uc3m | Universidad Carlos III de Madrid

Grado Universitario en Ingeniería Informática (2022-2023)

Trabajo Fin de Grado

"Desarrollo de una aplicación web usando la API de Spotify"

Marcos Terroso Prieto

Tutor:

Alejandro Rey López

Leganés, septiembre de 2023

RESUMEN

La aplicación Spotify es una de las aplicaciones de música en streaming más utilizadas. Entre otras muchas funcionalidades, Spotify genera listas de reproducción, o *playlists,* para todos los usuarios, como pueden ser listas de éxitos por países, y también *playlists* personalizadas, en función de las preferencias musicales de cada usuario. Una de las playlists que genera Spotify para sus usuarios es "Radar de novedades". En ella hay 30 canciones publicadas recientemente de artistas a los que el usuario sigue y que suele escuchar, además de recomendaciones según tus gustos. Esta playlist se actualiza cada semana por lo que las canciones anteriores son reemplazadas por otras nuevas.

El objetivo de este trabajo es crear una aplicación web que sea capaz de almacenar las canciones de la *playlist* Radar de Novedades, de forma que los usuarios puedan acceder al histórico de esta playlist y no pierdan potenciales canciones de su interés por no haberlas guardado manualmente. Para ello, se utilizará la API (Application Programming Interface) de Spotify para obtener la información de la *playlist,* que será guardada en una base de datos. Las canciones almacenadas se podrán visualizar desde la aplicación, y crear las playlists de forma permanente en la cuenta de Spotify del usuario.

Palabras clave: Spotify, streaming, playlist, API, base de datos

ABSTRACT

The Spotify application is one of the most widely used music streaming services worldwide. Among many other features, Spotify generates playlists for all users, such as lists of hits by country. It also creates personalized playlists, which depend on the musical preferences of each user. One of the playlists that Spotify generates for its users is "Release Radar". It has 30 recently published songs from artists that the user follows and usually listens to, as well as recommendations according to your tastes. This playlist updates its songs every week.

The objective of this project is to develop a web application that allows programmatically storing the songs of “Release Radar” playlist, in order to allow the user revisit songs that may be lost if the user never manually liked or saved them before the playlist update. To achieve this, the API (Application Programming Interface) provided by Spotify will be used, as it allows to obtain the playlist information, which will be stored in a database. The stored songs can be viewed from the application, and playlists can be created permanently in the user's Spotify account.

Key words: Spotify, streaming, playlist, API, database

ÍNDICE

1. INTRODUCCIÓN

1.1. Contexto y motivación

Actualmente, una de las formas más populares de escuchar y descubrir música es a través de las plataformas de streaming. Con estas plataformas, los usuarios disponen de un inmenso catálogo de música de forma gratuita o con un coste reducido desde cualquier lugar.

Una de las plataformas más importantes en este ámbito es Spotify, en la que nos enfocamos en este trabajo.

Spotify y sus alternativas han cambiado la forma en la que accedemos a música, descubrimos artistas o podcasts; y nos permite organizar y ampliar nuestra biblioteca personal a través de diversas funcionalidades que ofrece, entre las que se incluye la creación de listas de reproducción, conocidas como "*playlists*". Estas listas pueden ser generales, como las listas de éxitos por países; pero también personalizadas, habiendo sido creadas por los propios usuarios o generadas por Spotify basado en las preferencias musicales de cada cual.

Una de las playlists que Spotify proporciona a sus usuarios se llama "Radar de novedades". Esta incluye 30 canciones publicadas recientemente por artistas a los que el usuario sigue y escucha con frecuencia, además de recomendaciones de otros artistas basadas en sus gustos. La lista de reproducción se actualiza semanalmente, concretamente los viernes de cada semana, lo que implica que las canciones sólo están disponibles en esa lista durante 7 días.

Dado que Spotify no permite guardar esta playlist autogenerada, sino que la sobreescribe, los usuarios pierden el acceso a las canciones a menos que las hayan guardado manualmente o le hayan dado a “Me gusta” a todas las canciones en la lista de reproducción.

Es por ello que el presente Trabajo Fin de Grado pretende proporcionar una solución a esta limitación de Spotify, permitiendo a los usuarios persistir en el tiempo estas *playlists*, para poder visualizarlas y consultarlas en cualquier momento.

1.2. Objetivos

El objetivo primordial de este trabajo es diseñar y desarrollar una aplicación web que permita almacenar las canciones de la playlist de Spotify “Radar de Novedades” para cada usuario, utilizando para ello los datos que la propia plataforma Spotify proporciona a través de su API.

Para la obtención de este objetivo, se establecen los siguientes objetivos secundarios:

- Creación de una interfaz web sencilla que permita al usuario realizar todas las funcionalidades necesarias.
- Creación de un servidor, que se conecta con la base de datos, con la interfaz web, y que realiza las llamadas a la API de Spotify.
- Autenticación del usuario en la aplicación, para verificar la identidad del usuario conectado.
- Concesión de permisos de Spotify, la cual solo se solicita una vez, ya que la información se asocia al usuario de la aplicación registrado previamente.
- Permitir la visualización de los datos obtenidos de las playlist al usuario en la interfaz web.
- Permitir que el usuario pueda crear una playlist en su cuenta de Spotify, usando también las llamadas de la API.

1.3. Estructura del documento

Este documento, además de la portada, el resumen y los índices contiene los siguientes capítulos:

- **Capítulo 1**: Cuenta con una introducción, explicando la motivación para su realización, los objetivos que se quieren completar, y la estructura del documento.
- **Capítulo 2**: Se hace referencia a otros **trabajos relacionados**. Tras ello, se analizan las posibles **tecnologías** que podrían usarse para la realización del proyecto, finalizando con la **justificación de las herramientas elegidas**.
- **Capítulo 3**: Se encuentra el **análisis** del proyecto, donde se detallan los casos de uso, y se introducen tanto los requisitos de usuario, como los requisitos de software, concluyendo con la matriz de trazabilidad.
- **Capítulo 4**: Se detalla la **implementación** del proyecto, siguiendo el proceso lógico que se ha seguido en el desarrollo de la aplicación, mostrando la relación que tienen entre sí todos los componentes de la aplicación.
- **Capítulo 5**: Se realizan las **pruebas** necesarias para verificar que la aplicación tiene el funcionamiento esperado.
- **Capítulo 6**: Se muestra la **planificación** del proyecto mediante un diagrama de Gantt, además de un **presupuesto** del proyecto, incluyendo costes en materiales, software, y costes humanos.
- **Capítulo 7**: Se analiza el **marco regulador**, analizando la legislación aplicable sobre la aplicación.
- **Capítulo 8**: Se analiza el **impacto social y económico que** tiene la aplicación.
- **Capítulo 9**: En este capítulo final, se verifica que los objetivos iniciales se cumplen, y se introducen las posibles características que está previsto que sean desarrolladas para mejorar y completar la aplicación, finalizando con las **conclusiones** personales del autor.

2. ESTADO DEL ARTE

En este capítulo se define el contexto tecnológico que rodea al proyecto. Se procede a analizar las diferentes tecnologías y herramientas que existen en la actualidad que permiten desarrollar una aplicación. Una vez conocidas las posibles tecnologías que se pueden utilizar, se enuncian otros proyectos con una temática similar a la del presente proyecto. Finalmente, una vez hecho este estudio, se justifica la elección de las herramientas utilizadas en este proyecto.

2.1. Tecnologías para el desarrollo web

Existen dos vertientes principales de desarrollo en este tipo de aplicaciones: frontend y backend.

El frontend hace referencia a la parte visual de la aplicación, todo el contenido de la web con el que el usuario interactúa directamente. Esto incluye todo lo que un usuario puede visualizar, incluyendo textos, imágenes, colores o botones. Esta parte de la aplicación está en el lado del cliente [1].

Por otra parte, el backend es la parte que el usuario no ve y con la que no interactúa directamente. El backend se encarga de gestionar la lógica que hay por debajo de la aplicación para que todo funcione correctamente, como las conexiones con bases de datos, o con APIs externas. Está parte de la aplicación está en el lado del servidor. En el desarrollo de una aplicación web, es importante la comunicación entre la parte del servidor y la parte del cliente, ya que el backend será quien proporcione información a mostrar en el frontend, y el backend gestiona información recibida por el frontend, como pueden ser los campos de un formulario [1].

2.1.1. Tecnologías frontend

Para el desarrollo frontend, existen tres principales lenguajes:

- HTML: El lenguaje HTML [2] es un lenguaje de marcado, y es el componente más básico de la web. Consiste en una serie de elementos identificados con etiquetas, que dan estructura y significado al contenido de las aplicaciones web. Estas etiquetas son las que identifican los diferentes elementos que se visualizan, como textos, imágenes o botones.
- CSS: Cascading Style Sheets [3] es un lenguaje de estilos usado para describir cómo se visualizan los elementos HTML en la web. Estos estilos son los que aportan la apariencia a los elementos, como puede ser el color o el tamaño de los elementos de una página web.
- JS: JavaScript [4] es un lenguaje de código abierto que nació con el propósito de introducir interactividad y dinamismo a las páginas web. HTML y CSS proporcionan los elementos básicos para crear la parte visual de una página web, es decir, los elementos, y cómo se ven estos elementos. Sin embargo, estos

consiguen una web estática. Con la ayuda de Javascript, las páginas web pueden modificarse dinámicamente sin necesariamente realizar peticiones al servidor.

Además de estos lenguajes, se pueden utilizar *frameworks* o marcos de desarrollo basados en JavaScript para la creación del frontend. Estos marcos de desarrollo contienen fragmentos de código HTML, CSS y JS prediseñados, que facilitan la creación de las interfaces de usuario. Estos fragmentos de código, llamados componentes, pueden ser reutilizados, y ayudan a proporcionar una estructura para mantener el código [5]. Existen muchos frameworks de desarrollo frontend, entre ellos:

- Angular: Angular es un framework basado en componentes desarrollado en Typescript. Este framework tiene varias bibliotecas integradas que proporcionan funcionalidades como enrutamiento o gestión de formularios [6]. Angular proporciona una arquitectura Modelo-Vista-Controlador (MVC), y existe un intercambio de datos bidireccional entre el modelo y la vista, proporcionando una sincronización entre ambas partes [7].
- React: React [8] es una biblioteca de JavaScript basada en componentes. Una de las principales características es que los archivos utilizan la extensión JSX (JavaScript Syntax). Esta sintaxis es una combinación de HTML y JavaScript, en la que se incrustan objetos JavaScript dentro de las etiquetas HTML. Otra funcionalidad de React es el Virtual DOM. El DOM Virtual es una copia del DOM real, y cuando se realiza algún cambio, en lugar de actualizar todo el DOM, se actualiza solamente las partes que han cambiado, haciendo que los cambios sean más rápidos.
- Vue: Vue [9] es un framework de JavaScript basado en componentes. Al igual que React, también utiliza el Virtual DOM, por lo que los cambios realizados sobre el DOM serán más rápidos. En Vue, el código HTML, CSS y JavaScript se encuentra en un solo archivo, proporcionando mejor legibilidad y comprensión, al tener todo el código de un componente en el mismo archivo. También proporciona bibliotecas para el enrutamiento de la aplicación.

2.1.2. Tecnologías Backend

- PHP: El lenguaje PHP [10] es un lenguaje de código abierto del lado del servidor. La principal característica es que permite incrustar código PHP en HTML. La diferencia con el uso de HTML y JS es que este código se ejecuta en el lado del servidor, lo cual implica que el lado cliente recibe el resultado final tras su ejecución, sin ver ni tener acceso al código.

  Entre los puntos fuertes de este lenguaje se encuentra la facilidad para su aprendizaje, las aplicaciones pueden ejecutarse en varios sistemas operativos, y la integración de módulos de conexión a bases de datos, que facilita la conexión con diferentes tipos de bases de datos [11].

  Al igual que en los lenguajes de frontend, los lenguajes de backend también usan frameworks. Uno de los más utilizados en PHP es Laravel. Laravel es un

que proporciona características como sistemas de autenticación, servicio de correo, o enrutamiento [12].

- Python: Python es un lenguaje de alto nivel e interpretado, y entre sus muchos usos, se puede utilizar para aplicaciones web en el lado del servidor.

  Una de las mayores ventajas es la facilidad de aprendizaje, ya que su sintaxis es similar al lenguaje natural, en inglés. Esto facilita entender el código, y permite crear programas con menos líneas de código en comparación con otros lenguajes [13]. Este lenguaje también es utilizado para ciencias de datos y aprendizaje automático, gracias a bibliotecas librerías integradas que están orientadas a estas áreas [14]. Uno de los framework más populares para Python es Django. Una de las principales ventajas que aporta es la velocidad para el desarrollo web, proporcionando funciones como una interfaz de administración de contenido, autenticación, un flujo de autenticación, o herramientas para interactuar de forma más sencilla con una base de datos [15].

- Java: Java es un lenguaje orientado a objetos, siendo uno de los primeros lenguajes en ser orientado a objetos. Es un lenguaje multiplataforma, lo que significa que el mismo código puede ser ejecutado en distintas plataformas[16]. Al compilar un programa en Java, se crea un código de bytes. Una vez compilado, se puede ejecutar este código de bytes en cualquier sistema operativo que tenga la máquina virtual de Java (JVM). La JVM es la encargada de interpretar el código de bytes a código máquina para que el programa pueda ser ejecutado [17].

  Otra ventaja es que está orientado a objetos, ya que permite organizar el código en clases y objetos. Las clases proporcionan la estructura de los objetos, que contienen atributos y funcionalidades, y a partir de estas, se crean los objetos, siendo los objetos instancias de las clases [18]. Algunos de los conceptos a destacar son los siguientes:

  - Encapsulación: la creación de las clases permite proteger el acceso a los datos y las funciones desde fuera de la clase, lo que proporciona seguridad.
  - Herencia: Esto permite a una clase heredar las propiedades de otra clase ya existente, haciendo que el código sea reutilizable.
  - Polimorfismo: Permite crear métodos que tengan el mismo nombre, pero con una firma distinta, lo que hace el código más legible.
- Node.js: Node.js [19] es un entorno de ejecución que permite usar JavaScript en el lado del servidor. Una de las principales características es que está orientado a eventos asíncronos, y solo tiene un hilo de ejecución. Node.js utiliza un bucle de eventos, que permite realizar operaciones de entrada y salida sin bloquear la ejecución. Este bucle infinito ejecuta las operaciones que le llegan, ejecutando primero las más antiguas y manteniendo el orden (FIFO), y queda a la espera mientras no reciba más operaciones. Las operaciones están en una cola de

eventos, y solo entra el siguiente evento, cuando no hay ninguna tarea ejecutándose.

Entre las ventajas de NodeJS se encuentra la escalabilidad, la velocidad de ejecución, que es multiplataforma, o el acceso al gestor de paquetes NPM (Node Package Manager), que permite añadir librerías al proyecto fácilmente [20].

Al tener solo un hilo de ejecución, no es recomendable utilizarlo para aplicaciones que requieran un uso excesivo de la CPU.

El framework más popular de NodeJS es Express. Las principales características de este framework son proporcionar una forma sencilla de gestionar el enrutamiento y las peticiones HTTP, y la inclusión de middleware, que permite ejecutar funciones antes de que la petición llegue al destino final [21].

2.1.3. Bases de datos

Una de las funcionalidades de la aplicación es persistir información. Para ello, se utilizan bases de datos, que permiten almacenar dicha información. Las bases de datos se dividen en dos tipos:

2.1.3.1. Base de datos relacional

Una base de datos relacional almacena los datos en tablas, compuestas por filas y columnas, basándose en el modelo relacional. El modelo relacional es una representación gráfica de las tablas que componen el sistema, y la relación entre ellas. Las columnas son los atributos, que especifican los tipos de datos, y cada fila es un registro, que tiene un valor para cada atributo. Cada fila posee una o más columnas que identifican cada elemento de la tabla de forma única, lo que se conoce como clave primaria. Una clave externa o foránea está formada por una o más columnas, que se corresponden con aquellas de la clave primaria de otra tabla, permitiendo así relacionar las tablas mediante esos atributos comunes. La clave primaria de una tabla, al ser única, impide que haya registros duplicados [22]. Para interactuar con la información estructurada de las bases de datos relacionales, se utiliza el lenguaje de consulta estructurado (SQL), que es un lenguaje estandarizado. Este lenguaje permite realizar diferentes operaciones sobre la base de datos, como consulta, modificación, inserción o eliminación de datos [23].

Las propiedades principales de una base de datos relacional son las denominadas ACID [24], que viene dado por el acrónimo en inglés. Estas propiedades son:

- Atomicidad: en caso de haber varias operaciones, se ejecutarán todas o ninguna, asegurando una transacción completa, que no se quedará a medias.
- Consistencia: permite realizar transacciones que se pueden concluir, y asegura que el estado final de la transacción es un estado válido.
- Aislamiento: cada transacción se realiza de forma que no sea visible para otras transacciones.

- Durabilidad: garantiza que tras realizar una transacción, los cambios persisten de forma permanente.

2.1.3.2. Base de datos no relacional

Las bases de datos no relacionales, a diferencia de las bases de datos relacionales, no existen relaciones entre las entidades. En este caso, las entidades no son tablas, sino que se almacenan utilizando documentos. Estos documentos tienen un formato BSON, que es un objeto JSON binario. Esto permite una mayor flexibilidad, ya que no es necesario definir un esquema de datos [25].

Las principales ventajas son:

- Flexibilidad en el almacenamiento de datos, lo que permite almacenar información no estructurada, especialmente útil con tipos de datos que puedan variar.
- Permite almacenar una gran cantidad de datos, gracias a que es escalable horizontalmente.
- Proporciona un alto rendimiento.

Entre las desventajas se encuentra que no garantiza las propiedades ACID (atomicidad, consistencia, aislamiento, durabilidad) y no posee un sistema estandarizado [26].

2.2. Trabajos relacionados

Una vez analizadas las tecnologías que se pueden usar para la realización de este proyecto, se van a analizar otros trabajos relacionados que tienen elementos comunes con el presente trabajo.

- **Stats for Spotify**

La web “Stats for Spotify” [27] es una aplicación que usa la API de Spotify para mostrar estadísticas acerca de la música que escucha el usuario. En ella se pueden visualizar las canciones, artistas y géneros musicales más escuchados, y en diferentes intervalos de tiempo; última semana, últimos 6 meses y desde siempre.

Esta aplicación tiene un objetivo en común, que es mostrar información personalizada usando la API de Spotify, y mostrar la información mediante una aplicación web.

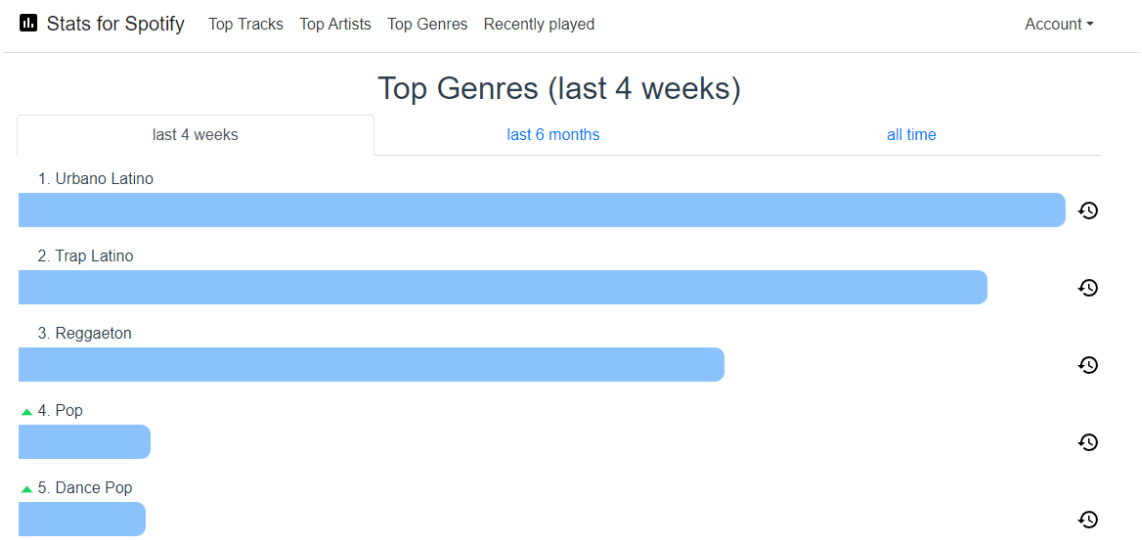

Figura 1: Stats for Spotify

- **Spotify Unchained**

La página web "Spotify Unchained" [28] tiene un objetivo muy similar al del presente trabajo, ya que permite almacenar y crear un archivo de una playlist de Spotify que se actualiza periódicamente. Para ello, de la misma forma, utiliza la API de Spotify, y permite al usuario a través de una aplicación web visualizar las playlists, así como añadirlas guardadas a su cuenta. La principal diferencia con el este trabajo se encuentra en que la playlist que se está guardando, no es una playlist personalizada, sino que es una playlist visible y disponible para todos los usuarios por defecto.

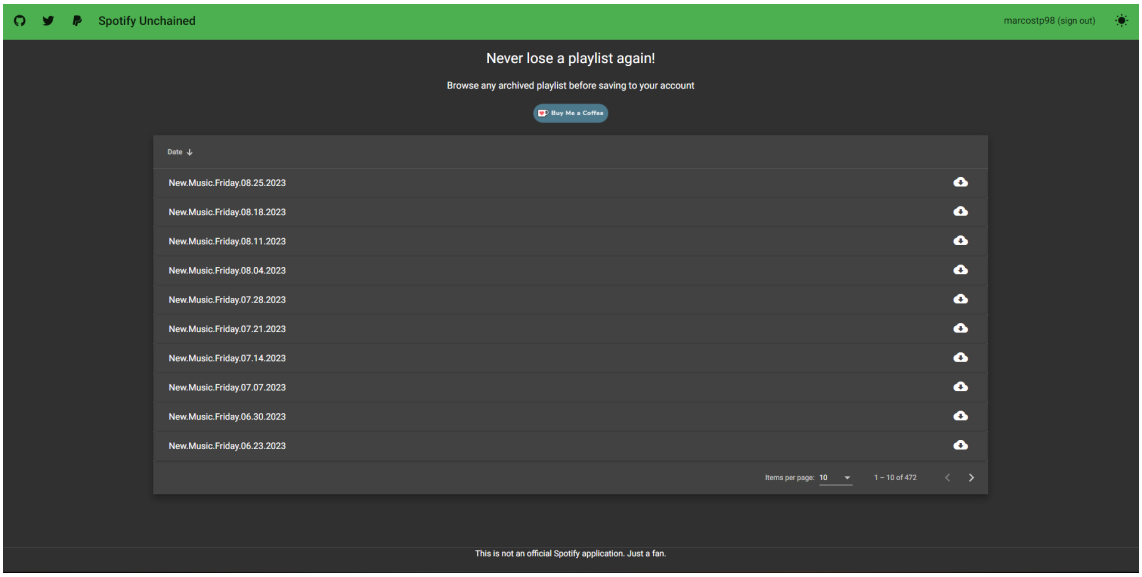

Figura 2: Spotify unchained

- **IFTTT**

IFTTT [29] es una herramienta que permite automatizar ciertas accionesa, que se activan cuando sucede algo en un servicio o aplicación. Se puede conectar con cientos de servicios, como aplicaciones, redes sociales o dispositivos inteligentes, y para cada servicio, se pueden programar una infinidad de acciones. Estas acciones, llamadas "applets", pueden ser creadas por otros usuarios, o que el propio usuario las cree.

Una de las "applets" que ya existen en la aplicación es una que precisamente crea un archivo de la playlist Radar de novedades. El evento que activa la acción es que se añadan nuevas canciones a la playlist Radar de novedades, y la acción que activa es crear una nueva playlist. La principal diferencia con lo que propone este trabajo, es que esta aplicación no proporciona una interfaz visual, ni una base de datos, solamente la creación de una nueva playlist que contenga las mismas canciones, quedando así guardado en la aplicación de Spotify.

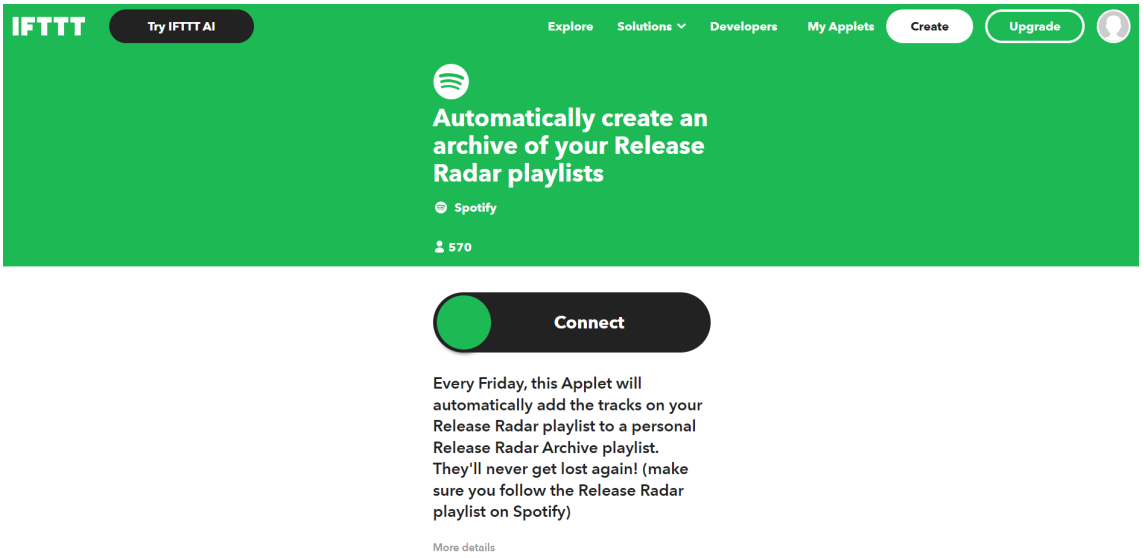

Figura 3: IFTTT

2.3. Conclusión del estado del arte

Para finalizar con el estado del arte, tras el análisis realizado sobre las diferentes tecnologías existentes para la realización de una aplicación web, se mencionan las tecnologías que van a ser utilizadas para la realización de este proyecto. Además de este análisis, se ha tenido en cuenta la experiencia previa del autor con cada tecnología.

Para el desarrollo de la parte frontend, se utilizará HTML, CSS y JavaScript, que son las bases del desarrollo de aplicaciones web. No se utilizará ningún framework de JavaScript para esta parte, ya que las funcionalidades pueden ser cubiertas fácilmente solamente con el uso de Vanilla JavaScript, y utilizar algo más sofisticado sólo introduciría complejidad innecesaria en la aplicación.

En cuanto al desarrollo de backend, la decisión es usar *Node.js*. Uno de los motivos es porque ya se tiene un conocimiento previo de JavaScript, ya que se usa en la parte de

frontend. Por lo tanto, esto permite usar el mismo lenguaje tanto en frontend como en backend, lo que simplifica y agiliza su realización. Otro motivo es el sistema de gestión de paquetes *npm*, que permite instalar bibliotecas y dependencias de forma sencilla. Se utilizará también el framework *express*, para la gestión de las peticiones HTTP y el uso de middleware.

Para almacenar los datos, se ha elegido una base de datos no relacional. En concreto, se usará MongoDB. El principal motivo es su facilidad para integrarse con un sistema que usa JavaScript, ya que las estructuras de datos son en formato BSON, que es un objeto JavaScript binario. La base de datos utilizada será MongoDB, y se utilizará el servicio MongoDB Atlas [30] para desplegar y gestionar la base de datos en la nube. Esta tiene un servicio de pago en función del uso mensual que tenga, y para el caso de esta aplicación, será totalmente gratuita.

3. ANÁLISIS DEL PROBLEMA

En este apartado del documento, primero se especifican los casos de uso de la aplicación. Posteriormente, se redactan los requisitos de usuario, mostrando las necesidades del usuario para utilizar la aplicación. De igual forma, se detallan los requisitos que tiene que tener el sistema para satisfacer dichas necesidades. Finalmente, se realizan las matrices de trazabilidad, donde se valida la especificación de requisitos realizada.

3.1. Casos de uso

La redacción de los casos de uso es importante para definir la interacción de los usuarios con el sistema, describiendo cómo actúa el usuario, y cómo reacciona el sistema a esas acciones [31].

La tabla para definir los casos de uso tiene el siguiente formato:

ID	
Título	
Precondiciones	
Escenario principal	
Postcondiciones	

Tabla 1: Formato tabla casos de uso

La tabla tiene los siguientes campos:

- **ID**: Es el identificador único de cada caso de uso. El identificador tendrá el siguiente formato: "CU-XX". Donde CU significa caso de uso, y las letras xx señalan el número que identifica cada caso de uso. Así, el caso de uso CU-01 identificará el primer caso de uso.
- **Título**: Nombre descriptivo del caso de uso
- **Precondiciones**: indica las condiciones del sistema previas a la realización de la acción.
- **Escenario principal**: Indica cómo se desarrolla la acción.
- **Postcondiciones**: indica las condiciones que el sistema tendrá después de realizar la acción.

Los casos de usos para la aplicación son los siguientes:

ID	CU-01
Título	Registro en la web
Precondiciones	El usuario no tiene una cuenta creada.

Escenario principal	El usuario pulsa el botón de “Crear cuenta” y rellena el formulario con su usuario y contraseña. Una vez finalizado pulsa el botón “crear cuenta”.
Postcondiciones	Una vez el usuario está registrado, no tiene que volver a introducir el usuario y contraseña para iniciar sesión. El sistema redirige a la página de permisos.

Tabla 2: Caso de uso CU-01

ID	CU-02
Título	Iniciar sesión
Precondiciones	El usuario ya tiene una cuenta creada con usuario y contraseña.
Escenario principal	El usuario pulsa el botón de “Iniciar sesión” y rellena el formulario con su usuario y contraseña. Una vez finalizado pulsa el botón “iniciar sesión”.
Postcondiciones	El sistema redirige a la página de permisos.

Tabla 3: Caso de uso CU-02

ID	CU-03
Título	Dar permisos para que la aplicación acceda a la API de Spotify en nombre del usuario
Precondiciones	El usuario ya tiene una cuenta creada con usuario y contraseña, y ha iniciado sesión.
Escenario principal	El usuario pulsa el botón “Dar permisos a Spotify”, lo cual redirige a una pantalla de Spotify, en la que el usuario tiene que iniciar sesión con sus credenciales de Spotify.
Postcondiciones	El sistema redirige a la página de “landing”.

Tabla 4: Caso de uso CU-03

ID	CU-04
Título	Guardar playlist semanal
Precondiciones	El usuario ya ha iniciado sesión y ha dado permisos a Spotify. El usuario sigue la playlist “Radar de novedades” en su cuenta de Spotify.
Escenario principal	El usuario pulsa el botón “Guardar playlist de esta semana”, lo cual muestra un diálogo “*alert*” avisando de que se ha guardado correctamente.

Postcondiciones	La playlist ya se podrá visualizar.

Tabla 5: Caso de uso CU-04

ID	CU-05
Título	Visualizar playlists guardadas
Precondiciones	El usuario ya ha guardado alguna playlist en la base de datos.
Escenario principal	El usuario puede ver las playlists que están guardadas
Postcondiciones	El usuario puede ver las playlists que están guardadas.

Tabla 6: Caso de uso CU-05

ID	CU-06
Título	Añadir playlist a Spotify
Precondiciones	El usuario ya ha guardado alguna playlist en la base de datos.
Escenario principal	El usuario pulsa el botón que está a la derecha de la playlist "Añadir a Spotify".
Postcondiciones	El usuario tendrá la playlist guardada en su biblioteca de Spotify.

Tabla 7: Caso de uso CU-06

El siguiente diagrama (ver figura 4) muestra los casos de uso que existen, y la relación que existe entre ellos

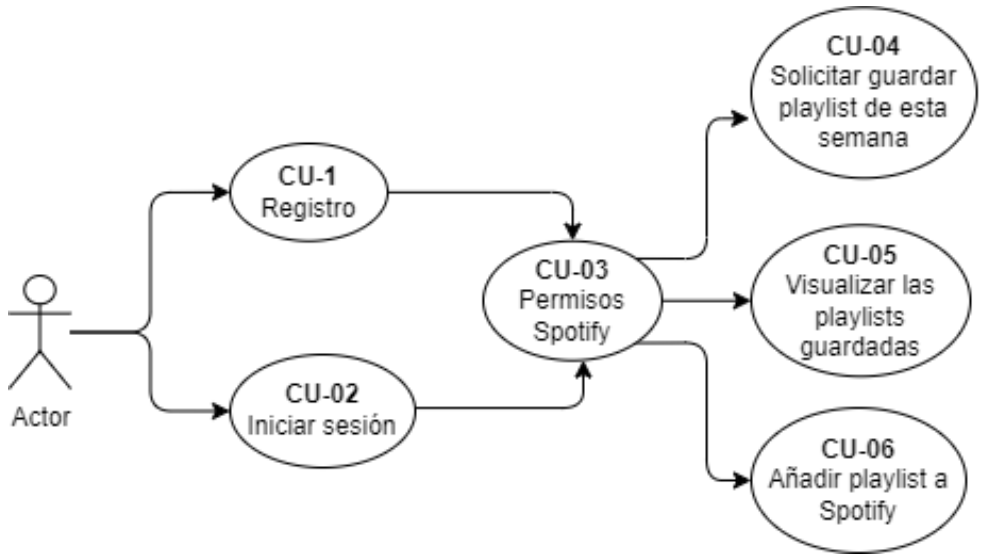

Figura 4: Diagrama de casos de uso

3.2. Requisitos

3.2.1. Requisitos de usuario

Los requisitos de usuario especifican cómo el usuario puede interactuar con el sistema. Estos se dividen en dos tipos:

- Capacidad: describen qué acciones pueden realizar los usuarios
- Restricción: describen las restricciones que tiene el sistema, es decir, limita las acciones de los usuarios.

ID	RU-XX-YY
Nombre	
Necesidad	Esencial | Deseada | Opcional
Prioridad	Alta | Media | Baja
Verificabilidad	Verificable | No verificable
Descripción	

Tabla 8: Formato tabla requisitos de usuario

La tabla tiene los siguientes campos:

- **ID**: Es el identificador único de cada requisito de usuario. El identificador tendrá el siguiente formato: "RU-X-NN". Donde RU significa requisito de usuario. La X denota el tipo de requisito de usuario, que puede ser de capacidad (representado con la letra C) o de restricción (representado con la letra R). Las letras NN señalan el número que identifica cada tipo de requisito. Así, el requisito RU-C-01 identificará el primer requisito de usuario de capacidad.
- **Nombre**: el título que recibe cada requisito.
- **Necesidad**: Muestra la necesidad de cumplimiento del requisito. Los posibles valores son: esencial, deseada y opcional.
- **Prioridad**: Representa el nivel de importancia y prioridad que tiene el requisito. Los posibles valores son: alta, media y baja.
- **Verificabilidad**: Representa la posibilidad de verificar o comprobar que este requisito se cumple mediante alguna prueba. Los posibles valores son: verificable y no verificable.
- **Descripción:** explicación breve.

3.2.1.1. Requisitos de capacidad

ID	RU-C-01
Nombre	Registro de usuario
Necesidad	Esencial
Prioridad	Alta
Verificabilidad	Verificable
Descripción	El usuario puede registrarse en la aplicación con usuario y contraseña

Tabla 9: RU-C-01

ID	RU-C-02
Nombre	Inicio de sesión de usuario
Necesidad	Esencial
Prioridad	Alta
Verificabilidad	Verificable
Descripción	El usuario puede iniciar sesión en la aplicación con su usuario y contraseña

Tabla 10: RU-C-02

ID	RU-C-03
Nombre	Permisos de Spotify
Necesidad	Esencial
Prioridad	Alta
Verificabilidad	Verificable
Descripción	El usuario puede conceder permisos a Spotify a través de la redirección a la pasarela de Spotify

Tabla 11: RU-C-03

ID	RU-C-04
Nombre	Almacenar playlist
Necesidad	Esencial
Prioridad	Alta

Verificabilidad	Verificable
Descripción	El usuario puede guardar la playlist de la semana actual en la base de datos

Tabla 12: RU-C-04

ID	RU-C-05
Nombre	Visualizar playlists
Necesidad	Esencial
Prioridad	Alta
Verificabilidad	Verificable
Descripción	El usuario puede visualizar las playlists que están almacenadas en la base de datos

Tabla 13: RU-C-05

ID	RU-C-05
Nombre	Guardar playlist en Spotify
Necesidad	Esencial
Prioridad	Alta
Verificabilidad	Verificable
Descripción	El usuario puede crear una playlist en su cuenta de Spotify que contenga las canciones de la playlist seleccionada

Tabla 14: RU-C-05

3.2.1.2. Requisitos de restricción

ID	RU-R-01
Nombre	Inicio de sesión para usar la web
Necesidad	Esencial
Prioridad	Alta
Verificabilidad	Verificable
Descripción	El usuario no podrá utilizar la aplicación si no ha iniciado sesión en la aplicación web.

Tabla 15: RU-R-01

ID	RU-R-02
Nombre	Permisos de Spotify para interactuar con Spotify
Necesidad	Esencial
Prioridad	Alta
Verificabilidad	Verificable
Descripción	El usuario no podrá visualizar ni interactuar con playlists si no concede permisos de uso a Spotify.

Tabla 16: RU-R-02

ID	RU-R-03
Nombre	Almacenamiento en base de datos
Necesidad	Esencial
Prioridad	Alta
Verificabilidad	Verificable
Descripción	El usuario no solo podrá visualizar e interactuar con las playlists que hayan sido almacenadas previamente en la base de datos

Tabla 17: RU-R-03

3.2.2. Requisitos del sistema

Los requisitos del sistema o requisitos de software describen las funcionalidades que tiene el sistema. Hay dos tipos de requisitos de sistema:

- Requisitos funcionales: detallan cómo debe funcionar el sistema para que los usuarios puedan realizar los objetivos de la aplicación.
- Requisitos no funcionales: especifican las propiedades del sistema.

La siguiente tabla presenta el formato de tabla con el que se representa la información necesaria de cada requisito del sistema:

ID	RS-XX-YY
Nombre	
Necesidad	Esencial | Deseada | Opcional
Prioridad	Alta | Media | Baja
Verificabilidad	Verificable | No verificable

Caso de uso	CU-NN
Descripción	

Tabla 18: Formato tabla requisitos del sistema

La tabla tiene los siguientes campos:

- **ID**: Es el identificador único de cada requisito de usuario. El identificador tendrá el siguiente formato: "RS-X-YY". Donde RS significa requisito de sistema. La X denota el tipo de requisito de usuario, que puede ser funcional (representado con la letra F) o no funcional (representado con las letras NF). Las letras YY señalan el número que identifica cada tipo de requisito. Así, el requisito RS-F-01 identificará el primer requisito de sistema funcional.
- **Nombre**: el título que recibe cada requisito.
- **Necesidad**: Muestra la necesidad de cumplimiento del requisito. Los posibles valores son: esencial, deseada y opcional.
- **Prioridad**: Representa el nivel de importancia y prioridad que tiene el requisito. Los posibles valores son: alta, media y baja.
- **Verificabilidad**: Representa la posibilidad de verificar o comprobar que este requisito se cumple mediante alguna prueba. Los posibles valores son: verificable y no verificable.
- **Caso de uso:** El identificador del caso de uso con el que está relacionado este requisito.
- **Descripción:** explicación breve del requisito.

3.2.2.1. Requisitos funcionales

ID	RS-F-01
Nombre	Registro en la web
Necesidad	Esencial
Prioridad	Alta
Verificabilidad	Verificable
Caso de uso	CU-01
Descripción	El sistema permite registrarse con un nombre de usuario y contraseña

Tabla 19: RS-F-01

ID	RS-F-02
Nombre	Inicio de sesión en la web
Necesidad	Esencial

Prioridad	Alta
Verificabilidad	Verificable
Caso de uso	CU-02
Descripción	El sistema permite iniciar sesión con un nombre de usuario y una contraseña

Tabla 20: RS-F-02

ID	RS-F-03
Nombre	Permisos de Spotify
Necesidad	Esencial
Prioridad	Alta
Verificabilidad	Verificable
Caso de uso	CU-03
Descripción	El sistema permite obtener permisos para acceder en nombre de un usuario a sus datos de Spotify

Tabla 21: RS-F-03

ID	RS-F-04
Nombre	Almacenar playlist
Necesidad	Esencial
Prioridad	Alta
Verificabilidad	Verificable
Caso de uso	CU-04
Descripción	El sistema permite almacenar una playlist en la base de datos

Tabla 22: RS-F-04

ID	RS-F-05
Nombre	Visualizar playlist
Necesidad	Esencial
Prioridad	Alta
Verificabilidad	Verificable

Caso de uso	CU-05
Descripción	El sistema permite visualizar las playlists guardadas

Tabla 23: RS-F-05

ID	RS-F-06
Nombre	Añadir playlist a Spotify
Necesidad	Esencial
Prioridad	Alta
Verificabilidad	Verificable
Caso de uso	CU-06
Descripción	El sistema permite añadir una playlist guardada a Spotify en nombre del usuario

Tabla 24: RS-F-06

3.3. Matriz de trazabilidad

La matriz de trazabilidad (ver tabla 25) muestra cómo están relacionados los requisitos del sistema con los requisitos de usuario.

	RU-C-01	RU-C-02	RU-C-03	RU-C-04	RU-C-05	RU-C-06
RS-F-01						
RS-F-02						
RS-F-03						
RS-F-04						
RS-F-05						
RS-F-06						

Tabla 25: Matriz de trazabilidad entre requisitos de usuario y del sistema

4. IMPLEMENTACIÓN

En este capítulo, se explica detalladamente el proceso de implementación del proyecto, mencionando los diferentes componentes que participan en la aplicación, su función, y la conexión entre ellos.

La aplicación se divide en cuatro componentes claramente definidos. En el siguiente diagrama se muestra un esquema de la división de componentes, junto con una breve descripción de la función de cada uno:

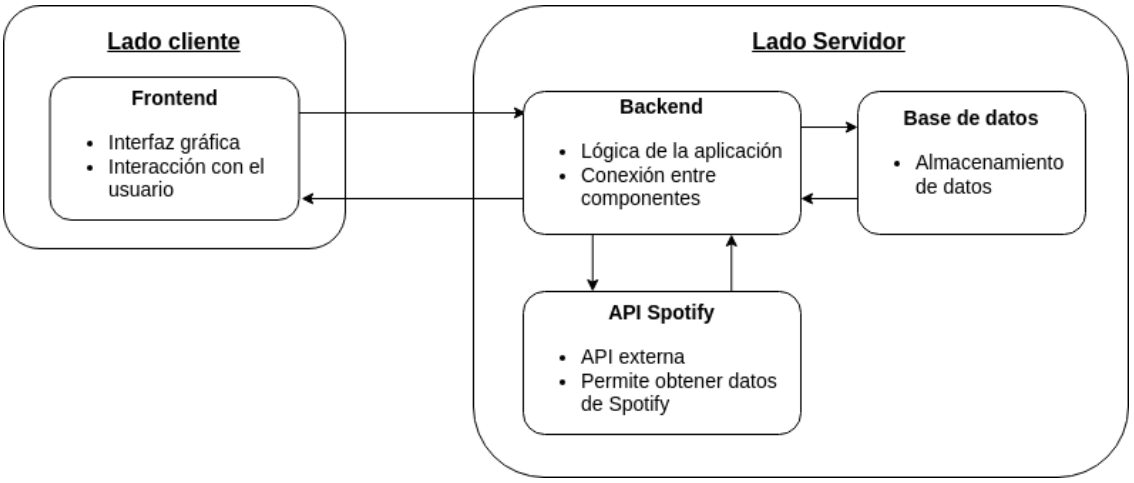

Figura 5: Diagrama de componentes de la aplicación

En el lado del cliente, se encuentra el proyecto del frontend, que es la parte con la que interactúa directamente el usuario. Este proyecto está realizado utilizando HTML, CSS y Vanilla JavaScript.

En el lado del servidor, se ha desarrollado el proyecto del backend, el cual se ha realizado mediante el uso de JavaScript, Node JS y Express JS, y es el encargado de toda la lógica de la aplicación, además de conectar todos los componentes entre sí. La base de datos es no relacional, gestionada mediante MongoDB Atlas. La API de Spotify es una API externa con la cual se pueden obtener datos de la aplicación de Spotify.

A continuación, se profundiza en el diseño e implementación de cada componente, así como en un desarrollo más detallado de la conexión entre los componentes.

4.1. Base de datos

En este apartado se expone cómo se ha estructurado la base de datos no relacional. Existen dos colecciones; la colección *users*, que contiene los usuarios registrados en la aplicación, y la colección *playlists*, que contiene las playlists que los usuarios han almacenado.

La colección de usuarios tiene los siguientes atributos:

- *username*: el nombre de usuario elegido al registrarse en la aplicación.
- *password*: la contraseña del usuario. Por motivos de seguridad y privacidad, lo que se almacena es un hash de la contraseña.
- *spotify*: este campo es un objeto que contiene los siguientes atributos:
    - *accessToken:* el token de acceso para el usuario,
    - *refreshToken:* el token de refresco, que permite obtener un nuevo token de acceso cuando éste expira.
    - *expirationDate:* la fecha en el que el token de acceso expira
    - *userId:* el nombre de usuario en Spotify

En la siguiente figura se muestra un ejemplo de un objeto usuario extraído de la colección de *users:*

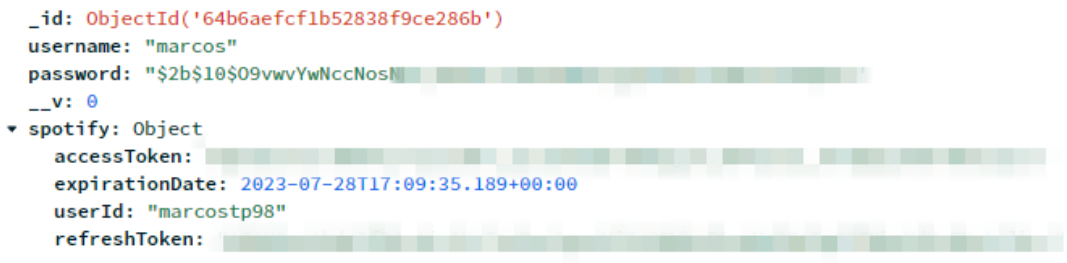

Figura 6: Ejemplo de un objeto user de la base de datos

La colección playlists tiene los siguientes campos:

- *id:* el identificador único de la playlist. Para cada usuario, su playlist de radar de novedades siempre tendrá el mismo valor, por lo tanto es un identificador único para Spotify, pero no para la base de datos.
- *date:* la fecha de la playlist, que sirve para identificar de qué semana es la playlist.
- *userId:* el nombre de usuario de Spotify.
- *songs:* es un array que contiene las canciones de la playlist. Cada canción es un objeto que contiene los siguientes atributos:
    - *name*: el título de la canción.
    - *trackId:* el identificador único para Spotify de cada canción. Este campo será necesario para poder añadir la canción al crear una nueva playlist en Spotify.
    - *artists:* un array que contiene los artistas que participan en la canción. Siempre tendrá mínimo un elemento, pero puede tener más.

La siguiente figura muestra un ejemplo de un objeto perteneciente a la colección de *playlists:*

Figura 6: Ejemplo de un objeto playlist de la base de datos

4.2. Frontend

En el proyecto de frontend, existen cinco páginas html, con sus correspondientes scripts de JavaScript.

- *index.html*: Contiene el nombre de la aplicación, y dos botones, uno para crear cuenta y otro para iniciar sesión (ver figura 16).
- register.html: Contiene el formulario de registro (ver figura 17).
- login.html: página similar a la de registro, en la que el usuario introduce sus credenciales para iniciar sesión (ver figura 18).
- permission.html: página que contiene un botón que redirige a Spotify, para conceder a la aplicación diferentes permisos, que permiten a la aplicación obtener datos de Spotify en nombre del usuario (ver figura 19).
- landing.html: la página que ve el usuario cuando ha iniciado sesión y ya ha concedido los permisos. En ella se puede guardar la playlist de la semana en la base de datos, y se visualizan las playlists que el usuario ya tiene guardadas. Cada playlist existente contiene la funcionalidad de crear una playlist con sus canciones en la cuenta de Spotify del usuario (ver figura 20).

En el apartado de los anexos se proporcionarán capturas de los prototipos de cada una de las páginas presentes en la aplicación web.

4.3. Backend

La funcionalidad del backend es conectar todos los componentes entre sí para que la aplicación funcione correctamente. El componente de la API de Spotify no requiere

un desarrollo directo por parte del autor. Spotify proporciona una serie de *endpoints* públicos, y una documentación [32] sobre cómo configurar tu aplicación para que permita utilizar estos *endpoints*, que serán necesarios para obtener la información deseada, por lo que el objetivo es lograr la conexión entre el servidor backend y la API de Spotify.

Tanto la API de Spotify, como el backend, utilizan endpoints para realizar las distintas funcionalidades. En este punto se van a enumerar los endpoints de cada parte, con una nomenclatura para identificarlos. Para sintetizar y simplificarlo, se van a representar primero con el método de HTTP que corresponda al endpoint (GET, POST, PUT o DELETE), y una url que represente al endpoint. Se utilizará */backend* y */spotify* respectivamente, seguido de la url real de cada endpoint. Los endpoints de backend son los que han sido desarrollados por el autor en esta parte, mientras que los de Spotify son los que se proporcionan en la documentación de la API.

- Endpoints de backend:

Método	URL representativa
POST	/backend/register
POST	/backend/login
GET	/backend/permission
GET	/backend/callback
GET	/backend/user/playlist
GET	/backend/playlist
POST	/backend/playlist

Tabla 26: Endpoints de backend

- Endpoints de Spotify

Método	URL representativa
GET	/spotify/token
GET	/spotify/me/playlists
GET	/spotify/playlist/{playlist_id}
POST	/spotify/users/{user_id}/playlist
POST	/spotify/playlist/{playlist_id}/tracks

Tabla 27: Endpoints de Spotify

El propósito de estas tablas es que se reconozcan y diferencien estos endpoints. En los siguientes apartados se entrará a describir su funcionamiento en mayor detalle, y se incluirán en unos diagramas que muestran la conexión entre los diferentes componentes de la aplicación.

4.3.1. Configuración de Spotify

Para el proyecto de backend, lo primero y paso imprescindible para nuestra aplicación es lograr acceso a la API de Spotify. El primer paso para poder usar la API es crear una "aplicación de Spotify" en su portal de desarrollo [33]. Con esto Spotify identifica nuestra aplicación y es capaz de controlar el propósito de la app, el número de peticiones que hacemos, número de usuarios y demás estadísticas. Además, una vez registrada nuestra app, obtenemos las credenciales para poder obtener respuestas de la API, el identificador del cliente (*client_id*) y el secreto del cliente (*client_secret*).

Asimismo, como la aplicación requiere conocer datos específicos a los usuarios (e.g., ver playlists propias), es necesario pedirle al usuario que permita el acceso a sus datos de Spotify. Por ello, además de un id de cliente, es necesario como aplicación indicar el *scope,* o alcance de nuestra aplicación. El *scope* [34] indica al usuario a qué información tendrá acceso la aplicación de terceros, indicando que proporciona permiso de acceso solamente a lo que se muestra en la pantalla (ver Figura 7).

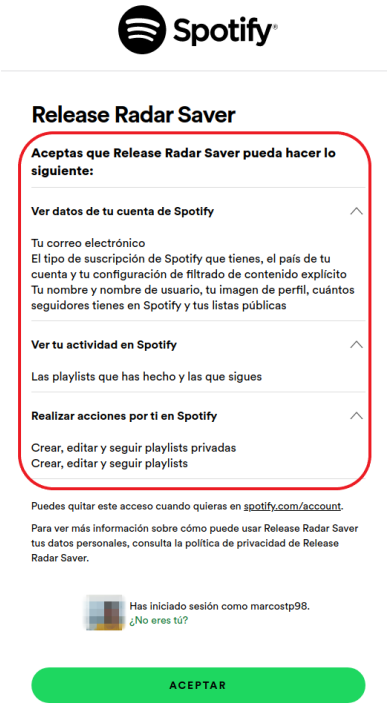

Figura 7: Permisos de Spotify

Para asegurar que nuestra aplicación web es la que está recibiendo los datos de acceso para los usuarios, Spotify también requiere indicar una url de redirección. La

redirect_uri indica la URI a la que se va a redirigir tras aceptar la concesión de permisos. Esta uri debe ser introducida en la aplicación creada previamente en Spotify. En este caso, la uri será una dirección capturada por un endpoint en el backend (ver figura 8).

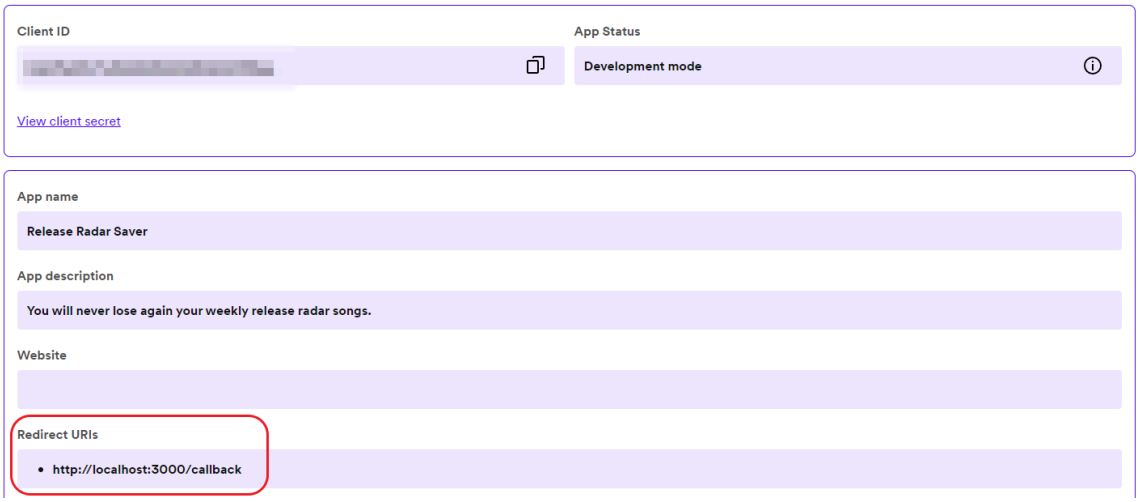

Figura 8: Dashboard de Spotify

Tras esta redirección, se obtiene un código de autorización (*authorization_code),* el cual permite obtener un token de acceso (*access_token*) para el usuario. Cuando se llama al endpoint del token, se obtiene el token de acceso, el token de refresco (*refresh_token*), y el tiempo en el que expira el token de acceso. Esta información se almacena en la base de datos, lo cual permite que el usuario solamente tenga que solicitar los permisos a Spotify una vez. A partir de entonces, se utilizará el token de acceso, o en su defecto el token de refresco, para realizar las peticiones en nombre del usuario. Todo este flujo de autenticación con Spotify, se ve reflejado en la Figura 9.

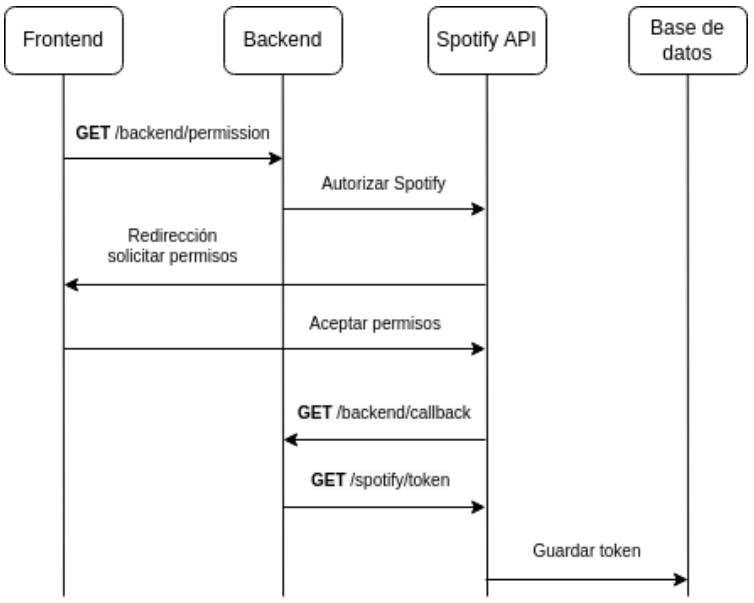

Figura 9: diagrama autenticación Spotify

4.3.2. Autenticación en la aplicación

La autenticación en la aplicación comienza con el registro. En el frontend existe una pantalla de registro que contiene un formulario, en el que se debe introducir el nombre de usuario y contraseña. Tras llamar al endpoint de registro, en el backend se utiliza la biblioteca bcrypt [35] para almacenar un hash de la contraseña, para no almacenar información sensible del usuario. Una vez hecho esto, se inserta el nuevo usuario en la base de datos, si no existía una cuenta previamente con ese nombre de usuario. Una vez el usuario existe en la base de datos, se genera un token JWT [36], el cual contendrá la información del usuario, y este token se enviará como respuesta al frontend. Una vez está en el frontend, se almacena en el *local storage*, y será utilizado para realizar llamadas a los endpoints, asegurando que el usuario que las realiza está con la sesión iniciada en la aplicación. En la siguiente figura se muestra un diagrama con este flujo:

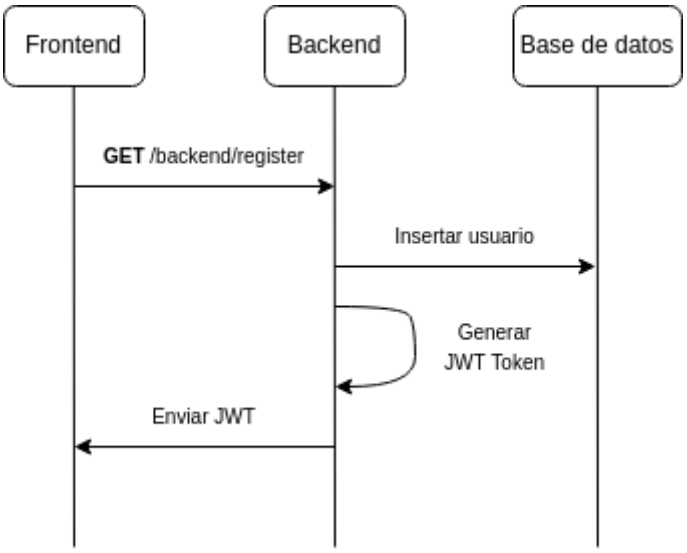

Figura 10: Flujo de registro en la aplicación

Por otra parte, en el endpoint de inicio de sesión, existe una pantalla de inicio de sesión, en el que hay un formulario en el que hay que introducir las credenciales. En el caso del endpoint de inicio de sesión, a diferencia del registro, se comprueba consultando la base de datos si el nombre de usuario existe, y se comparan los hash de las contraseñas, para así determinar si ese usuario está registrado y las credenciales son válidas. A partir de este punto, ambos flujos son equivalentes, enviando al frontend un JWT permitiendo identificar a partir de ese momento al usuario con la sesión iniciada. El flujo para el inicio de sesión se ve representado en la siguiente figura:

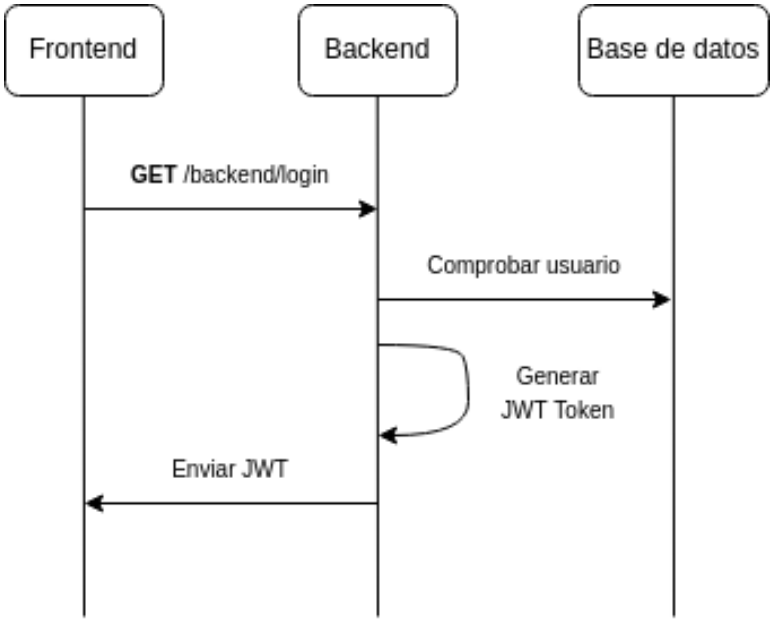

Figura 11: Flujo inicio de sesión en la aplicación

4.3.3. Funcionalidades de Spotify

La primera funcionalidad requerida de Spotify es obtener la playlist de Radar de novedades. Para ello, el primer requisito es que el usuario siga en Spotify dicha playlist, pues el endpoint obtiene únicamente las playlists seguidas por el usuario. Entre esas playlists, se busca por el nombre la playlist del Radar de novedades, para obtener su id. Con este id, se llama al siguiente endpoint, que obtiene las canciones de esta playlist con toda su información. Una vez obtenidos estos datos, se formatean de forma que se guarden solo los datos relevantes para la aplicación, y se almacenan en la colección de playlists de la base de datos (Ver figura 12)

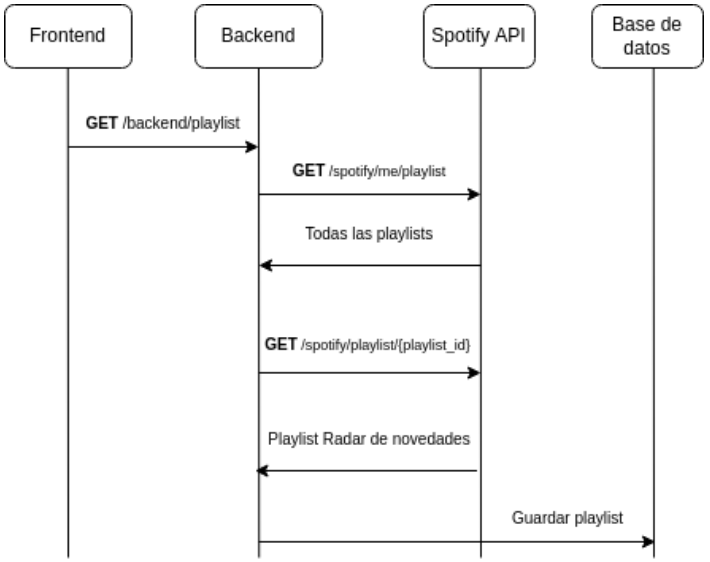

Figura 12: Flujo obtener playlist Radar de novedades

La última funcionalidad de la API de Spotify es la de crear una playlist en la cuenta de Spotify de usuario. Una vez se selecciona la playlist que se quiere añadir a Spotify, se realiza una consulta a la base de datos, para obtener sus canciones. Después, es necesario realizar la llamada el endpoint de creación de playlist, para la cual es necesario el id de usuario de Spotify. Una vez creada esta playlist, se realiza una nueva llamada al endpoint que permite añadir canciones a la playlist. Para ello, utilizando el id de playlist obtenido en la llamada anterior, se añade en el cuerpo de la petición un array con los identificadores de las canciones, obtenidos también previamente (ver Figura 13).

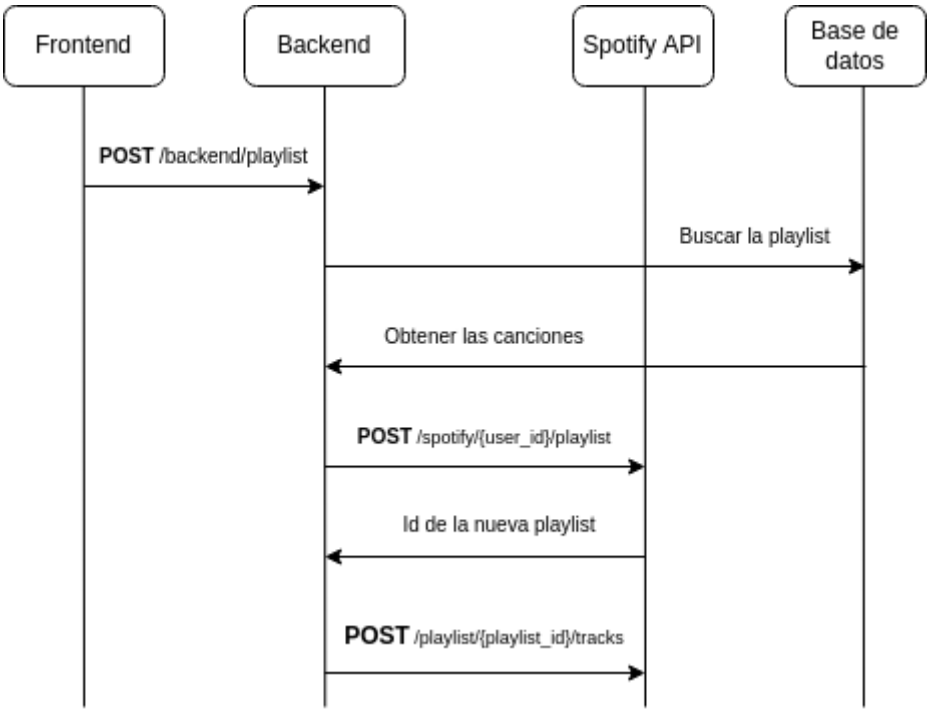

Figura 13: Flujo de creación de playlist en Spotify

4.3.4. Funcionalidad de base de datos

En cuanto a la interacción de la base de datos, además de almacenar información, también es requerida para recuperar dicha información cuando sea necesaria. En el caso de visualizar las playlists en el frontend, es necesario obtener la información de todas las playlists almacenadas para el usuario que las solicita. Una vez obtenida está información, es enviada al frontend, que será el encargado de mostrar los datos en la interfaz (ver Figura 14).

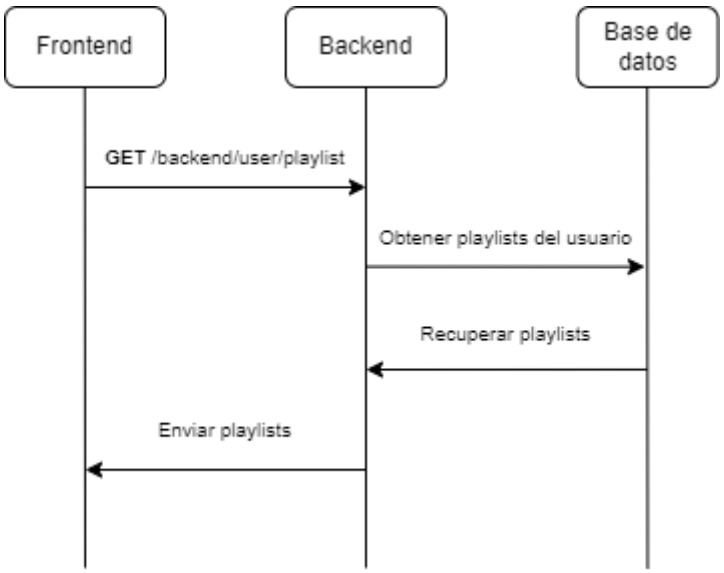

Figura 14: Flujo de obtención de playlists de la base de datos

4.3.5. Recursos adicionales

En este apartado, se comentan recursos adicionales utilizados en el backend, destacando el uso de librerías.

Con el uso de *Express*, se utilizan dos funciones middleware. La primera, denominada *authenticateToken*, se utiliza para obtener el JWT proveniente del frontend, para verificar su validez. Si el token es válido, añadirá el usuario al objeto de solicitud (*req)* y pasará a la siguiente función. La segunda, llamada *checkSpotifyAccessToken*, la cual tendrá el valor del usuario en el objeto de solicitud, añadido en el anterior middleware. Con la información de este usuario, se accede a comprobar la validez del token de acceso de Spotify. Si ha expirado o quedan menos de cinco minutos para que expire, se refresca el token de acceso, y sino, no es necesario realizar ninguna acción. Estos middleware permiten verificar si el usuario está registrado en la aplicación, y asegurará que el usuario puede realizar las llamadas a la API de Spotify, asegurando que

el usuario posee un token de acceso válido. Estas funciones serán añadidas como middleware en las llamadas que lo necesiten.

Tanto para conectar con la base de datos, como para realizar consultas e inserciones, se utiliza la biblioteca *mongoose* [37]. En un fichero aparte, se encuentran la definición de los esquemas y las funciones que realizan cualquier operación sobre la base de datos.

Se utiliza la biblioteca *dotenv* [38] para almacenar información sensible que no debe ser publicada, como las credenciales de acceso a la base de datos, el id de cliente y el secreto de la aplicación de Spotify, o el secreto utilizado para generar los token JWT.

5. EVALUACIÓN

En este apartado, se realizan varias pruebas con el objetivo de comprobar que el sistema tiene el comportamiento esperado. La siguiente tabla presenta el formato de tabla con el que se representa la información necesaria de prueba de evaluación realizada:

ID	P-XX
Nombre	
Precondiciones	
Descripción	
Resultados	
Requisitos relacionados	
Estado	Verificado | Rechazado

Tabla 26: Formato tabla de evaluación

La tabla tiene los siguientes campos:

- **ID**: Identifica los casos de uso. El identificador tendrá el siguiente formato: "P-XX". Donde P significa prueba, y las letras xx señalan el número que identifica cada prueba. Así, el caso de uso PR-01 identificará la primera prueba.
- **Nombre**: Nombre descriptivo de la prueba.
- **Precondiciones:** las condiciones previas para la realización de las acciones.
- **Descripción:** Descripción de los pasos realizados.
- **Resultados**: Indica cómo reacciona el sistema tras los pasos realizados en la descripción.
- **Requisitos relacionados**: Indica los requisitos del sistema que tienen lugar en la realización de la prueba.
- **Estado**: El estado final de la prueba, que puede tomar el valor verificado o rechazado.

En las siguientes tablas se detallan las pruebas realizadas:

ID	P-01
Nombre	Registro de un usuario
Precondiciones	El usuario no tiene una cuenta en el sistema
Descripción	1. El usuario pulsa el botón de crear cuenta. 2. El sistema redirige a la página de registro, donde hay un formulario.

	3. El usuario rellena el formulario con sus credenciales
Resultados	El usuario visualiza la página “permission”. Si el registro es fallido, saldrá un mensaje de error y el sistema no navegará a la siguiente página.
Requisitos relacionados	RS-F-01
Estado	Verificado

Tabla 27: Prueba 1

ID	P-02
Nombre	Inicio de sesión de un usuario
Precondiciones	El usuario tiene una cuenta en el sistema
Descripción	1. El usuario pulsa el botón de iniciar sesión. 2. El sistema redirige a la página de registro, donde hay un formulario. 3. El usuario rellena el formulario con sus credenciales
Resultados	El usuario visualiza la página “permission”. Si el inicio de sesión es fallido, saldrá un mensaje de error y el sistema no navegará a la siguiente página.
Requisitos relacionados	RS-F-02
Estado	Verificado

Tabla 28: Prueba 2

ID	P-03
Nombre	Obtener permisos de Spotify
Precondiciones	El usuario tiene la sesión iniciada en la aplicación
Descripción	1. El usuario pulsa el botón de obtener permisos de Spotify. 2. El sistema redirige a la pasarela de Spotify, donde el usuario inicia sesión con sus credenciales de Spotify. 3. El usuario concede a la aplicación los permisos que se muestran en la página de Spotify.
Resultados	El usuario visualiza la página landing. En esta página visualiza las acciones que se pueden realizar con Spotify, además de las playlists que ya tenga almacenadas en la base de datos.
Requisitos	RS-F-03, RS-F-05

relacionados	
Estado	Verificado

Tabla 30: Prueba 3

ID	P-04
Nombre	Almacenar playlists
Precondiciones	El usuario tiene la sesión iniciada y ha concedido los permisos a Spotify. Además, el usuario sigue la playlist “Radar de novedades” en Spotify.
Descripción	1. El usuario pulsa el botón de guardar playlist de la semana.
Resultados	El usuario visualiza un mensaje que muestra que la playlist ha sido guardada correctamente. Desde ese momento, el usuario puede visualizar la nueva playlist en la lista de todas las playlists almacenadas
Requisitos relacionados	RS-F-04, RS-F-05
Estado	Verificado

Tabla 31: Prueba 4

ID	P-05
Nombre	Crear playlist en Spotify
Precondiciones	El usuario tiene al menos una playlist almacenada en la base de datos.
Descripción	1. El usuario pulsa el botón de añadir a Spotify que está a la derecha de la playlist.
Resultados	El usuario visualiza un mensaje de éxito, que indica que la playlist se ha guardado correctamente. El usuario podrá visualizar la playlist en su cuenta de Spotify.
Requisitos relacionados	RS-F-06
Estado	Verificado

Tabla 32: Prueba 5

A continuación, se muestra la matriz de trazabilidad entre los casos de prueba, y los requisitos del sistema:

	P-01	P-02	P-03	P-04	P-05
RS-F-01					
RS-F-02					
RS-F-03					
RS-F-04					
RS-F-05					
RS-F-06					

Tabla 33: Matriz de trazabilidad entre requisitos del sistema y casos de prueba

6. PLANIFICACIÓN Y PRESUPUESTO

En este apartado del documento, se detalla la planificación del proyecto, incluyendo la metodología de trabajo que se ha seguido, y un diagrama de Gantt. Asimismo, en esta sección se encuentra el presupuesto, en el cual se detallan los costes del proyecto, tanto humanos como materiales.

6.1. Planificación

6.1.1. Metodologías de trabajo

La planificación del proyecto comienza analizando las metodologías de trabajo existentes, para posteriormente decidir cuál es la que se ajusta mejor al proyecto.

- **Metodología en cascada**: consiste en un modelo lineal y secuencial. El proyecto se divide en varias fases, y cada una de ellas debe de estar completada antes de comenzar con la siguiente fase. Puede haber distintas fases, pero generalmente suelen ser: diseño de requisitos, diseño del sistema, implementación, pruebas, despliegue y mantenimiento. Esta metodología es fácil de comprender y de aplicar, y está indicada para proyectos que tengan los requisitos claramente definidos y no tenga previsto cambiar [39].
- **Metodología scrum**: consiste en realizar entregas parciales en intervalos cortos de tiempo, entre 2 y 4 semanas. Existen varias ceremonias para organizar estos intervalos, llamados *sprints*. Primero se realiza una planificación, en el que se establecen las prioridades de las tareas, y el equipo estima el esfuerzo para la realización de las tareas. También existe una reunión diaria, en la que se informa sobre las tareas que tiene cada miembro del equipo en progreso. Al final del sprint, se realiza una revisión, en la que se presentan las tareas realizadas durante ese periodo de tiempo, y una retrospectiva, en la cual se analiza cómo se ha trabajado y se identifican posibles mejoras para el equipo. Esta metodología está indicada para proyectos que pueden ser cambiantes, ya que la organización en sprints de corta duración, permite una gran flexibilidad [40].

La metodología elegida para este proyecto es la metodología en cascada. La razón principal es porque es un proyecto pequeño y realizado en solitario, y la metodología scrum está más pensada para proyectos de mayor alcance y equipos de varias personas. Además, el flujo secuencial del modelo en cascada se adapta bien al trabajo en solitario. Habrá una división de tareas en la que se realizarán primero partes del presente documento, pasando posteriormente al desarrollo e implementación de la aplicación, para finalizar con las últimas partes del documento.

6.1.2. Diagrama de Gantt

En este apartado, se incluye un diagrama de Gantt (ver figura 15), en el que se puede visualizar la división de tareas y el tiempo que se ha empleado en cada una de ellas. En este diagrama, se puede ver el nombre descriptivo de la tarea, la fecha de inicio y la fecha de finalización. Estas fechas están en formato dd/mm/aaaa.

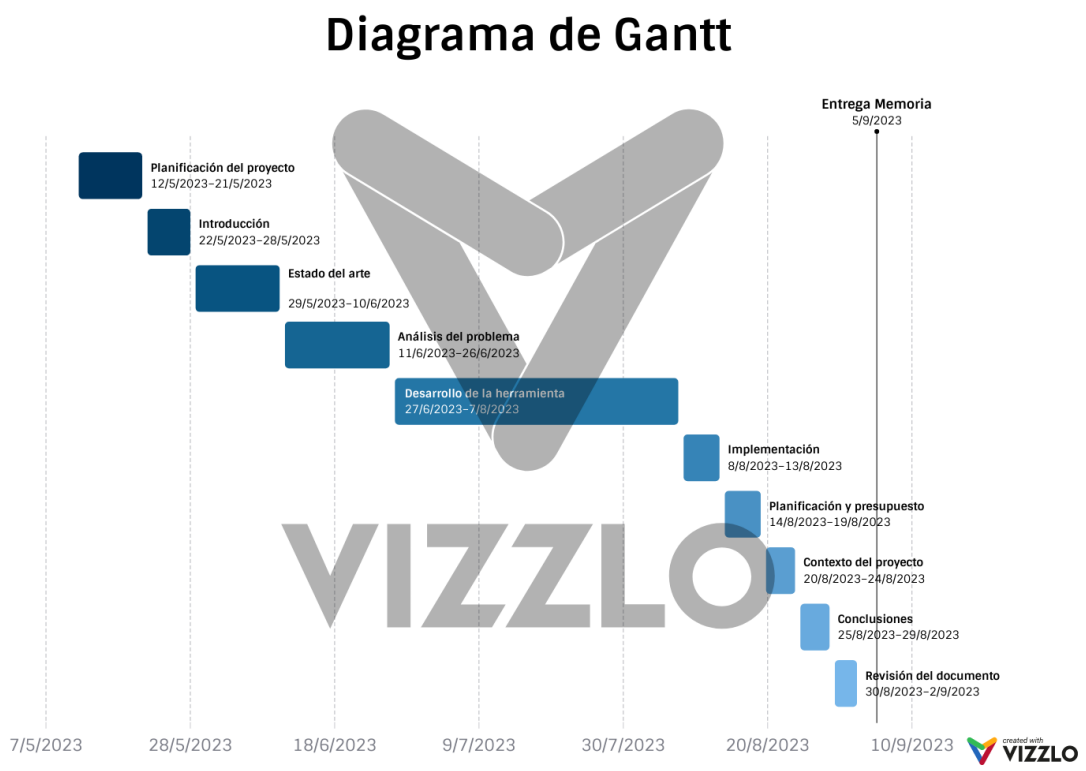

Figura 15: Diagrama de Gantt

6.2. Presupuesto

Esta sección detalla los gastos asociados con la realización del presente proyecto, estando categorizados en: recursos de software, recursos de hardware, recursos humanos y costes indirectos. Se concluye con un resumen que muestra el coste total de realización del proyecto.

Para realizar una estimación de los costes de recursos tanto de software como de hardware, se ha tenido en cuenta la amortización de los activos, que refleja la devaluación de los bienes con el paso del tiempo. Para este cálculo, se utiliza una fórmula que da un valor de amortización en función de un coeficiente en valor porcentual, que depende del activo. Esta fórmula está en términos anuales, para adaptarla a la parte proporcional de la duración del proyecto [41]. Para ello, estimando una duración del proyecto de 4 meses, la fórmula quedaría de la siguiente forma:

$$Amortización = Valor\ del\ activo * \%\ amortización\ anual * (\frac{nº\ meses\ de\ uso}{nº\ meses\ del\ año})$$

Según la web sobre amortizaciones de la agencia tributaria [42], el porcentaje de amortización anual para equipos electrónicos, que es la categoría que aplica para los recursos de hardware, es de un 20%. Por otra parte, para la categoría de Sistemas y programas informáticos, que aplica a los recursos de software, es de un 33% anual.

- **Recursos humanos**

En el diagrama de Gantt (Figura 15) se puede ver que el proyecto ha tenido una duración de 114 días, contando tanto el día de inicio como el día de finalización. Durante estos días se estima una dedicación media de 3 horas diarias, obteniendo un total de 342 horas.

Para el cálculo del coste en recursos humanos, se tiene en cuenta el tiempo empleado por el autor, el cual asume diferentes roles a lo largo de la realización del proyecto. Estos roles son el de jefe de proyecto y desarrollador full-stack, empleo que incluye tanto el desarrollo de frontend como de backend. Para este cálculo, se ha tenido en cuenta el salario promedio en España de ambos puestos de trabajo, obtenido de la web *talent.com* [43]. El desglose de horas y costes se muestra en la siguiente tabla:

Rol	Horas	Sueldo / hora	Coste
Jefe de proyecto	30	19,36 €	580,80 €
Desarrollador Full Stack	312	18,46 €	5756,52 €
		Coste total	**6337,32 €**

Tabla 34: Costes recursos humanos

- **Recursos software**

Teniendo en cuenta la amortización aplicable para los recursos de software, utilizando la fórmula presentada anteriormente, el desglose queda representado en la siguiente tabla:

Producto	Precio	Amortización
Microsoft Windows 11	15,95 €	15,95 €
Microsoft Word	-	-
Visual Studio Code	-	-
MongoDB Atlas	-	-
GitHub	-	-
Zotero	-	-
Draw.io	-	-
	Coste total	**15,95 €**

Tabla 35: Costes recursos software

- **Recursos hardware**

Teniendo en cuenta la amortización aplicable para los recursos de hardware, utilizando la fórmula presentada anteriormente, el desglose queda representado en la siguiente tabla:

Producto	Precio	Amortización
Ordenador sobremesa por piezas	1050,00 €	70,00 €
Monitor LG	145,00 €	9,67 €
	Coste total	**79,67 €**

Tabla 36: Costes recursos hardware

- **Costes indirectos**

En la siguiente tabla se recoge una estimación de los costes indirectos asociados a la realización del proyecto, teniendo en cuenta una duración de 4 meses:

Producto	Coste mensual	Total
Internet	15,00 €	60,00 €
Luz	15,00 €	60,00 €
	Coste total	**120,00 €**

Tabla 37: Costes indirectos

- **Costes totales**

En el resumen de los costes totales, a los costes calculados anteriormente, hay que añadir un margen de beneficios del 20%. A este total contando el margen de beneficio, hay que aplicarle el valor de los impuestos. Para ello, se suma un 21% correspondiente al IVA. El cálculo total queda reflejado en la tabla:

Categoría	Importe
Recursos humanos	6337,32 €
Recursos software	15,95 €
Recursos hardware	79,67 €
Costes indirectos	120,00 €
Subtotal	6552,94 €
Margen de beneficios (20%)	1310,59 €
Total sin IVA	7861,53 €
IVA (21%)	1650,92 €
Total con IVA	**9512,45 €**

Tabla 38: Costes totales

7. MARCO REGULADOR

Para este proyecto, no es necesario el garantizar el cumplimiento del marco legal relativo a aplicaciones web como la que se presenta, puesto que es un prototipo que no es de acceso público. Sin embargo, es importante remarcar el conocimiento de la legalidad vigente en caso que la aplicación fuese desplegada en un contexto real en un futuro.

En el caso de una aplicación como la desarrollada, sería necesario garantizar el cumplimiento de regulaciones tanto españolas como europeas, especialmente relativas a la protección de datos. Estas leyes son las siguientes:

- **Reglamento (UE) 2016/679, Reglamento General de Protección de Datos (RGPD)** [34], es un reglamento europeo que protege a las personas físicas cuando existe un tratamiento de datos personales. Esta ley proporciona a las personas información acerca de cómo se van a tratar sus datos, además de la obligación de ser informados en caso de haber una violación en la seguridad de sus datos por parte de las empresas. Esta ley también concede a las personas el derecho a que, en caso que no haya un motivo para conservarlos, se dejen de tratar sus datos si así lo desea [35].
- **Ley Orgánica 3/2018, de Protección de Datos Personales y garantía de los derechos digitales (LOPDGDD)** [36], es una ley española que tiene como objetivo garantizar la confidencialidad de los datos. Además, se debe proporcionar un consentimiento de tratamiento de datos con una finalidad, no pudiendo ser utilizados con otra finalidad sin que exista un consentimiento específico.

En la aplicación presentada, se trabaja con datos personales (al haber un inicio de sesión), por lo que para garantizar el cumplimiento de las regulaciones sería necesario (1) informar a los usuarios de sus derechos y el propósito de uso de los datos recogidos, (2) permitir la eliminación de las cuentas de los usuarios, e (3) incorporar mecanismos (como e-mails automatizados) que puedan notificar a los usuarios en caso de brechas de seguridad que comprometan sus datos.

8. ENTORNO SOCIO-ECONÓMICO

La creación de está aplicación surge por la identificación por parte del autor de la necesidad de almacenar esta playlist semanalmente. De la misma forma, otros usuarios que utilicen Spotify para escuchar música, y quieran tener siempre disponibles las canciones nuevas de cada semana, podrían utilizar la presente aplicación, lo cual tendría un impacto social para las personas que la usen.

Actualmente, la aplicación no está desplegada en la web, pero en caso de estarlo, podría tener un impacto económico, ya que podría generar beneficios, ya sea porque la

aplicación se venda a alguna empresa, o monetizando por publicidad o una forma de suscripción dentro de la aplicación.

Finalmente, la creación de esta aplicación también tiene un impacto importante sobre el autor. Además de poder utilizar la funcionalidad de la aplicación como usuario, se ha obtenido experiencia en la gestión y organización de un proyecto. De igual modo, se ha aplicado de forma práctica lo aprendido durante el grado en Ingeniería Informática, obteniendo así un gran conocimiento sobre el desarrollo web, tanto de frontend como de backend, siendo está experiencia muy valiosa para el desarrollo profesional.

9. CONCLUSIONES

9.1. Conclusión sobre los objetivos

En este apartado, se observa en retrospectiva la sección 1.2 Objetivos, para comprobar si la aplicación final cumple con los objetivos fijados inicialmente.

El objetivo principal era crear una aplicación web que permitiese almacenar la playlist Radar de Novedades. Para analizar si se ha cumplido el objetivo principal, se mira también a los objetivos secundarios que fueron definidos.

- Creación de una interfaz web sencilla e intuitiva. Este objetivo se ha logrado creando una interfaz construida con HTML, CSS y JavaScript, manteniendo un estilo claro y simple.
- Creación de un servidor web, que se conecta con la base de datos, con la interfaz web, y que realiza las llamadas a la API de Spotify. Este objetivo ha sido cumplido, y se ha expresado con más detalle en los apartados anteriores acerca de la conexión entre los distintos componentes de la aplicación.
- Autenticación del usuario en la aplicación, para verificar la identidad del usuario conectado. Este objetivo se ha conseguido utilizando los token JWT, que permiten identificar a los usuarios.
- Concesión de permisos de Spotify, la cual solo se solicita una vez, ya que la información se asocia al usuario de la aplicación registrado previamente. Esto se ha conseguido gracias al flujo de autenticación de Spotify, y almacenando las credenciales de Spotify en la base de datos, para que solo se tengan que solicitar los permisos la primera vez.
- Permitir la visualización de los datos obtenidos de las playlist al usuario en la interfaz web. Este objetivo se ha logrado utilizando los elementos almacenados en la base de datos, pudiendo ser visualizados desde el frontend.
- Permitir que el usuario pueda crear una playlist en su cuenta de Spotify, usando también las llamadas de la API. Como se explicó anteriormente de forma detallada, este objetivo también ha sido cumplido.

Tras esta recapitulación, se puede comprobar que todos los objetivos fijados inicialmente han sido cumplidos.

9.2. Trabajos Futuros

Con este proyecto se han cumplido los objetivos que se identificaron al comienzo. Sin embargo, existen ciertas características que podrían incorporarse para ampliar y complementar las funcionalidades existentes de la aplicación, entre las cuales se encuentran:

- Permitir el registro e inicio de sesión con Google. Esto permitiría no tener que realizar la gestión de usuarios, siendo además gestionado por Google, lo que transmite confianza y seguridad a los usuarios.
- Tener la web desplegada en un servidor, lo cual permitiría la inclusión de herramientas para poder realizar la acción de guardar las playlists en la base de datos de forma automática cada semana, cuando se actualiza la playlist. Esto evitaría tener que hacerlo manualmente, evitando así la posibilidad de no guardar alguna playlist porque el usuario se haya olvidado de hacerlo.
- Realizar una aplicación móvil, ya que muchos usuarios utilizan Spotify desde sus dispositivos móviles.
- Ampliar las funcionalidades de la aplicación a la hora de añadir las playlists a la cuenta de Spotify, por ejemplo, creando las playlists en diferentes intervalos de tiempo, como mensual o anual.

9.3. Conclusiones personales

Para finalizar, se incluyen unas conclusiones a nivel personal tras la realización de este proyecto.

Este Trabajo Fin de Grado ha supuesto un reto en el que he adquirido muchos conocimientos. En primer lugar, este es el proyecto en solitario más grande al que me he enfrentado, lo que ha exigido que desarrolle una capacidad de gestión y organización del trabajo para continuar avanzando en el proyecto sin perder el foco. También considero muy importante haber podido aplicar lo aprendido a lo largo del grado en ingeniería informática, además de ampliar el conocimiento sobre el desarrollo web. En definitiva, considero que la experiencia de haberme enfrentado en solitario a crear un proyecto desde cero, tanto a nivel organizativo como a nivel de programación, me va a ser de gran ayuda en el ámbito personal y para el mundo laboral.

10. SUMMARY

10.1. Motivation and objectives

Currently, one of the most popular ways to listen to and discover music is through streaming platforms, as they allow users to access a huge catalog of music for free or at a reduced cost from anywhere.

One of the most important platforms in this area is Spotify, on which we focus in this paper.

Spotify and its alternatives have changed the way we access music, discover artists or podcasts. It allows us to organize and expand our personal library through various features it offers, including the creation of playlists, known as "playlists". These lists can be general, such as the lists of hits by country. They can also be customized, either created by users themselves or generated by Spotify based on the musical preferences of each one.

One of the playlists that Spotify provides to its users is called "Release Radar". This includes 30 songs recently released by artists that the user follows and listens to frequently, plus recommendations from other artists based on their tastes. The playlist is updated weekly, specifically on Fridays each week, which means that songs are only available on that playlist for 7 days.

Since Spotify does not allow saving this auto-generated playlist, but rather overwrites it, users lose access to the songs unless they have manually saved them or have "Liked" all the songs in the playlist.

This is why this Final Degree Project aims to address this limitation of Spotify, allowing users to persist these playlists over time, to be able to view and consult them at any time.

10.2. Objectives

The primary objective of this work is to design and develop a web application that allows to store the songs of the Spotify playlist "Release Radar" for each user, using the data that the Spotify platform itself provides through its API.

In order to achieve this objective, the following secondary objectives are established:

- Creation of a simple and intuitive web interface, which allows the user to access and perform all the necessary functionalities.

- Creation of a server, which connects with the database, with the web interface, and which makes calls to the Spotify API.
- Authentication of the user in the application, to verify the identity of the connected user.
- Granting Spotify permissions, which is only requested once, as the information is associated with the previously registered application user.
- Allow the display of data obtained from the playlists to the user in the web interface.
- Allow the user to create a playlist in his Spotify account, also using API calls.

10.3. State of the art

After the analysis of the different existing technologies for the development of a web application, the decision about the technologies that will be used for the realization of this project is taken. In addition to this analysis, the author's previous experience with each technology has been taken into account.

For the development of the frontend, HTML, CSS and JavaScript will be used, which are the basics of web application development. No JavaScript framework will be used for this part, since the functionalities can be easily covered only with the use of Vanilla JavaScript, and using something more sophisticated would only introduce unnecessary complexity in the application.

As for backend development, the decision is to use Node.js. One of the reasons is because you already have a prior knowledge of JavaScript, since it is used in the frontend part. Therefore, this allows using the same language in both frontend and backend, which simplifies and speeds up its implementation. Another reason is the npm package management system, which allows you to install libraries and dependencies in a simple way. The express framework will also be used for the management of HTTP requests and the use of middleware.

The choice of database will be a non-relational database. Specifically, MongoDB will be used. The main reason is its ease of integration with a system using JavaScript, since the data structures are in BSON format, which is a binary JavaScript object. The database used will be MongoDB, and the MongoDB Atlas service will be used to deploy and manage the database in the cloud. This has a paid service depending on the monthly usage, and for the case of this application, it will be totally free.

10.4. Implementation

The application is divided into four clearly defined components.

On the client side, there is the frontend project, which is the part with which the user interacts directly. This project was developed using HTML, CSS and Vanilla JavaScript.

On the server side, the backend project has been developed, which has been done using JavaScript, Node JS and Express JS, and is responsible for all the application logic, as well as connecting all the components together. The database is a non-relational database, managed using MongoDB Atlas. The Spotify API is an external API with which you can get data from the Spotify application.

10.5. Planification

The methodology chosen for this project is the waterfall methodology. The main reason is because it is a small and solitary project, and the scrum methodology is more intended for larger projects and teams of several people. Also, the sequential flow of the waterfall model is well suited for solo work. There will be a division of tasks in which parts of this document will be performed first, then moving on to application development and implementation, and finishing with the last parts of the document.

10.6. Budget

The budget for this project is done taking into account human resources, hardware and software resources, and indirect costs. The total cost for the project, including a profit margin of 20% and VAT, is 9512,45 €

10.7. Social and economic impact

The creation of this application arises from the author's identification of the need to store this playlist on a weekly basis. In the same way, other users who use Spotify to listen to music, and want to have always available the new songs of each week, could use this application, which would have a social impact for people who use it.

Currently, the application is not deployed on the web, but in case it is, it could have an economic impact, as it could generate profits. These benefits would come either because the application is sold to some company, or monetizing by advertising or some form of subscription within the application.

Finally, the creation of this application also has an important impact on the author. In addition to being able to use the functionality of the application as a user, experience has been gained in managing and organizing a project. Similarly, it has been applied in

a practical way what has been learned during the degree in Computer Engineering, thus obtaining a great knowledge about web development, both frontend and backend, being this experience very valuable for professional development.

10.8. Conclusions

The realization of this work has been a challenge, in which I have gained a lot of knowledge. First of all, the ability to manage and organize a project of this size, something I had never faced before. I also consider it very important to have been able to apply what I learned throughout my degree in computer engineering, in addition to expanding my knowledge of web development. The experience of facing alone to create a project from scratch, both organizationally and in terms of programming, will be of great help to me personally and for the working world.

11. BIBLIOGRAFÍA

[1] «Qué es Frontend y Backend: características, diferencias y ejemplos», *Platzi*. https://platzi.com/blog/que-es-frontend-y-backend/

[2] «HTML: Lenguaje de etiquetas de hipertexto | MDN», 24 de julio de 2023. https://developer.mozilla.org/es/docs/Web/HTML

[3] «CSS | MDN», 13 de marzo de 2023. https://developer.mozilla.org/es/docs/Web/CSS

[4] «Acerca de JavaScript | MDN», 8 de agosto de 2023. https://developer.mozilla.org/es/docs/conflicting/Web/JavaScript

[5] «Front End Frameworks: What are the Best Options and Benefits?» https://www.revelo.com/blog/front-end-frameworks

[6] «Angular - ANGULAR FEATURES». https://angular.io/features

[7] «The Good and the Bad of Angular Development», *AltexSoft*. https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-angular-development/

[8] «What are the features of ReactJS ?», *GeeksforGeeks*, 21 de octubre de 2021. https://www.geeksforgeeks.org/what-are-the-features-of-reactjs/

[9] «The Good and the Bad of Vue.js Framework Programming», *AltexSoft*. https://www.altexsoft.com/blog/engineering/pros-and-cons-of-vue-js/

[10] «PHP: ¿Qué es PHP? - Manual». https://www.php.net/manual/es/intro-whatis.php

[11] «Advantages and Disadvantages of PHP», *GeeksforGeeks*, 5 de noviembre de 2020. https://www.geeksforgeeks.org/advantages-and-disadvantages-of-php/

[12] «Laravel - Overview». https://www.tutorialspoint.com/laravel/laravel_overview.htm

[13] «¿Qué es Python? - Explicación del lenguaje Python - AWS», *Amazon Web Services, Inc.* https://aws.amazon.com/es/what-is/python/

[14] «Why Python for Machine Learning? - Python Tutorial». https://pythonbasics.org/why-python-for-machine-learning/

[15] «¿Qué es Django? - Explicación del software Django - AWS», *Amazon Web Services, Inc.* https://aws.amazon.com/es/what-is/django/

[16] «¿Qué es Java? Guía para principiantes de Java | Microsoft Azure». https://azure.microsoft.com/es-es/resources/cloud-computing-dictionary/what-is-java-programming-language/

[17] «Is Java Interpreted or Compiled - Javatpoint». https://www.javatpoint.com/is-java-interpreted-or-compiled

[18] «What is Object-Oriented Programming - Javatpoint». https://www.javatpoint.com/what-is-object-oriented-programming

[19] «Express/Node introduction - Learn web development | MDN», 26 de julio de 2023. https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction

[20] «Node.js Event Loop», *GeeksforGeeks*, 13 de marzo de 2020. https://www.geeksforgeeks.org/node-js-event-loop/

[21] «What is Express JS: A Node.js Express Framework | Intellipaat». https://intellipaat.com/blog/what-is-express-js/
[22] «What is a relational database?» https://www.oracle.com/database/what-is-a-relational-database/
[23] «¿Qué es SQL? - Explicación de lenguaje de consulta estructurado (SQL) - AWS», *Amazon Web Services, Inc.* https://aws.amazon.com/es/what-is/sql/
[24] K. Team, «¿Qué es el modelo ACID en bases de datos?», 19 de abril de 2022. https://keepcoding.io/blog/que-es-acid-bases-datos/
[25] «Bases de datos no relacionales | Bases de datos de gráficos | AWS», *Amazon Web Services, Inc.* https://aws.amazon.com/es/nosql/
[26] «Base de datos no relacional. ¿Qué es? Características y ejemplos», *Ayuda Ley Protección Datos*. https://ayudaleyprotecciondatos.es/bases-de-datos/no-relacional/
[27] «View your personal spotify statistics - Stats for Spotify». https://www.statsforspotify.com/
[28] «Spotify Unchained». https://spotifyunchained.com/
[29] Spotify, «Automatically create an archive of your Release Radar playlists», *IFTTT*. https://ifttt.com/applets/aF9MupAY-automatically-create-an-archive-of-your-release-radar-playlists
[30] «MongoDB Atlas Database | Multi-Cloud Database Service», *MongoDB*. https://www.mongodb.com/atlas/database (accedido 29 de agosto de 2023).
[31] C. M. Whitney EPS Software Corp ,. <a href="/People/Bio/Ellen Whitney">Ellen, «Introduction to Gathering Requirements and Creating Use Cases». https://www.codemag.com/article/0102061/Introduction-to-Gathering-Requirements-and-Creating-Use-Cases
[32] «Web API | Spotify for Developers». https://developer.spotify.com/documentation/web-api
[33] «Dashboard | Spotify for Developers». https://developer.spotify.com/dashboard
[34] «Scopes | Spotify for Developers». https://developer.spotify.com/documentation/web-api/concepts/scopes
[35] «bcrypt», *npm*, 16 de agosto de 2023. https://www.npmjs.com/package/bcrypt
[36] auth0.com, «JWT.IO - JSON Web Tokens Introduction». http://jwt.io/
[37] «Mongoose ODM v7.4.5». https://mongoosejs.com/
[38] «dotenv», *npm*, 17 de junio de 2023. https://www.npmjs.com/package/dotenv
[39] «What is Waterfall model- Examples, advantages, disadvantages & when to use it?» https://tryqa.com/what-is-waterfall-model-advantages-disadvantages-and-when-to-use-it/
[40] «Qué es SCRUM», *Proyectos Ágiles*, 4 de agosto de 2008. https://proyectosagiles.org/que-es-scrum/
[41] B. Santander, «¿Qué es la amortización, qué tipos hay y cómo se calcula?», *Banco Santander*. https://www.bancosantander.es/glosario/amortizacion
[42] «Agencia Tributaria: Amortizaciones».

https://sede.agenciatributaria.gob.es/Sede/impuesto-sobre-sociedades/que-base-imponible-se-determina-sociedades/amortizaciones.html?faqId=42c3904421205710VgnVCM100000dc381e0aRCRD

[43] «Búsqueda de empleo en Talent.com | Encuentra vacantes disponibles cerca de ti.» https://es.talent.com

12. ANEXOS

Anexo A: Prototipos de la aplicación

Figura 16: index.html

Figura 17: register.html

Figura 18: login.html

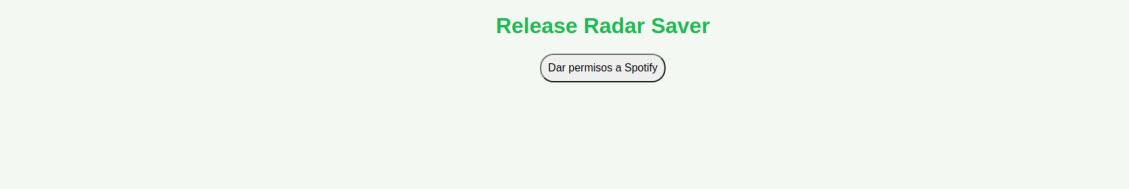

Figura 16: permission.html

Figura 16: landing.html