en el desarrollo de aplicaciones web. Publicaciones e Investigación, 10. 1900-6608

Park, Y.-J., & Tuzhilin, A. (2008). The long tail of recommender systems and how to leverage it. Proceedings of the 2008 ACM Conference on Recommender Systems, 11-18. 10.1145/1454008.1454012

Pérez Ibarra, S. G., Quispe, J. R., Mullicundo, F. F., & Lamas, D. A. (2021). Herramientas y tecnologías para el desarrollo web desde el Front-end al Back-end. In XXIII Workshop de Investigadores en Ciencias de la Computación (WICC 2021, Chilecito, La Rioja). Red de Universidades con Carreras en Informática. http://sedici.unlp.edu.ar/handle/10915/120476

The PostgreSQL Global Development Group. (2024). PostgreSQL. PostgreSQL: The world's most advanced open source database. Retrieved April 2, 2024, from https://www.postgresql.org/

Priestley, M. (2011). A Science of Operations: Machines, Logic and the Invention of Programming. Springer.

Ramón Sanchis, D. A. (2023). AcWeb: una aplicación web para detectar copias en prácticas de programación. [Trabajo de Fin de Grado Curso 2022–2023]. https://docta.ucm.es/rest/api/core/bitstreams/e9d6b73e-33f9-4cc0-be82-3a11d382dd0a/content

Reenskaug, T. (2003). The Model-View-Controller (MVC) Its Past and Present [Presentation]. https://nandgatetech.com/files/ComputerHistory/MVC_pattern.pdf

Ricci, F., Rokach, L., & Shapira, B. (Eds.). (2022). Recommender Systems Handbook. Springer US.

Ricci, F., Rokach, L., Shapira, B., & Kantor, P. B. (Eds.). (2010). Recommender Systems Handbook. Springer.

Rodríguez Rodríguez, J. E., Rojas Blanco, E. A., & Franco Camacho, R. O. (2007). Clasifi cación de datos usando el método k-nn. Vínculos, 4(1), 4-18. 10.14483/2322939X.4111

Schwaber, K., & Sutherland, J. (2020). La Guía Scrum [La Guía Definitiva de Scrum: Las Reglas del Juego]. https://objetivoscrum.com/wp-content/uploads/2021/01/2020-Scrum-Guide-Spanish-European-2.0_objetivoScrum.pdf

scikit-learn developers. (n.d.). Getting Started — scikit-learn 1.4.1 documentation. Scikit-learn. Retrieved March 31, 2024, from https://scikit-learn.org/stable/getting_started.html

SciPy Steering Council. (2024, January 20). SciPy. SciPy -. Retrieved March 31, 2024, from https://scipy.org/

Singh, R. H., Maurya, S., Tripathi, T., Narula, T., & Srivastav, G. (2020). Movie Recommendation System using Cosine Similarity and KNN. International Journal of Engineering and Advanced Technology (IJEAT), 9(5). 10.35940/ijeat.E9666.069520

Sinha, A., Ranjan, A., & Battewad, R. (2020). JavaScript for Modern Web Development: Building a Web Application Using Html, Css, and JavaScript (English Edition). Bpb Publications.

Son, J., & Kim, S. B. (2017). Content-based filtering for recommendation systems using multiattribute networks. Expert Systems with Applications, 89, 404-412. 10.1016/j.eswa.2017.08.008

Sondur, S. D., Chigadani, A. P., & Nayak, S. (2016). Similarity Measures for Recommender Systems: A Comparative Study. Journal for Research, 2(3).

Spotify AB. (2024). ¿Qué es Spotify? - Spotify. Spotify Support. Retrieved May 1, 2024, from https://support.spotify.com/es/article/what-is-spotify/

Srinath, K. R. (2017). Python - The Fastest Growing Programming Language. International Research Journal of Engineering and Technology (IRJET), 4(12).

Tintarev, N., & Masthoff, J. (2007, Abril). A Survey of Explanations in Recommender Systems. 2007 IEEE 23rd International Conference on Data Engineering Workshop, 801-810. 10.1109/ICDEW.2007.4401070

Touriño Calvo, D. (2020, junio). Recomendación de canciones y listas de reproducción sobre Spotify. Facultade de Informática Universidade Da Coruña.

https://ruc.udc.es/dspace/bitstream/handle/2183/26156/D.Touri%c3%b1o_Calvo_2020_Recomendacion_de_canciones_y_listas_de_reproduccion_sobre_Spotify.pdf?sequence=3&isAllowed=y

Trigas Gallego, M. (2012). Metodología Scrum [Desarrollo detallado de la fase de aprobación de un proyecto informático mediante el uso de metodologías ágiles].

Vidal-Silva, C. L., Sánchez-Ortiz, A., Serrano, J., & Rubio, J. M. (2021). Experiencia académica en desarrollo rápido de sistemas de información web con Python y Django. Formación universitaria, 14(5). 10.4067/S0718-50062021000500085

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R.J., Jones, E., Kern, R., Larson, E., … SciPy 1.0 Contributors. (2020). SciPy 1.0: fundamental algorithms for scientific computing in Python. Nature Methods, 17, 261-272. https://doi.org/10.1038/s41592-019-0686-2

Voorhees, D. P. (2020). Guide to Efficient Software Design: An MVC Approach to Concepts, Structures, and Models. Springer International Publishing.

Zhang, Y., & Chen, X. (2020). Explainable Recommendation: A Survey and New Perspectives. Foundations and Trends® in Information Retrieval, 14(1), 1-101. 10.1561/1500000066

Zumba Gamboa, J. P., & León Arreaga, C. A. (2018). Evolución de las Metodologías y Modelos utilizados en el Desarrollo de Software. INNOVA Research Journal, 3(10), 20-33. ISSN 2477-90