eman ta zabal zazu

Universidad del País Vasco
Euskal Herriko Unibertsitatea

INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Trabajo de Fin de Grado

Grado en Ingeniería Informática

Ingeniería de Software

Desarrollo de una Plataforma Web Interactiva de Datos Musicales Obtenidos de la API de Spotify

Jon Ortega Goikoetxea

Dirección
Miren Bermejo Llopis

31 de enero de 2025

Agradecimientos

En caso de querer añadir agradecimientos, escribir aquí el texto.

En caso de no querer este apartado, comentalo en el fichero *main.tex*.

Resumen

Escribe aquí el resumen.

Objetivos de Desarrollo Sostenible

Este proyecto se alinea con varios Objetivos de Desarrollo Sostenible (ODS) de las Naciones Unidas. En concreto, se toma como referencia el marco de la *EHUagenda 2030* de la UPV/EHU. Esta agenda, "recoge la contribución de la UPV/EHU a 12 de los 17 ODS de la Agenda 2030, al que ha sumado el su compromiso con la diversidad lingüística y cultural a través del ODS 17+1" [1]. A continuación, se detallan los objetivos específicos relacionados:

Figura 0.1: Los ODS alineados con este trabajo.

- **ODS 3: Salud y Bienestar**. La música juega un papel importante en el bienestar emocional y mental. La capacidad que ofrece esta aplicación de mejorar la experiencia musical y permitir a los usuarios conectar más profundamente con su música contibuye positivamente a su bienestar general.
- **ODS 9: Industria, Innovación e Infraestructura**. Como este proyecto implica la utilización de tecnologías modernas como React, Next.js y Vercel durante el desarrollo, se fomenta la innovación tecnológica y se contribuye al desarrollo de infraestructuras digitales eficientes.
- **ODS 18 (17+1): Garantizar la diversidad lingüística y cultural**. Al permitir que los usuarios exploren y aprecien música de diferentes culturas y en diversos idiomas, se facilita la exposición a estas, fomentando así el entendimiento y la apreciación cultural, alineándose así con el objetivo 18 propuesto por la UPV/EHU.

Índice de contenidos

Índice de contenidos IV

Índice de figuras VI

Índice de tablas VIII

Índice de algoritmos IX

1 Introducción 1
- 1.1. Contexto . . . 1
- 1.2. Motivación . . . 2
- 1.3. Objetivos del Proyecto . . . 2
- 1.4. Estructura de la Memoria . . . 3

2 Contexto Competitivo 4

3 Planificación 6
- 3.1. Alcance . . . 6
  - 3.1.1. Funcionalidades Incluidas . . . 6
  - 3.1.2. Exclusiones . . . 6
  - 3.1.3. Limitaciones . . . 7
- 3.2. Gestión de Tareas . . . 7
  - 3.2.1. Descripción de Tareas . . . 7
  - 3.2.2. Dedicaciones . . . 11
  - 3.2.3. Dependencias entre Tareas . . . 12
  - 3.2.4. Periodos de Desarrollo e Hitos . . . 13
- 3.3. Gestión de Riesgos . . . 14
- 3.4. Herramientas y Tecnologías . . . 16

4 Análisis 20
- 4.1. Estudio de la API de Spotify . . . 20
  - 4.1.1. Descripción General . . . 20
  - 4.1.2. Autenticación y Autorización . . . 21
  - 4.1.3. Principales Endpoints Relevantes para el Proyecto . . . 23

4.1.4. Scopes Necesarios . . . 31
4.1.5. Limitaciones y Consideraciones . . . 32
4.1.6. Política y Términos de Uso de la API . . . 32
4.2. Requisitos Funcionales . . . 34
4.2.1. Autenticación y Sesión . . . 34
4.2.2. Navegación . . . 34
4.2.3. Estadísiticas Generales (Home) . . . 34
4.2.4. Estadísticas Avanzadas (Stats) . . . 35
4.3. Requisitos No Funcionales . . . 37
4.4. Casos de Uso . . . 37
4.4.1. Actores . . . 38
4.4.2. Modelo de Casos de Uso . . . 38

5 Diseño 45
5.1. Arquitectura del Sistema . . . 45
5.1.1. Rutas del Frontend . . . 46
5.1.2. Endpoints del Backend . . . 46
5.2. Diagrama de Componentes de React . . . 47
5.3. Interfaz de Usuario . . . 49
5.3.1. Principios de Diseño . . . 49
5.3.2. Tailwind CSS . . . 50
5.3.3. Páginas y Gráficos . . . 50
5.4. Diagramas de Secuencia . . . 51
5.5. Seguridad . . . 55
5.5.1. Gestión de Credenciales . . . 55
5.5.2. Routing y Conexiones Seguras . . . 56
5.5.3. Otras Medidas . . . 56
5.6. Diseño de Pruebas . . . 57

6 Implementación 58

7 Pruebas 59

8 Despliegue 60
8.1. Vercel . . . 60
8.2. CD/CI . . . 60

9 Conclusiones 61

Apéndice 62

Bibliografía 63

Índice de figuras

2.1. Ejemplo de estadística de "oscuridad" de *Obscurify*. 5
2.2. Imagen generada por MusicScapes. 5

3.1. Diagrama EDT con los paquetes de trabajo del proyecto. 8
3.2. Diagrama de dependencias entre las tareas y paquetes de trabajo del proyecto. 12
3.3. Diagrama Gantt con los tiempos de realización de las tareas y paquetes de trabajo. 13

4.1. Pantalla de autorización de scopes en Spotify. 21
4.2. Plantilla visual para representar los endpoints. 23
4.3. Endpoint de *Request User Authorization*. 24
4.4. Endpoint de *Request Access Token*. 24
4.5. Grupos de endpoints ofrecidos y cuáles se van a usar. 25
4.6. Endpoint de *Get Current User's Profile*. 26
4.7. Endpoint de *Get User's Top Items (Tracks)*. 27
4.8. Endpoint de *Get User's Top Items (Artists)*. 28
4.9. Endpoint de *Get Recently Played Tracks*. 29
4.10. Endpoint de *Get User's Saved Tracks*. 30
4.11. Endpoint de *Get Several Artists*. 31
4.12. Gráfica del funcionamiento de la tasa de peticiones, obtenida de la documentación oficial de Spotify. 32
4.13. Modelo de casos de uso del sistema. 38

5.1. Diagrama de la arquitectura del sistema haciendo uso de *Next.js*. 45
5.2. Diagrama de la jerarquía de componentes de React creados en el proyecto. . . 48
5.3. Colores seleccionados para la paleta de colores de la aplicación. 49
5.4. Muestra de caracteres de la tipografía *Inter*. 50
5.5. Diagrama de secuencia: Iniciar Sesión. 51
5.6. Diagrama de secuencia: Cerrar Sesión. 51
5.7. Diagrama de secuencia: Acceder a *Home*. 52
5.8. Diagrama de secuencia: Acceder a *Stats*. 52
5.9. Diagrama de secuencia: Ver Hall Of Fame. 53
5.10. Diagrama de secuencia: Ver Huella Del Día. 53
5.11. Diagrama de secuencia: Ver Estaciones Musicales. 53
5.12. Diagrama de secuencia: Ver Tus Décadas. 54

5.13. Diagrama de secuencia: Ver La Bitácora. 54
5.14. Diagrama de secuencia: Ver Índice De Interferencia. 54

Índice de tablas

2.1. Comparativa de funcionalidades ofrecidas por otros servicios afines. 4

3.1. Tabla con las estimaciones de tiempo por paquete de trabajo y tarea del proyecto. 11

4.1. Authorization flows definidos por OAuth 2.0 y cuáles implementa Spotify. . . . 22

Lista de Algoritmos

5.1. Ejemplo de aplicación de estilos a un componente JSX usando *Tailwind CSS.* 50

Capítulo 1

Introducción

1.1. Contexto

El uso de datos personalizados es un componente esencial en el desarrollo de aplicaciones y servicios digitales actuales. Las plataformas buscan ofrecer experiencias ajustadas a las preferencias de los usuarios, utilizando grandes volúmenes de datos para generar contenido adaptado. En este contexto, *Spotify* se ha consolidado como una de las plataformas más destacadas de *streaming* musical, no solo por su extenso catálogo, sino también por su capacidad de ofrecer recomendaciones y estadísticas de uso personalizadas.

Una de las características más valoradas por los usuarios de *Spotify* es la posibilidad de acceder a estadísticas personales que les permiten comprender mejor sus hábitos de escucha. En 2016, como producto de una campaña de márketing viral, *Spotify* lanzó *Spotify Wrapped*, un resumen anual de los datos musicales de cada usuario, que generó un gran interés y participación en las redes sociales. Sin embargo, el acceso a estas estadísticas solo está disponible en el mes de diciembre, lo que genera una oportunidad para desarrollar herramientas complementarias que ofrezcan este tipo de análisis de manera más frecuente.

El acceso a estos datos es posible gracias a la *Web API* pública que *Spotify* pone a disposición de los desarrolladores. De esta manera, es posible obtener una amplia gama de datos sobre el comportamiento del usuario, como sus canciones más escuchadas, artistas favoritos y playlists. Además, la API ofrece acceso a muchos datos adicionales no utilizados en *Spotify Wrapped*, pero que, realizando diferentes combinaciones, permiten crear nuevas formas enriquecidas de análisis que presenten la información de manera más atractiva y personalizada.

También es necesario mencionar, que el crecimiento de las aplicaciones web interactivas ha sido posible gracias a tecnologías y frameworks modernos, los cuales permiten crear interfaces rápidas y eficientes, optimizando el rendimiento y la experiencia de usuario y simplificando el desarrollo. Además, los sistemas de integración y despliegue continuo (CI/CD) facilitan la entrega de aplicaciones de manera automatizada y escalable, garantizando que estén siempre actualizadas y accesibles para los usuarios.

1.2. Motivación

La elección de este Trabajo de Fin de Grado surge de un interés personal que he tenido desde el inicio de mi carrera universitaria. Desde el primer año, he tenido la idea base para implementar un servicio en torno a la *Web API* de *Spotify*. Sin embargo, en ese momento no contaba con los conocimientos necesarios para llevarlo a cabo. Ahora, tras completar la carrera, me siento con las capacidades necesarias para poder realizar dicho proyecto y materializar esta idea.

Como usuario habitual de *Spotify*, siempre me ha fascinado la función de *Spotify Wrapped*. No obstante, me resulta limitante que esta información solo esté disponible durante un breve periodo del año. Existen otras herramientas y páginas web que analizan datos de *Spotify*, pero suelen ofrecer estadísticas genéricas y demasiado básicas que carecen de profundidad. Estoy convencido de que es posible obtener estadísticas mucho más interesantes y, conversando con compañeros y otros usuarios, existe un interés general por acceder a estas estadísticas en cualquier momento, no solo una vez al año. La música que cada individuo escucha es algo personal y, a menudo, forma parte de su identidad. Por ello, considero que este proyecto no solo es interesante y motivador para mí, sino que también puede aportar valor a otros usuarios que comparten esta misma idea.

La posibilidad de crear un proyecto que me interese de principio a fin, y que yo mismo desearía utilizar, es una gran motivación. Además, la selección de tecnologías que he decidido emplear, como se detallará más adelante, no solo son ampliamente demandadas en el mercado actual, sino que también contribuyen a mi crecimiento profesional y enriquecen mis conocimientos.

1.3. Objetivos del Proyecto

Para empezar a caracterizar el proyecto de manera concreta, en este apartado se definen los objetivos que guiarán el desarrollo. El **objetivo general** de este proyecto es desarrollar una plataforma web interactiva que permita a los usuarios de *Spotify* acceder a las visualizaciones y análisis de sus datos musicales personales cuando lo deseen. Para alcanzar este objetivo general, se plantean los siguientes **objetivos específicos**:

- Analizar la API de *Spotify* para obtener los datos necesarios.
- Desarrollar la lógica necesaria para filtrar, organizar y transformar los datos obtenidos, preparándolos para su representación gráfica.
- Diseñar una interfaz de usuario intuitiva, interactiva y adaptable a diferentes dispositivos.
- Adoptar tecnologías modernas como *Next.js* y *TypeScript* para beneficiarse de las ventajas que ofrecen.
- Implementar prácticas de CI/CD usando la plataforma de *Vercel*.
- Implementar políticas de seguridad con respecto a los datos de usuarios establecidas por *Spotify* para desarrolladores.
- Documentar el proceso de desarrollo.

1.4. Estructura de la Memoria

La estructura de esta memoria refleja las diferentes etapas por las que atraviesa un proyecto de desarrollo software. Cada capítulo aborda aspectos clave que contribuyen al logro de los objetivos propuestos.

En la Introducción se ha establecido el contexto del proyecto, la motivación que impulsa su realización y los objetivos. Para terminar de definir el contexto en el que se está desarrollando, en el Contexto Competitivo se realizará un análisis de las soluciones ya existentes. A continuación, en la Planificación, se define el alcance del proyecto y se aborda lodo lo relacionado con la gestión de tareas, riesgos y calidad del proyecto. Además, se presentan las tecologías que se han utilizado durante todo el desarrollo.

El Análisis se centra en el estudio de la API de *Spotify*, el cual es **fundamental para el desarrollo del proyecto**. Se especifican los requisitos y se presentan los casos de uso que guiarán el desarrollo de la aplicación. En conjunto con el capítulo anterior, en el Diseño se describe la arquitectura del sistema y la estructura de la interfaz de usuario (UI), tanto a nivel técnico como visual. Además, se abordan aspectos de seguridad y se planifica el diseño de las pruebas.

La Implementación detalla el proceso de desarrollo de la aplicación, incluyendo las decisiones tomadas durante esta fase. Se describen los retos enfrentados y cómo se han resuelto. Seguido, en el capítulo de las Pruebas se exponen las pruebas realizadas y se analizan los resultados obtenidos. Como consecuencia lógica, en Despliegue se explica el proceso de puesta en producción de la aplicación y las estrategias de CI/CD implementadas.

Finalmente, en Conclusiones se hace una reflexión sobre los resultados alcanzados, evaluando el cumplimiento de los objetivos iniciales, el seguimiento en comparación a la planificación inicial y se discuten las lecciones aprendidas y se proponen líneas futuras de trabajo.

Mediante este flujo, se espera que el público lector no tenga problemas para seguir la línea de pensamiento que se ha llevado a cabo y que cada apartado quede aclarado o, en menor medida, justificado por los capítulos anteriores.

Capítulo 2

Contexto Competitivo

Antes de proceder con la planificación del proyecto, es un buen ejercicio analizar las alternativas similares que actualmente existen en el mercado. Como se menciona en la sección de Contexto, la mayoría de otras webs ofrecen estadísticas muy básicas, como visualizar los *top géneros*, *artistas* y *álbumes* del usuario, además de su *historial de reproducción* más reciente. Cada una de estas funcionalidades corresponde a llamadas directas a la *Web API* de *Spotify*, siendo estadísticas triviales de implementar, con una mínima necesidad de procesamiento de datos.

Estadísticas	stats.fm	Obscurify	Stats for Spotify	Replayify	Zodiac Affinity	Music Scapes
Básicas	Top canciones Top artistas Top géneros Historial reciente	Top canciones Top artistas Top géneros	Top canciones Top artistas Top géneros Historial reciente	Top canciones Top artistas Historial reciente		
Avanzadas	Tiempo escuchado Número de artistas Número de álbumes	Porcentaje por décadas				
Creativas	Escuchas por hora del día	Rating de "obscurity" Top artistas "oscuros" Top canciones "oscuras" Tus estados de ánimo			5 canciones según tu signo zodiacal	Imagen basada en las propiedades de las canciones que más escuchas

Tabla 2.1: Comparativa de funcionalidades ofrecidas por otros servicios afines.

Una de las opciones más populares es la aplicación web *stats.fm*[1] (anteriormente *Spotistats*). Esta ofrece las funcionalidades básicas mencionadas a todos los usuarios, pero, mediante un plan de pago, se habilitan gráficas más avanzadas. Estas gráficas no se obtienen directamente de la API, sino que requieren que el usuario descargue manualmente los datos históricos guardados por *Spotify* y los suba a la página web como un archivo comprimido.

[1] stats.fm: https://spotistats.app/

De esta manera, se generan estadísticas relativamente más complejas, pero que aún carecen de "creatividad" en su presentación.

Otro servicio notable es *Obscurify*[2], que se centra en la "oscuridad" o "rareza" de la música que el usuario escucha. El concepto principal de *Obscurify* es identificar las canciones y artistas que son menos populares entre otros usuarios, clasificándolos como más "oscuros". Esta funcionalidad permite que el usuario se sienta especial al escuchar música que no es común. Sin embargo, su enfoque está limitado en su mayoría a este concepto, lo que deja fuera otras formas de visualización o análisis más amplios y variados.

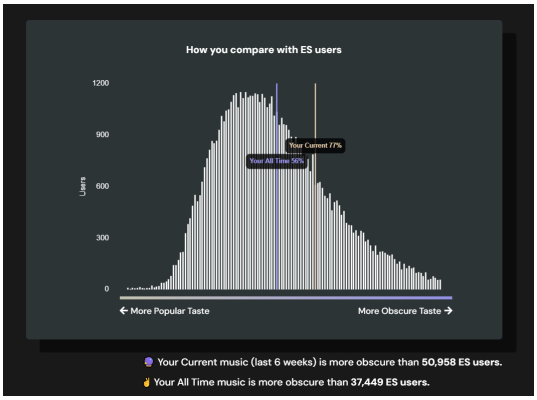

Figura 2.1: Ejemplo de estadística de "oscuridad" de *Obscurify*.

Además de estas páginas principales, existen otras que se enfocan toda su funcionalidad en una única característica original. Dos ejemplos destacados son *Zodiac Affinity*[3] y *MusicScapes*[4]. La primera genera una recomendación de cinco canciones en base a los hábitos de escucha del usuario y su signo zodiacal, mientras que la segunda crea de manera procedural una imagen basada en diferentes propiedades de las canciones que el usuario ha escuchado recientemente. Estas páginas no ofrecen ninguna funcionalidad adicional, limitándose a esa única característica. Aunque hay otros servicios similares, he elegido estos dos ejemplos por su aspecto más "acabado", ya que muchas otras alternativas se presentan más como una prueba de concepto o una demo, en lugar de páginas completamente desarrolladas.

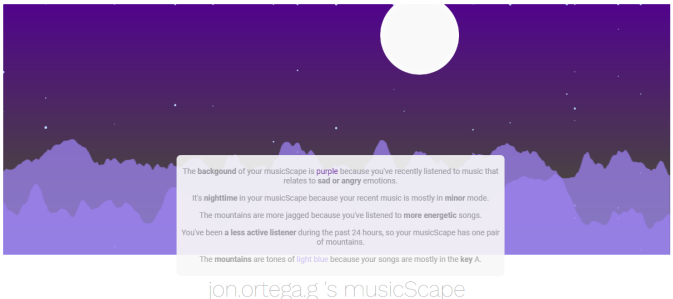

Figura 2.2: Imagen generada por MusicScapes.

[2]Obscurify: https://www.obscurifymusic.com/
[3]Zodiac Affinity: https://zodiacaffinity.eu/
[4]MusicScapes: https://musicscapes.herokuapp.com/

Capítulo 3

Planificación

3.1. Alcance

Definir con precisión el alcance es fundamental para asegurar que el desarrollo se ajuste a los objetivos propuestos y se realice dentro de los recursos y tiempos establecidos. Para ello, en esta sección se delimitarán las funcionalidades, exclusiones y limitaciones que se esperan en este proyecto.

3.1.1. Funcionalidades Incluidas

En la plataforma web se ofrecen las siguientes funcionalidades principales:

- Autenticación segura mediante las credenciales de *Spotify*.
- *Home* o Panel Inicial donde se muestran la información y estadísticas básicas de la cuenta.
- Gráficos interactivos para representar los datos del usuario obtenidos en tiempo real.
- Interfaz intuitiva y responsiva.
- Cierre de sesión seguro.

3.1.2. Exclusiones

Para establecer expectativas claras sobre el alcance del proyecto, se detallan a continuación las funcionalidades que **no** serán incluidas en la plataforma web:

- No se desarrollarán aplicaciones nativas de otras plataformas como móvil, PC, Mac o Linux; el acceso será exclusivamente a través de la web.
- Aunque se seguirá un diseño intuitivo, no se implementarán funcionalidades específicas de accesibilidad avanzadas como compatibilidad con lectores de pantalla o navegación por teclado.
- La plataforma se enfoca exclusivamente en la integración con *Spotify*; se excluyen todos los otros servicios de streaming como *Apple Music*, *Deezer*, etc.

- No se almacenarán de forma persistente datos personales del usuario en servidores propios más allá de lo necesario para la sesión actual; todos los datos se obtendrán directamente de la API de *Spotify* y se manejarán en tiempo real.
- Se excluye el desarrollo de funcionalidades relacionadas con la interacción social (envío de mensajes, compartir estadísticas, rankings entre usuarios, etc.) dentro o a través de la plataforma, ya que superarían el alcance recogido dentro de un TFG.

3.1.3. Limitaciones

Durante el desarrollo del proyecto, se han identificado las siguientes limitaciones que han afectado al alcance y a las funcionalidades de la web:

- Las políticas de seguridad de *Spotify* impiden el almacenamiento persistente de datos personales, limitando funcionalidades que requieran conservar información del usuario entre sesiones.
- El procesamiento de los datos se ve limitado por los recursos computacionales que la nube de *Vercel* ofrece, descartando técnicas avanzadas como el aprendizaje automático.
- El tiempo y los recursos disponibles para el desarrollo del proyecto son finitos, lo que ha obligado a priorizar funcionalidades esenciales y descartar características adicionales.
- Al hacer uso de una API de terceros, todas las funcionalidades necesitan una conexión activa a Internet para poder funcionar de manera correcta.

3.2. Gestión de Tareas

Una vez definido el alcance, es necesario detallar las tareas requeridas para el desarrollo del proyecto. En esta sección se caracterizará todo lo necesario en relación a las tareas, incluyendo su definición, relaciones y tiempos asignados, para asegurar una gestión estructurada y alineada con los objetivos del proyecto.

3.2.1. Descripción de Tareas

Para gestionar de manera efectiva el conjunto de actividades, se ha elaborado una Estructura de Desglose de Trabajo (EDT). Esta EDT (figura 3.1) proporciona una visión general de las principales áreas de trabajo, desglosando el proyecto en paquetes específicos que abarcan cada tarea esencial, facilitando así la gestión.

En este caso, el proyecto se organiza en cinco áreas principales, que abarcan todas las fases del desarrollo de la aplicación; abordando tanto las tareas relacionadas con la creación de la aplicación en sí misma (el producto final) como aquellas enfocadas en la gestión y redacción del proyecto para su documentación. Esta estructura garantiza una distribución clara de las tareas, cubriendo tanto los aspectos técnicos como los organizativos.

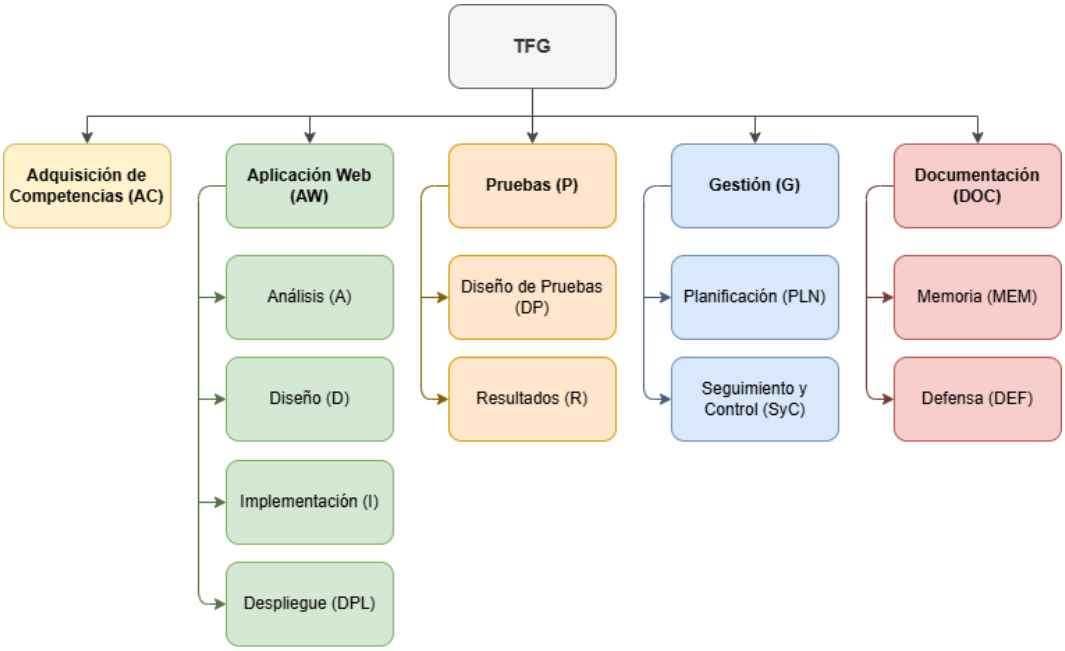

Figura 3.1: Diagrama EDT con los paquetes de trabajo del proyecto.

A continuación se detalla cada paquete de trabajo y las tareas correspondientes contenidas en cada una:

3.2.1.1. Adquisición de Competencias (AC):

Este paquete de trabajo incluye todas las tareas necesarias para adquirir el conocimiento sobre las tecnologías y herramientas clave para el desarrollo del proyecto.

- **AC.1:** Aprender *TypeScript*, *React.js* y *Next.js* para el desarrollo de la aplicación web.
- **AC.2:** Estudiar el uso de *Vercel* para el hosting y despliegue de la aplicación.
- **AC.3:** Hacer un reconocimiento inicial de la *Web API* de *Spotify*.

3.2.1.2. Aplicación Web (AW):

En este paquete se engloban todas las fases de desarrollo de la aplicación web, desde la planificación inicial hasta el despliegue final.

- **Análisis (A):**
  - **AW.A.1:** Estudiar y analizar en profundidad la *Web API* de *Spotify* para determinar el alcance y sus limitaciones.
  - **AW.A.2:** Definir los requisitos funcionales y no funcionales del sistema.
  - **AW.A.3:** Desarrollar los principales casos de uso del sistema.

- **Diseño (D):**
  - **AW.D.1:** Diseñar la arquitectura del sistema.
  - **AW.D.2:** Crear un diagrama de componentes React para establecer la jerarquía y realizar un diseño general de la interfaz de usuario.
  - **AW.D.3:** Definir los diagramas de secuencia de los casos principales.
  - **AW.D.4:** Realizar una gestión de la seguridad y asegurar que se implementarán las medidas definidas por *Spotify* para el uso de la API.
- **Implementación (I):**
  - **AW.I.1:** Implementar el login de la página web, usando el protocolo OAuth 2.0 implementado por *Spotify*.
  - **AW.I.2:** Implementar el panel inicial (dashboard) de la web.
  - **AW.I.3:** Implementar la sección principal de estadísiticas.
  - **AW.I.4:** Implementar la funcionalidad de cerrar sesión.
  - **AW.I.5:** Realizar optimizaciones y correcciones en la implementación.
- **Despliegue (DPL):**
  - **AW.DPL.1:** Configurar el despliegue en *Vercel* para crear un proceso automático de despliegue.
  - **AW.DPL.2:** Monitorear el funcionamiento del despliegue y los logs.

3.2.1.3. Pruebas (P):

Este paquete agrupa las tareas relacionadas con la verificación y validación de la aplicación, garantizando su correcto funcionamiento y calidad.

- **Diseño de Pruebas (DP):** Planificar y diseñar pruebas unitarias, de integración y de carga para evaluar el rendimiento y la estabilidad de la aplicación.
- **Resultados (R):**
  - **P.R.1:** Realizar las pruebas planificadas y documentar los errores encontrados.
  - **P.R.2:** Definir y, en caso de que sea posible, implementar las correcciones necesarias.

3.2.1.4. Gestión (G):

- **Planificación (PLN):**
  - **G.PLN.1:** Realizar una primera estimación de tiempos de las tareas generales.
  - **G.PLN.2:** Establecer el alcance inicial del proyecto según las características del producto seleccionadas.
  - **G.PLN.3:** Definir la planificación del proyecto.
  - **G.PLN.4:** Revisar y, si fuera necesario, modificar la planificación.

- **Seguimiento y Control (SyC):**
  - **G.SyC.1:** Conversaciones y comentarios de la tutora a lo largo del desarrollo.
  - **G.SyC.2:** Elaboración de un documento para registrar las actividades y dedicaciones realizadas a lo largo del proyecto.
  - **G.SyC.3:** Comparación de los datos del seguimiento con los de la placificación, identificación de las desviaciones y riesgos significativos.

3.2.1.5. Documentación (DOC):

Este paquete agrupa las tareas necesarias para la elaboración de la memoria y la preparación de la defensa del proyecto.

- **Memoria (MEM):**
  - **DOC.MEM.1:** Preparar el entorno de trabajo en LaTeX utilizando *Visual Studio Code* y establecer la estructura básica de la memoria a partir de la plantilla proporcionada por la facultad.
  - **DOC.MEM.2:** Redactar la memoria.
- **Defensa (DEF):**
  - **DOC.DEF.1:** Identificar los puntos y conceptos clave que se presentarán en la defensa.
  - **DOC.DEF.2:** Crear los elementos visuales de apoyo para la defensa.
  - **DOC.DEF.3:** Preparar y ensayar la defensa.

3.2.2. Dedicaciones

A continuación, en la tabla 3.1, se presentan las horas estimadas para las tareas descritas en el apartado anterior. También se muestran las sumas de las dedicaciones por paquete de trabajo y la suma total de horas que se espera que lleve el desarrollo del proyecto completo.

	Tarea	Horas
	Adquisición de Competencias (AC)	**24**
	AC.1: TS, React, Next.js	18
	AC.2: Vercel	2
	AC.3: Spotify API	4
Aplicación Web (AW)	**Análisis (A)**	**30**
	AW.A.1: Estudio de Spotify API	16
	AW.A.2: Captura de requisitos	8
	AW.A.3: Casos de uso	6
	Diseño (D)	**37**
	AW.D.1: Aquitectura	5
	AW.D.2: Interfaz de usuario	10
	AW.D.3: Lógica de negocio	18
	AW.D.4: Seguridad	4
	Implementación (I)	**80**
	AW.I.1: Login	7
	AW.I.2: Home	15
	AW.I.3: Estadísticas	40
	AW.I.4: Logout	2
	AW.I.5: Optimizaciones	16
	Despliegue (DPL)	**3**
	AW.D.1: Configurar CI/CD	2
	AW.D.2: Monitorear logs	1
	Pruebas (P)	**30**
	P.DP: Diseñar pruebas	18
	P.R: Resultados	12
	Gestión (G)	**20**
	G.PLN: Planificación	10
	G.SyC: Seguimiento y Control	10
	Documentación (DOC)	**90**
	DOC.MEM: Memoria	80
	DOC.DEF: Defensa	10
	TOTAL	**314**

Tabla 3.1: Tabla con las estimaciones de tiempo por paquete de trabajo y tarea del proyecto.

3.2.3. Dependencias entre Tareas

En la figura 3.2 se muestra un diagrama representando las dependencias que existen entre las diferentes tareas. De esta manera, se puede apreciar de forma visual las tareas que requieren la finalización de una o varias tareas para su comiezo.

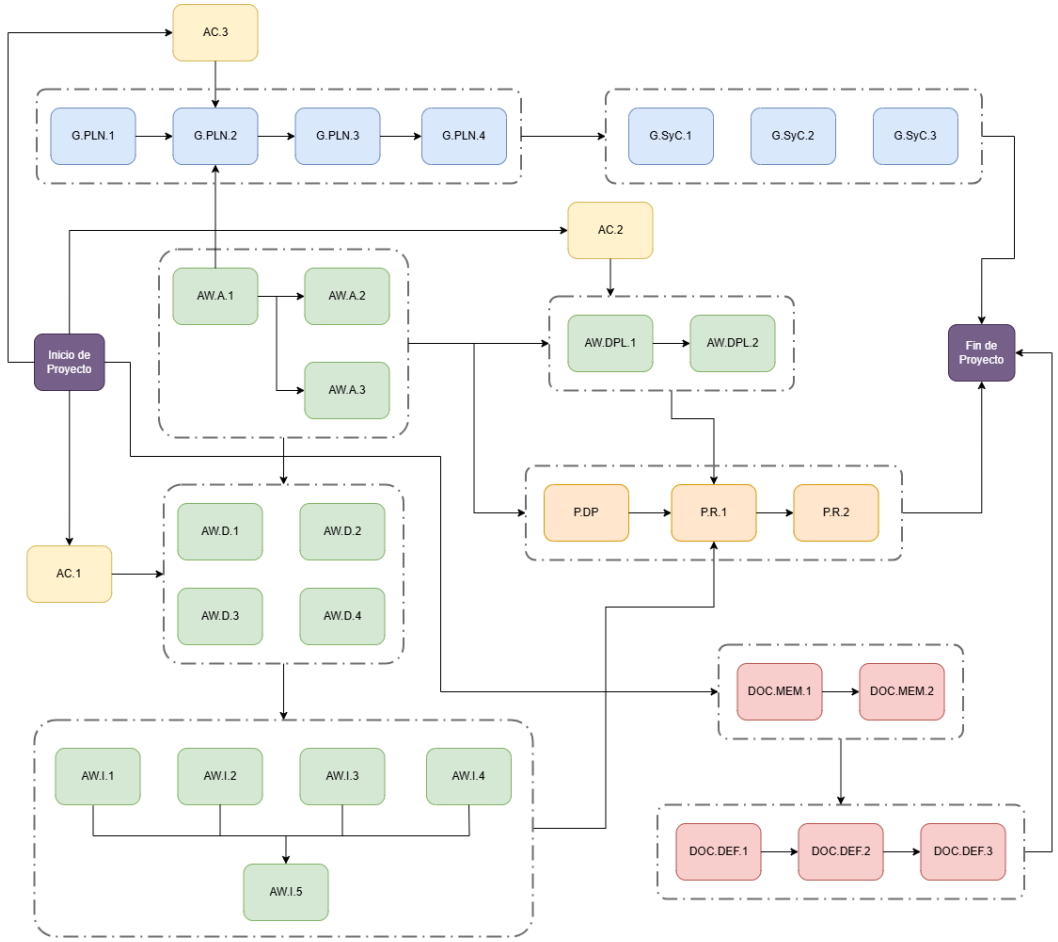

Figura 3.2: Diagrama de dependencias entre las tareas y paquetes de trabajo del proyecto.

Como se puede apreciar, el proyecto debe iniciarse con las tareas de la planificación (paquete de trabajo **G.PLN**) y análisis (**AW.A**), al igual que el desarrollo de las tareas relacionadas con la memoria (**DOC.MEM**), que se realizan desde el inicio del proyecto hasta casi la finalización del TFG. Para ordenar temporalmente todas las tareas, en la siguiente sección se tratarán los periodos de desarrollo de cada una.

3.2.4. Periodos de Desarrollo e Hitos

En esta sección se presenta el diagrama Gantt del proyecto (figura 3.3), el cual ilustra los periodos de realización de las tareas y los paquetes de trabajo; siempre teniendo en cuenta las dedicaciones asignadas y las dependencias entre las tareas que se han establecido en la sección anterior. Además, se destacan los hitos del proyecto, permitiendo una visualización clara y organizada del cronograma planificado.

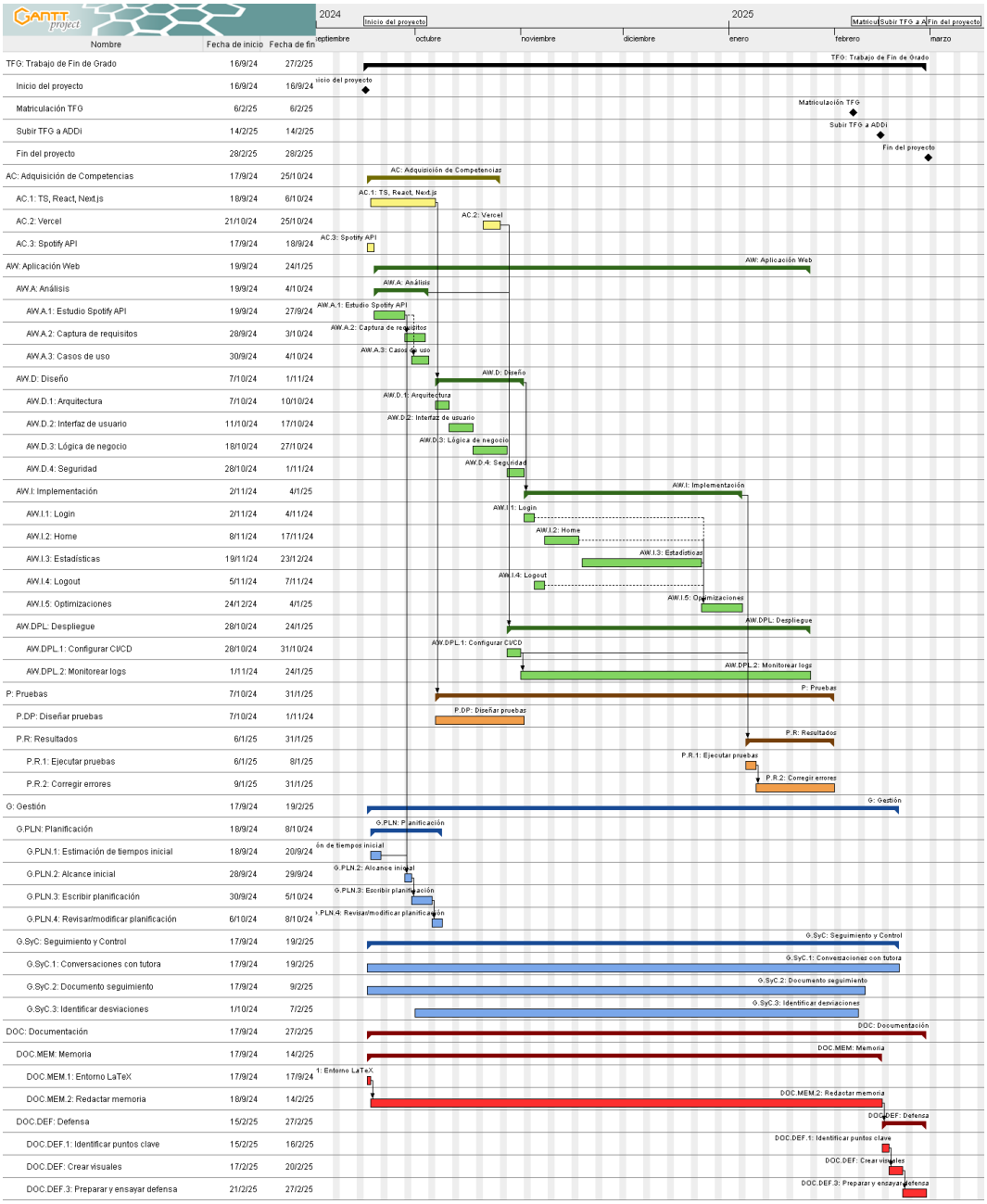

Figura 3.3: Diagrama Gantt con los tiempos de realización de las tareas y paquetes de trabajo.

3.3. Gestión de Riesgos

La gestión de riesgos desempeña un papel crucial en cualquier proyecto, ya que permite anticiparse a posibles inconvenientes, definiendo estrategias adecuadas para minimizar su impacto sobre el proyecto.

A continuación, se presentan los riesgos identificados, especificando para cada uno de ellos la probabilidad de que se materialice, el impacto que podría ocasionar, las medidas implementadas para prevenirlos y las acciones que se llevarían a cabo en caso de que se acabe materializando alguno de ellos.

R01: Limitaciones con respecto a los datos disponibles de la API

Este riesgo se refiere a la posible falta de datos suficientes o adecuados para implementar ciertas funcionalidades previstas, como estadísticas o visualizaciones concretas.

- **Probabilidad**: Media.
- **Impacto**: Medio.
- **Prevención**: Analizar detenidamente los endpoints de la API antes de planificar funcionalidades dependientes de datos específicos.
- **Plan de mitigación**: Proponer gráficos o funcionalidades alternativas que no dependan de esos datos.

R02: Cambios en la política de acceso a la API

Como *Spotify* se reserva el derecho de modificar en cualquier momento las políticas de uso de su API, existe la posibilidad de que algunas funcionalidades del proyecto se vean afectadas o limitadas debido a cambios en los permisos o en la disponibilidad de ciertos endpoints.

- **Probabilidad**: Baja.
- **Impacto**: Alto.
- **Prevención**: Revisar con frecuencia la documentación oficial y adaptar el proyecto a los permisos disponibles desde el inicio.
- **Plan de mitigación**: Ajustar el alcance del proyecto para trabajar con los datos que sigan siendo accesibles y documentar los cambios en la memoria del TFG.

R03: Interrupciones en la disponibilidad de servicios de terceros

Este riesgo engloba posibles problemas de disponibilidad en servicios externos críticos para el proyecto, como la API de *Spotify* o el servicio de hosting de *Vercel*.

- **Probabilidad**: Baja.
- **Impacto**: Alto.

- **Prevención**: Mantener una planificación que contemple margen suficiente para posibles retrasos causados por la indisponibilidad de estos servicios. Además, probar despliegues en un entorno local para avanzar en el desarrollo mientras el servicio de hosting se restablece.
- **Plan de mitigación**: Continuar con el desarrollo de los aspectos que no dependan de estos servicios y retomar las tareas una vez se solucione la interrupción.

R04: Incompatibilidad de versiones de las tecnologías a utilizar

Este riesgo está relacionado con posibles problemas de compatibilidad entre *TypeScript*, *React*, *Next.js* y las librerías utilizadas.

- **Probabilidad**: Media.
- **Impacto**: Alto.
- **Prevención**: Investigar y verificar compatibilidades antes de seleccionar versiones específicas. Evitar usar, en la medida de lo posible, versiones muy recientes que puedan tener problemas de estabilidad y de soporte por parte de otras herramientas.
- **Plan de mitigación**: Actualizar o cambiar las herramientas o librerías problemáticas por alternativas compatibles y realizar pruebas exhaustivas después de cada cambio.

R05: Dificultad en el aprendizaje de las herramientas a utilizar

Al trabajar con herramientas, frameworks y tecnologías con los que no se ha tenido un contacto previo, se pueden ocasionar retrasos por el proceso de aprendizaje.

- **Probabilidad**: Media.
- **Impacto**: Medio.
- **Prevención**: Reservar un tiempo de adquisición de conocimientos y realizar pruebas antes de comenzar con las implementaciones críticas.
- **Plan de mitigación**: Consultar documentación oficial, foros o buscar ayuda en comunidades de desarrollo en línea para resolver dudas rápidamente. Si no es suficiente, se puede avanzar con otra tarea que no dependa de la resolución del problema actual, permitiendo ganar tiempo mientras se sigue investigando la solución al inconveniente técnico.

R06: Dificultad para compaginar el proyecto con las obligaciones académicas

Este riesgo hace referencia a la posibilidad de tener problemas a la hora de gestionar el tiempo disponible para dedicar al proyecto, ya que se está cursando una asignatura en paralelo.

- **Probabilidad**: Media.
- **Impacto**: Alto.
- **Prevención**: Planificar un calendario detallado y realista que reserve horas específicas para trabajar en el proyecto, priorizando las tareas críticas.
- **Plan de mitigación**: Ajustar la planificación redistribuyendo tareas menos prioritarias y minimizando así el impacto en el desarrollo del proyecto.

3.4. Herramientas y Tecnologías

Durante el desarrollo del TFG se usarán diferentes herramientas, tanto para la implementacion de la aplicación web como en la planificación y redacción de la memoria. Cada una de ellas ha sido seleccionada por su idoneidad para la tarea en cuestión. A continuación se mencionan dichas herramientas y tecnologías, su funcion en el proyecto y una justificación de su selección.

3.4.1. TypeScript

Lenguaje de programación que extiende JavaScript, añadiendo un sistema de tipos estáticos y funcionalidades adicionales que mejoran la robustez y mantenibilidad del código. Permite detectar errores durante el desarrollo, en lugar de en tiempo de ejecución, lo que reduce significativamente los problemas en aplicaciones grandes y complejas.

En este proyecto, *TypeScript* se utiliza como lenguaje principal para implementar la aplicación web, gracias a sus múltiples ventajas:

- **Tipado estático**: Permite definir explícitamente los tipos de variables, funciones y componentes, evitando errores comunes como el mal uso de datos o la incompatibilidad entre módulos.
- **Mejor autocompletado y documentación**: Herramientas como *VSCode* ofrecen un autocompletado más preciso y una navegación clara del código, mejorando la productividad del desarrollo.
- **Compatibilidad con JavaScript**: Al ser un superconjunto de JavaScript, es totalmente compatible con cualquier código JavaScript existente, facilitando la integración.
- **Detección temprana de errores**: Como ya se ha mencionado, gracias a su compilador, los errores se detectan antes de ejecutar el código, garantizando una mayor calidad en las entregas.

3.4.2. React.js

Biblioteca de JavaScript desarrollada por *Facebook*, diseñada para la creación de interfaces de usuario modernas y dinámicas. Es ampliamente reconocida por su enfoque declarativo, que facilita la construcción de componentes reutilizables. En este proyecto, *React.js* sirve como base para desarrollar la interfaz de usuario de la aplicación web, aprovechando su capacidad para gestionar de manera eficiente la interacción entre los componentes y el estado de la aplicación.

3.4.3. Next.js

Next.js es un framework de desarrollo web basado en *React.js*, diseñado para facilitar la creación de aplicaciones modernas, escalables y de alto rendimiento. En este proyecto, se utiliza para implementar la página web principal, proporcionando funcionalidades avanzadas como el renderizado híbrido (estático y dinámico), rutas dinámicas y optimización automática de recursos.

El framework emplea varias tecnologías clave para garantizar un entorno de desarrollo eficiente y una construcción optimizada de la aplicación:

- **SWC**: Compilador de alto rendimiento escrito en *Rust* utilizado durante el proceso de construcción.
- **Turbopack**: Empaquetador moderno que reemplaza a *Webpack*, ofreciendo tiempos de desarrollo significativamente más rápidos.
- **ESLint**: Herramienta integrada para el análisis estático de código, que garantiza la detección de errores y el cumplimiento de buenas prácticas.
- **Node.js**: Entorno de ejecución de JavaScript necesario tanto para el desarrollo como para la compilación y el despliegue de la aplicación.

3.4.4. TailwindCSS

Framework de CSS de utilidad que permite crear interfaces de usuario de manera rápida mediante clases predefinidas. Su integración con *Next.js* permite una optimización automática de los estilos, eliminando clases no utilizadas durante el proceso de compilación para reducir el tamaño final del archivo CSS.

3.4.5. Spotify Web API

Siendo el principal servicio utilizado en este proyecto, es una interfaz de programación que permite acceder a diversos datos de usuario y a información relacionada con el contenido disponible en la plataforma de *Spotify*. Esta API ofrece múltiples endpoints que proporcionan acceso a los datos, los cuales gozan de una muy buena documentación en la página web oficial.

3.4.6. Git y GitHub

Git es un sistema de control de versiones ampliamente utilizado en el desarrollo de software, que permite gestionar y realizar un seguimiento de los cambios en el código fuente del proyecto de manera eficiente. Por su parte, *GitHub* es una plataforma basada en la nube que facilita el almacenamiento y la colaboración en proyectos gestionados con *Git*. Juntas, estas herramientas forman una combinación esencial en cualquier proceso de desarrollo de software, proporcionando un entorno robusto para el control de versiones y el respaldo seguro del código.

3.4.7. Vercel

Plataforma de hosting y despliegue continuo, optimizada para aplicaciones web basadas en frameworks como *Next.js*, *React.js*, *Vue.js* y otros. En este proyecto, *Vercel* se utiliza para alojar y mantener la página web, asegurando un proceso de despliegue automatizado gracias a su integración nativa con *GitHub*.

Una de las principales ventajas de *Vercel* es su integración directa con repositorios en *GitHub*. Esto permite que cada vez que se realiza un cambio en el código fuente (como un *push* en la rama principal), se active un flujo de trabajo automatizado para desplegar la versión más reciente de la aplicación. Aunque *Vercel* cuenta con flujos de trabajo internos para automatizar los despliegues, también es posible personalizarlos utilizando *GitHub Actions*, lo que ofrece un mayor control sobre el proceso de CI/CD.

Además de la facilidad de despliegue, *Vercel* proporciona funcionalidades avanzadas que resultan beneficiosas para este proyecto:

- **Renderizado optimizado**: Soporte nativo para el renderizado estático (*Static Site Generation*) y dinámico (*Server-Side Rendering*), fundamentales para proyectos basados en *Next.js*.
- **Red global de distribución de contenido (CDN)**: Los recursos de la aplicación se sirven desde una red global de servidores, garantizando tiempos de carga rápidos para los usuarios en diferentes ubicaciones.
- **Escalabilidad automática**: La plataforma ajusta automáticamente los recursos según la demanda, lo que elimina la necesidad de gestionar manualmente la infraestructura.

3.4.8. Visual Studio Code (VSCode)

Editor de código empleado para el desarrollo del proyecto. Se trata de un editor de código abierto y gratuito, creado por Microsoft y diseñado para soportar una amplia variedad de lenguajes de programación y tecnologías. Para el beneficio del proyecto, cuenta con la integración nativa de *TypeScript* y *React.js*

Además, también se emplea para la edición en local de la memoria del TFG escrita en LaTeX, siendo una mejor alternativa a otros editores específicos del lenguaje y la preferencia personal a los editores locales en comparación con las herramientas online como *Overleaf*. La posibilidad de gestionar tanto el código de la aplicación como la memoria en un mismo entorno mejora significativamente la organización y productividad durante el desarrollo.

3.4.9. LaTeX (MikTeX)

Sistema de preparación de documentos ampliamente utilizado en entornos académicos y técnicos por su capacidad para generar documentos de alta calidad tipográfica. En este proyecto, LaTeX se utiliza para la escritura y edición de la memoria del TFG.

Para la compilación de los documentos, se emplea *MikTeX*, una distribución de LaTeX que incluye las herramientas necesarias para gestionar paquetes y generar archivos en formato PDF. Como ya se ha mencionado, esta distribución se combina con *VSCode* para una agradable experiencia de edición.

3.4.10. Chart.js

Librería de gráficos sencilla y flexible que permite crear visualizaciones interactivas. Se utiliza para representar datos de usuario en la sección de estadísticas de la aplicación web, incluyendo gráficos de barras, líneas y áreas, entre otros. Su facilidad de uso y su integración con *React.js* hacen de esta librería una opción ideal.

3.4.11. Jest

Framework de testing desarrollado por *Facebook*, diseñado para realizar pruebas unitarias y de integración en aplicaciones basadas en JavaScript y *TypeScript*. Es una herramienta clave para identificar y resolver errores antes del despliegue y poder garantizar una buena calidad del producto final.

3.4.12. K6

Herramienta de pruebas de carga (*load testing*) que permite evaluar el rendimiento de aplicaciones web simulando diferentes niveles de tráfico. Sirve para analizar cómo responde la aplicación bajo diferentes condiciones, identificando posibles cuellos de botella y asegurando un rendimiento óptimo incluso en escenarios de alta demanda. Aunque no se espera que la página reciba un tráfico elevado en un principio, realizar pruebas de carga es una buena práctica que contribuye a garantizar la calidad del producto.

3.4.13. Google Drive

Servicio de almacenamiento gratuito en la nube utilizado para guardar y respaldar de manera segura documentos relacionados con el TFG. Es una herramienta muy popular y fácil de usar, perfecta para garantiza que la información esté siempre accesible evitando pérdidas de datos.

3.4.14. Microsoft Excel

Herramienta muy popular para la edición de hojas de cálculo, ampliamente utilizada en diferentes ámbitos debido a su versatilidad y funcionalidades avanzadas. Gracias a su capacidad para generar tablas claras y visuales, se ha empleado para crear las tablas que se incluyen en la memoria del TFG, las cuales se exportan y adaptan fácilmente para su posterior integración en el documento de LaTeX, complementando el contenido técnico con representaciones más visuales.

3.4.15. Draw.io

Herramienta web utilizada para la creación de diagramas incluidos en la memoria del TFG. Su interfaz intuitiva permite elaborar diagramas técnicos con facilidad. Estos diagramas ayudan a ilustrar conceptos del proyecto, facilitando la comprensión de los aspectos técnicos por parte del lector.

Capítulo 4

Análisis

4.1. Estudio de la API de Spotify

La API de Spotify es el núcleo del proyecto, ya que proporciona acceso a los datos necesarios para generar las estadísticas y visualizaciones que constituyen el objetivo principal de este trabajo. Este capítulo detalla el análisis realizado sobre la API, explorando sus capacidades, limitaciones y los recursos que ofrece para su integración.

4.1.1. Descripción General

La API de Spotify es una interfaz que permite a las aplicaciones externas interactuar con la plataforma de manera programática. Su propósito principal es proporcionar acceso estructurado a los datos, permitiendo a los desarrolladores integrar funcionalidades avanzadas en sus propias aplicaciones. A través de esta API, es posible obtener información sobre las canciones, artistas, álbumes, listas de reproducción y **datos personalizados del usuario**, como sus preferencias musicales o sus hábitos de escucha. Gracias a estos últimos, es posible ofrecer una experiencia personalizada para cada usuario.

Entre las características técnicas más destacadas se encuentran:

- **Tipo de API**: RESTful.
- **Protocolo**: HTTPS.
- **Métodos soportados**: `GET`, `POST`, `PUT` y `DELETE`.
- **Formato de respuesta**: JSON
- **Seguridad**: Acceso protegido mediante OAuth 2.0.

Gracias a estas características, la API de Spotify facilita la manipulación de datos gracias a las respuestas en formato JSON, al tiempo que garantiza la seguridad y privacidad del usuario mediante el uso del protocolo OAuth 2.0. Estas cualidades la convierten en una herramienta versátil y segura, capaz de adaptarse a una amplia variedad de proyectos.

4.1.2. Autenticación y Autorización

Antes de avanzar, es importante entender que en este proceso hay dos conceptos clave: **autenticación** y **autorización**. Aunque están relacionados, existen ciertas diferencias importantes:

- **Autenticación**: Es el paso en el que se verifica la identidad del usuario. En este caso, ocurre cuando el usuario inicia sesión en Spotify para confirmar que es quien dice ser. Este proceso es transparente para el desarrollador ya que Spotify, mediante OAuth 2.0, se encarga de gestionarlo.
- **Autorización**: Es el paso en el que el usuario concede permisos para que la aplicación acceda a ciertos recursos de su cuenta. Este proceso es clave, ya que sin estos permisos, la aplicación no podría acceder a los datos necesarios para ofrecer sus funcionalidades.

Una vez que la aplicación obtiene la autorización, Spotify permite el acceso a los datos solicitados de forma controlada. Un concepto fundamental en esta etapa son los **scopes**, que determinan exactamente qué datos y funcionalidades están disponibles para la aplicación.

Scopes: Controlando el Acceso a los Recursos

Los scopes (alcances) permiten a los usuarios tener la tranquilidad de que únicamente compartirán la información que han autorizado explícitamente. Cuando un programador configura el flujo de autorización, debe especificar los scopes necesarios (de un total de 24) para que la aplicación pueda acceder a los recursos protegidos. En función de los scopes solicitados, Spotify mostrará al usuario una pantalla indicando qué permisos específicos se están requiriendo (figura 4.1). El usuario puede entonces aceptar o rechazar estos permisos.

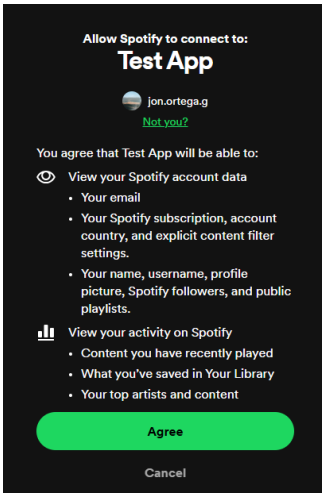

Figura 4.1: Pantalla de autorización de scopes en Spotify.

Una vez que la aplicación obtiene la autorización, el siguiente paso es conseguir el `access_token`, necesario para acceder a los recursos autorizados. Un concepto clave para entender cómo se realiza este proceso es el **authorization flow**, que define las interacciones entre la aplicación, el usuario y Spotify.

OAuth Flows: Elegir el Camino Adecuado

En el marco de OAuth 2.0, el término **flow** (flujo) se refiere a los diferentes procesos diseñados para obtener un `access_token`, dependiendo de las necesidades y características de la aplicación. Estos flujos existen para cubrir una variedad de escenarios, desde aplicaciones web con servidores backend hasta aplicaciones móviles o servicios que no requieren acceso a datos del usuario.

OAuth 2.0 define seis tipos principales de flows, sin embargo, Spotify implementa solo cuatro de ellos (tabla 4.1):

OAuth 2.0	Spotify
Authorization Code	✓
PKCE	✓
Client Credentials	✓
Device Code	✗
Implicit Flow [legacy]	✓
Password Grant [legacy]	✗

Tabla 4.1: Authorization flows definidos por OAuth 2.0 y cuáles implementa Spotify.

Cada uno de estos flujos está diseñado para un escenario específico de uso, por lo que, para poder tomar una buena elección, analizaremos las situaciones en las que cada uno resulta más adecuado:

- **Authorization Code Flow**: Ideal para aplicaciones web que cuentan con un backend seguro donde almacenar el `client_secret`. Proporciona tanto un `access_token` como un `refresh_token`, lo que permite mantener el acceso sin requerir que el usuario se autentique nuevamente. Es el flujo recomendado para aplicaciones con servidores backend que necesitan acceso a datos específicos del usuario.
- **Authorization Code Flow con PKCE**: Es una extensión del anterior, diseñado para escenarios donde no es seguro almacenar el `client_secret`, como en aplicaciones móviles o *Single Page Applications* (SPA). Añade una capa de seguridad utilizando un `code_verifier` y un `code_challenge` para evitar que el código de autorización sea interceptado y utilizado de manera malintencionada.
- **Client Credentials Flow**: Adecuado para aplicaciones backend o servicios que no requieren acceso a datos específicos del usuario, sino que interactúan con recursos de Spotify de manera general. No incluye un proceso de autorización por parte del usuario y no proporciona datos personales.
- **Implicit Grant Flow**: En desuso debido a limitaciones de seguridad. Fue diseñado para aplicaciones cliente que no tienen un backend, pero carece de soporte para `refresh_tokens` y expone el `access_token` en la URL, lo que lo hace menos seguro.

De los cuatro flujos de autorización implementados por Spotify, este proyecto utilizará el **Authorization Code Flow** (sin PKCE). Esta elección se debe a que la aplicación cuenta con un backend seguro implementado con *Route Handlers* de Next.js, lo que permite almacenar de forma segura el `client_secret`. Además, este flujo proporciona tanto un `access_token` como un `refresh_token`, lo que asegura un acceso continuo a los datos del usuario sin necesidad de repetir el proceso de autenticación. Dado que es el flujo recomendado por Spotify para aplicaciones web que necesitan acceder a datos específicos del usuario, garantiza un balance óptimo entre seguridad, funcionalidad y cumplimiento de estándares.

4.1.3. Principales Endpoints Relevantes para el Proyecto

Para facilitar la comprensión de los endpoints utilizados en este proyecto, se ha diseñado un sistema visual que resume la información clave de cada uno, evitando la sobrecarga de texto y permitiendo al lector identificar rápidamente los elementos esenciales.

Cada endpoint se presenta como una interacción entre la solicitud (*request*) y la respuesta (*response*). A la izquierda, la *request* incluye el método HTTP (`GET`, `POST`, `PUT`, `DELETE`), la URL de la petición y los parámetros requeridos, que pueden estar en la URL (para métodos `GET`) o en el cuerpo de la solicitud (*body*, para métodos como `POST` o `PUT`). Si existen parámetros en los *headers*, se mostrarán sobre los parámetros del *body*/URL. A la derecha se encuentra la *response*, con los campos principales que estarán presentes en el JSON devuelto por la API de Spotify, si la solicitud se procesa correctamente.

En la figura 4.2 se muestra una plantilla genérica que sirve como referencia para entender este sistema visual, que se rellenará con la información específica de cada endpoint en las siguientes secciones.

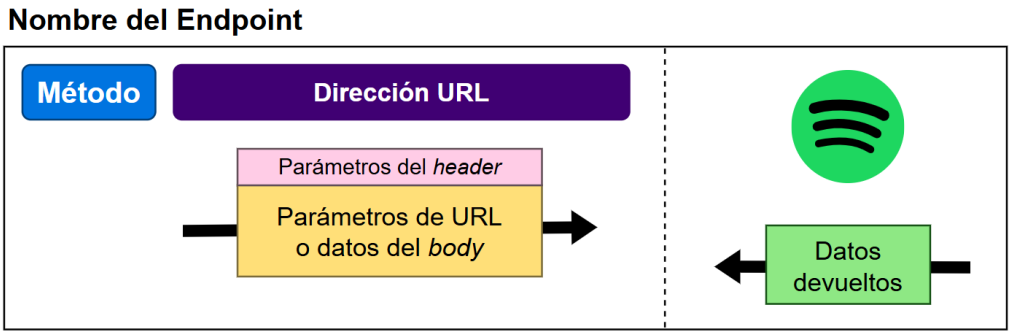

Figura 4.2: Plantilla visual para representar los endpoints.

4.1.3.1. Endpoints de Autenticación

La interacción con Spotify comienza con un endpoint dedicado al proceso de autorización: **Request User Authorization** (figura 4.3). Este endpoint genera la pantalla de autorización que se le muestra al usuario y, en el caso de que acepte, se le redirige a la `redirect_uri` indicada en la *request*. En esta URI siempre se suele implementar la llamada al segundo endpoint llamado **Request Access Token** (figura 4.4), necesario para finalizar el proceso de autorización. Este permite intercambiar el `code` obtenido en el paso anterior por un `access_token`, el cual es requerido para realizar cualquier otra solicitud posterior a la API.

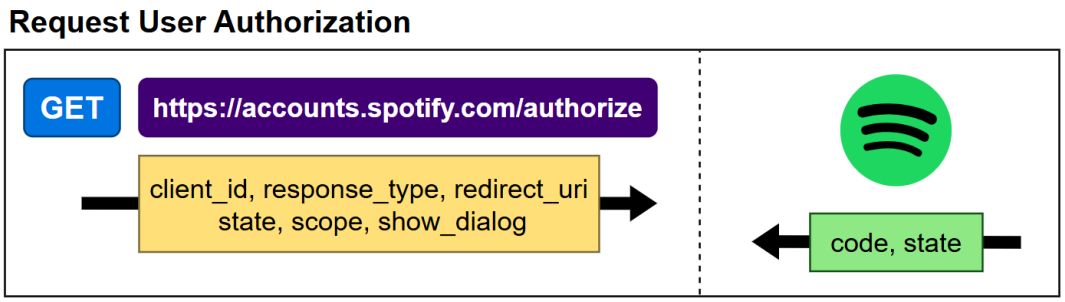

Figura 4.3: Endpoint de *Request User Authorization*.

- **Parámetros del Request**
  - `client_id`: El ID de cliente generado al registrar la aplicación.
  - `response_type`: Se establece en "`code`", indicando que se solicita un código de autorización.
  - `redirect_uri`: URI a la que se redirige al usuario después de aceptar o rechazar los permisos. Debe coincidir exactamente con uno de los valores configurados al registrar la aplicación.
  - `state`: Parámetro utilizado para proteger contra ataques como *cross-site request forgery* (CSRF). Su valor debe ser validado al recibir la respuesta.
  - `scope`: Lista de scopes separados por espacios, indicando los permisos requeridos por la aplicación. Si no se especifican, solo se concederá acceso a información pública.
  - `show_dialog`: Determina si se fuerza al usuario a aprobar nuevamente la aplicación, incluso si ya lo hizo previamente. Si se establece en `true`, el usuario verá el diálogo de autorización; de lo contrario, será redirigido automáticamente.
- **Campos del Response**
  - `code`: Código de autorización que puede intercambiarse posteriormente por un `access_token`.
  - `state`: El valor del parámetro `state` enviado originalmente en la solicitud. Su valor debe ser comparado para garantizar la validez de la respuesta.

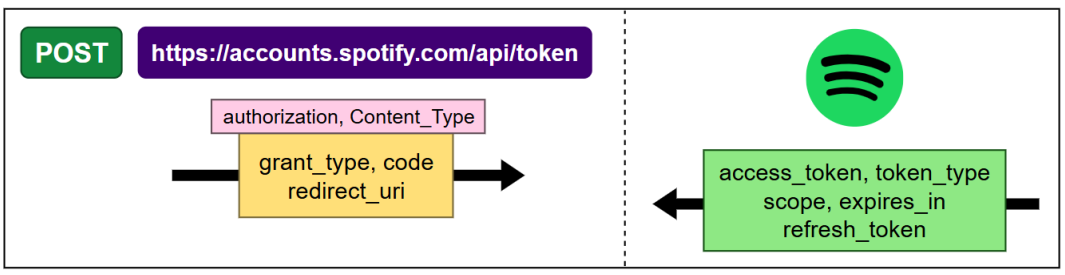

Figura 4.4: Endpoint de *Request Access Token*.

- **Parámetros del Request**
  - **Body**
    - `grant_type`: Este campo debe contener el valor "`authorization_code`".
    - `code`: El código de autorización devuelto de la solicitud previa.
    - `redirect_uri`: Este parámetro se utiliza únicamente para validación (no se realiza una redirección real). El valor debe coincidir exactamente con el valor de `redirect_uri` utilizado al solicitar el código de autorización.
  - **Headers**
    - `Authorization`: Cadena codificada en Base64 con el siguiente formato: `Basic <base64 encoded client_id:client_secret>`.
    - `Content-Type`: Establecido en "`application/x-www-form-urlencoded`".
- **Campos del Response**
  - `access_token`: Token de acceso que permite hacer las posteriores llamadas a la API.
  - `token_type`: Indica cómo se puede usar el token de acceso; siempre tiene el valor "`Bearer`".
  - `scope`: Lista de copes separados por espacios que han sido concedidos para este `access_token`.
  - `expires_in`: Periodo de tiempo (en segundos) durante el cual el token de acceso es válido. Siempre es de 1 hora.
  - `refresh_token`: Token utilizado para renovar el `access_token` cuando este expira.

4.1.3.2. Endpoints de Datos

Una vez completado el proceso de autorización y obtenido el `access_token`, la aplicación puede interactuar con los endpoints de datos proporcionados por Spotify. Hay un total de 88 endpoints agrupados en 14 grupos. En la figura 4.5 se indican los grupos cuyos endpoints se van a utilizar en este proyecto.

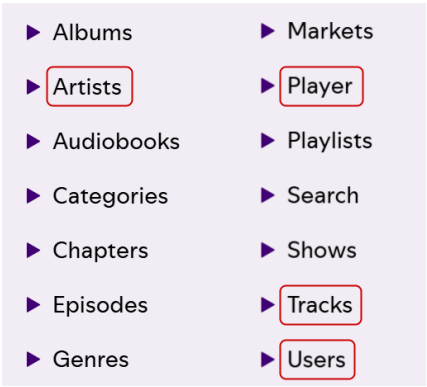

Figura 4.5: Grupos de endpoints ofrecidos y cuáles se van a usar.

Por razones prácticas, solo se van a mencionar los campos más relevantes en los *request* y *response* de los endpoints, ya que los objetos devueltos por la API suelen ser extensos y contener una gran cantidad de información, además de redundante en algunos casos. También se omite el campo de `Authorization`, que contine el `access_token` en el *header* del *request*, ya que es necesario incluirlo en todas las peticiones.

Users

En el grupo **Users** se encuentran los endpoints relacionados con la información de la cuenta del usuario. Usaremos el endpoint de **Get Current User's Profile** (figura 4.6) para obtener datos como el nombre, correo electrónico y la imagen de perfil de la cuenta. Por otro lado, el endpoint de **Get User's Top Items** permite obtener los "elementos" más escuchados por el usuario, que en este caso podemos elegir entre *tracks* (figura 4.7) o *artists* (figura 4.8).

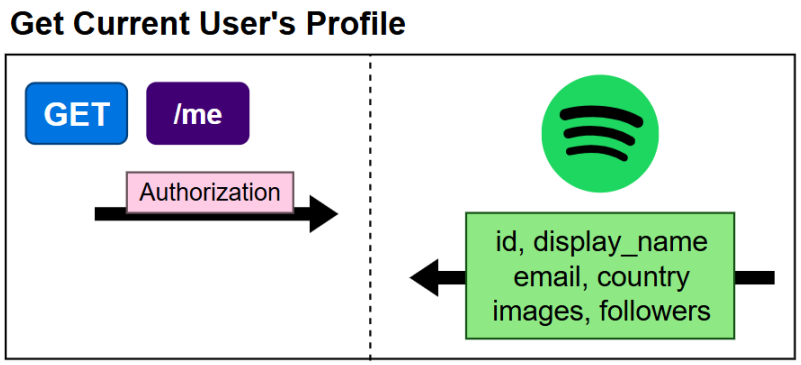

Figura 4.6: Endpoint de *Get Current User's Profile*.

- **Parámetros del Request**
  - No requiere parámetros adicionales.
- **Campos del Response**
  - `country`: El país del usuario, en formato ISO.
  - `display_name`: Nombre mostrado en el perfil del usuario.
  - `email`: Dirección de correo del usuario, ingresada al crear la cuenta.
  - `id`: ID del usuario en Spotify.
  - `images`: Array conteniendo las URLs de la imagen del perfil del usuarios en distintos tamaños.
  - `followers`: Objeto con la información sobre los seguidores del usuario.

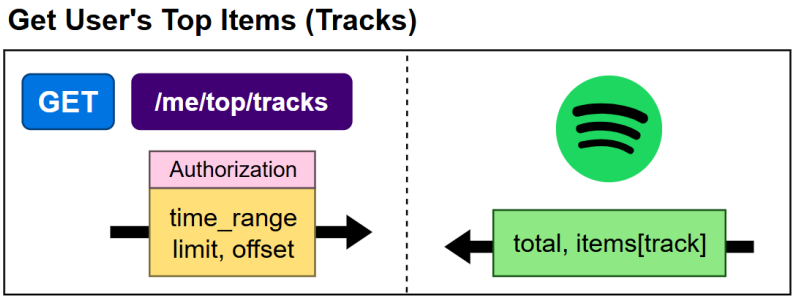

Figura 4.7: Endpoint de *Get User's Top Items (Tracks)*.

- **Parámetros del Request**
  - **Body**
    - `time_range`: El marco temporal para calcular las afinidades (`"long_term"`, `"medium_term"` (por defecto), `"short_term"`).
    - `limit`: Máximo número de elementos a devolver. Rango: 1-50.
    - `offset`: Índice del primer elemento a devolver.
- **Campos del Response**
  - `next`: URL de la página siguiente de elementos. `null` si no hay más elementos.
  - `total`: Número total de elementos disponibles.
  - `items`: Array de objetos de los datos de cada track.
    - `name`: Nombre del track.
    - `popularity`: Su popularidad (0-100).
    - `duration_ms`: Su duración en milisegundos.
    - `explicit`: Valor booleano indicando si el track contiene letras explícitas.
    - `album`: Información sobre el álbum donde aparece.
      - `id`: ID del álbum en Spotify.
      - `images`: Array conteniendo las URLs de la imagen de la portada del álbum en distintos tamaños.
      - `name`: Nombre del álbum.
      - `release_date`: Fecha de lanzamiento del álbum.
    - `artists`: Array con los objetos relacionados con los artistas que participaron en el track.
      - `name`: Nombre del artista.
      - `id`: ID del artista en Spotify.

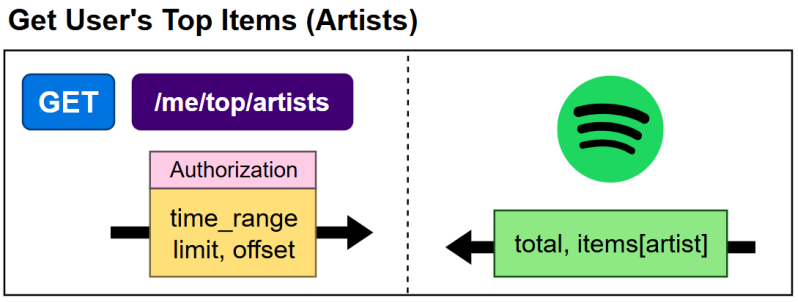

Figura 4.8: Endpoint de *Get User's Top Items (Artists)*.

- **Parámetros del Request**
  - Los mismos que en el endpoint de *Get User's Top Items (Tracks)*.
- **Campos del Response**
  - `id` (string): ID del artista en Spotify.
  - `name`: Nombre del artista.
  - `popularity`: Su popularidad (0-100).
  - `followers`: Objeto con la información sobre los seguidores del artista.
  - `genres`: Lista de géneros asociados al artista. Si el artista aún no está clasificado, el array estará vacío.
  - `images`: Array conteniendo las URLs de la imágene del artista en distintos tamaños.

Player

En el grupo **Player** podemos encontrar endpoints para controlar y obtener el estado de reproducción de la cuenta. Además de poder iniciar, pausar, adelantar o controlar el volumen de la reproducción, también podemos saber las últimas canciones escuchadas por el usuario mediante el endpoint de **Get Recently Played Tracks** (figura 4.9).

Cabe destacar que este endpoint tiene una limitación muy considerable: **solo permite obtener las últimas 50 canciones escuchadas por el usuario**. Es decir, no es posible obtener un historial de escucha en base a un periodo de tiempo establecido, ya que esas 50 canciones pueden haber sido escuchadas en un periodo largo o en un solo día, según los hábitos de escucha del usuario. Esta limitación ha afectado en el diseño de algunas estadísticas de la aplicación, en concreto aquellas que requieren conocer los gustos musicales del usuario a lo largo del tiempo.

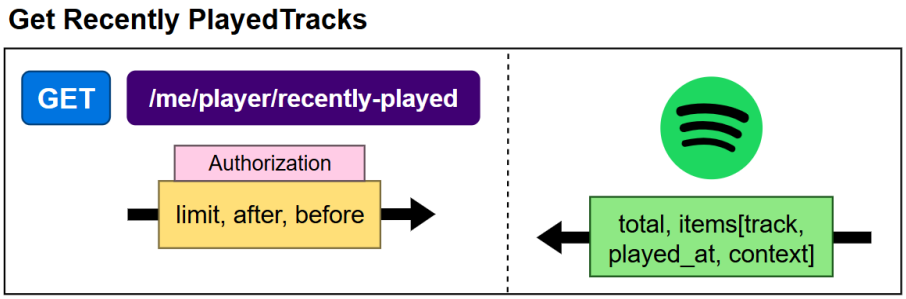

Figura 4.9: Endpoint de *Get Recently Played Tracks*.

- **Parámetros del Request**
  - **Body**
    - `limit`: Número máximo de elementos a devolver. Rango: 1-50.
    - `after`: Marca de tiempo Unix en milisegundos. Devuelve todos los elementos posteriores (excluyendo el indicado). Si se especifica `after`, no se debe especificar `before`.
    - `before`: Marca de tiempo Unix en milisegundos. Devuelve todos los elementos anteriores (excluyendo el indicado). Si se especifica `before`, no se debe especificar `after`.
- **Campos del Response**
  - `next`: URL a la página siguiente de elementos. `null` si no hay más elementos.
  - `total`: Número total de elementos disponibles.
  - `items`: Objetos conteniendo la información sobre los tracks del historial de reproducción.
    - `name`: Nombre del track.
    - `duration_ms`: Duración en milisegundos.
    - `played_at`: Fecha y hora en la que se reprodujo el track.
    - `explicit` (boolean): Indica si el track contiene contenido explícito.
    - `popularity` (integer): Popularidad del track (0-100).
    - `album`: Información sobre el álbum en el que se encuentra.
      - `name`: Nombre del álbum.
      - `release_date`: Fecha de lanzamiento.
      - `images`: Las URLs de la portada del álbum en varios tamaños.
    - `artists`: Información sobre los artistas que participaron.
      - `name`: Nombre del artista.
      - `id`: ID del artista en Spotify.

Tracks

Mediante los endpoints del grupo **Tracks**, se pueden obtener datos sobre canciones específicas. El endpoint **Get User's Saved Tracks** (figura 4.10) devuelve las canciones guardadas en la lista de favoritos del usuario.

Este endpoint, además de permitir identificar las canciones favoritas del usuario, **proporciona una alternativa para analizar sus gustos musicales a lo largo del tiempo**. Aunque no se trata de un historial de escucha, ofrece una lista de canciones que el usuario ha decidido guardar junto con la fecha correspondiente, lo que puede servir como indicativo de sus preferencias. Esta alternativa ha sido la solución adoptada para suplir la limitación mencionada en el endpoint de **Get Recently Played Tracks**. Gracias al valor proporcionado en el campo `added_at`, es posible considerar que el acto de añadir una canción a favoritos funciona como un proxy para representar los gustos musicales y las canciones escuchadas en torno a ese periodo de tiempo.

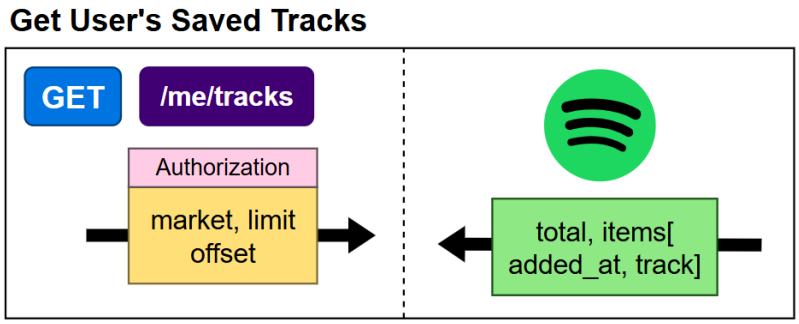

Figura 4.10: Endpoint de *Get User's Saved Tracks*.

- **Parámetros del Request**
  - **Body**
    - `market`: Código de país para indicar el mercado, en formato ISO.
    - `limit`: Número máximo de elementos a devolver. Rango: 1-50.
    - `offset`: Índice del primer elemento a devolver.
- **Campos del Response**
  - `next`: URL a la página siguiente de elementos. `null` si no hay más elementos.
  - `total`: Número total de elementos disponibles.
  - `items`: Array de objetos con la información del los tracks guardados.
    - `added_at`: Fecha y hora en la que se guardó el track en formato ISO.
    - `track`: Objeto con la información sobre el track.
      - Contiene los mismos campos que en el endpoint de *Get Recently Played Tracks*.

Artists

Al igual que el grupo anterior, en **Artists** se pueden consultar datos sobre artistas específicos. **Get Several Artists** (figura 4.11) permite obtener información sobre varios artistas a la vez, reduciendo el número de peticiones realizadas a la API.

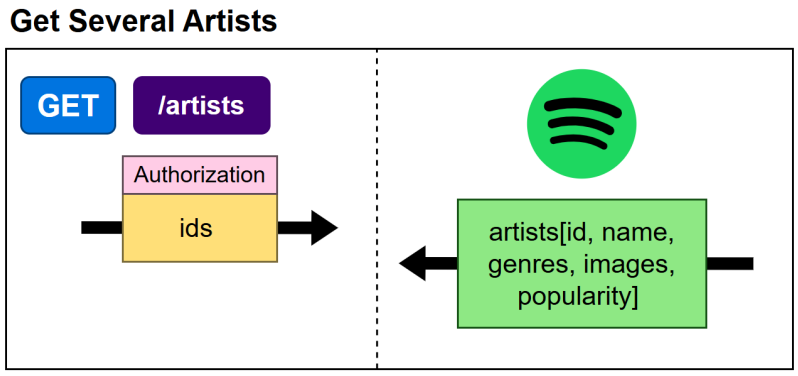

Figura 4.11: Endpoint de *Get Several Artists*.

- **Parámetros del Request**
  - **URL params**
    - `ids`: Lista separada por comas de los IDs de Spotify de los artistas.
- **Campos del Response**
  - `artists`: Lista de objetos con la información de los artistas.
    - `name`: Nombre del artista.
    - `id`: Su ID en Spotify.
    - `popularity`: Su popularidad (0-100).
    - `followers`: Información sobre sus seguidores.
    - `genres`: Lista de géneros asociados al artista.
    - `images`: Array de URLs de la imagen del artista en varios tamaños.

Los datos proporcionados por estos endpoints serán utilizados para poblar de información la aplicación, procesándolos y adaptándolos cuando sea necesario para ofrecer una experiencia personalizada al usuario. Además, estos datos influirán directamente en el diseño de la aplicación, tanto en los componentes del frontend, como en la estructura del servidor y la comunicación entre ambos.

4.1.4. Scopes Necesarios

Como ya se ha mencionado en el apartado 4.1.2, para poder tratar los datos necesarios, el usuario deber autorizar el acceso a ciertos recursos de su cuenta. Para ello, es necesario indicar los scopes adecuados al solicitar la autorización. En este proyecto, se necesitarán los siguientes scopes:

- `user-top-read`
- `user-read-private`
- `user-read-email`
- `user-read-recently-played`
- `user-library-read`

4.1.5. Limitaciones y Consideraciones

Además del límite de 50 canciones en el endpoint de **Get Recently Played Tracks** y otras limitaciones relacionadas con el acceso a datos concretos, la principal limitación impuesta por la API de *Spotify* es la **tasa de peticiones** o ***rate limit***. Esta limitación se establece para evitar la sobrecarga de los servidores y garantizar un funcionamiento estable del servicio. La tasa de peticiones de Spotify se calcula en una ventana de 30 segundos (figura 4.12). Si la aplicación supera este límite en dicho periodo, recibirá una respuesta de error `429 Too Many Requests` y los recursos solicitados quedarán temporalmente inaccesibles.

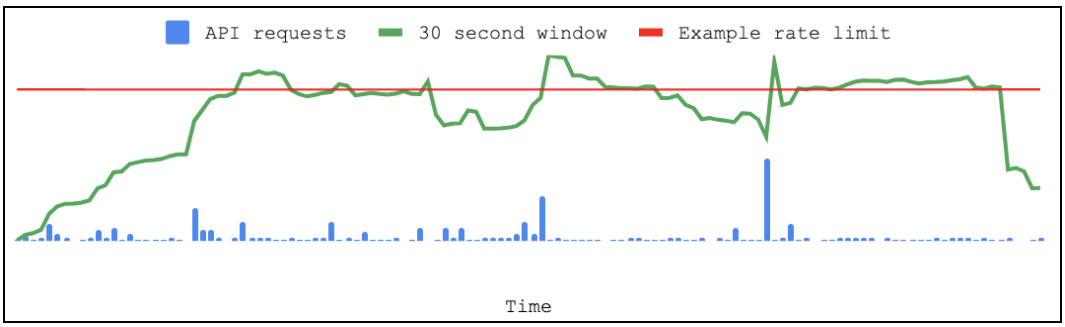

Figura 4.12: Gráfica del funcionamiento de la tasa de peticiones, obtenida de la documentación oficial de Spotify.

Para evitar alcanzar este límite, se pueden implementar técnicas para optimizar el número de peticiones. Algunas de las que se implementarán en este proyecto son:

- Batch APIs: Utilizar endpoints que permiten obtener lotes de datos en una sola petición, como el endpoint **Get Several Artists**.
- Lazy Loading: Retrasar las solicitudes de datos hasta que el usuario interactúe con un elemento específico, como al abrir una estadística.

4.1.6. Política y Términos de Uso de la API

El uso de la API de *Spotify* está regulado por un marco legal detallado, compuesto por los ***Spotify Developer Terms*** y la ***Spotify Developer Policy***. Estas normativas establecen las condiciones y restricciones bajo las cuales se puede acceder, procesar y utilizar los datos proporcionados por la API. A continuación, se analizan los aspectos que afectan el desarrollo del proyecto en el marco del TFG.

4.1.6.1. Restricciones y Obligaciones

Se subrayan dos principios básicos en sus términos: **la protección de los derechos de los usuarios** y **la garantía de que el contenido ofrecido en su plataforma está debidamente licenciado**. Esto implica que cualquier aplicación desarrollada debe respetar la privacidad de los usuarios y manejar los datos conforme a las preferencias que ellos definan. Además, está prohibido utilizar los datos o contenidos de *Spotify* para fines no autorizados, como entrenar modelos de inteligencia artificial o crear bases de datos derivadas.

El uso de la API está sujeto a restricciones específicas, que se berán de cumplir en este proyecto. Entre las más relevantes se destacan:

- **Acceso y uso de datos:** Solo se pueden solicitar los datos necesarios para operar la aplicación y deben eliminarse si un usuario decide desconectar su cuenta de *Spotify*.
- **Prohibiciones explícitas:** Está prohibido transferir datos a terceros, crear funcionalidades que permitan *stream ripping*, o desarrollar servicios destinados al uso empresarial o con contenido dirigido a menores de edad.
- **Caché local:** Aunque se permite el almacenamiento temporal de ciertos datos para mejorar el rendimiento, no se debe almacenar contenido indefinidamente.
- **Mecanismos de desconexión:** La aplicación debe ofrecer a los usuarios una forma accesible y funcional de desconectar su cuenta de *Spotify* en cualquier momento.

4.1.6.2. Protección de Datos y Seguridad

Los desarrolladores son responsables de implementar medidas de seguridad estándar para proteger los datos personales de los usuarios. Esto incluye:

- Proporcionar una política de privacidad clara y accesible que explique cómo se recopilan, procesan y comparten los datos.
- Asegurar la confidencialidad de los datos personales y notificarlos a *Spotify* en caso de incidentes de seguridad que puedan comprometer esta información.
- Respetar las solicitudes de los usuarios relacionadas con sus datos, como la rectificación o eliminación de la información.

4.1.6.3. Implicaciones para el TFG

La implementación de la aplicación debe garantizar el cumplimiento de las políticas mencionadas, especialemente la correcta gestión de los datos personales de los usuarios y el uso correcto de los recursos. En este sentido, es importante solicitar únicamente los *scopes* necesarios para las funcionalidades definidas, asegurando que cada autorización concedida por los usuarios esté justificada. También queda claro que el único uso válido del almacenamiento de los datos es el de la caché temporal en el servidor, con el propósito de optimizar el rendimiento. Además, en caso de que el usuario decida desconectar su cuenta de la aplicación, se debe implementar procesos claros y que aseguren la eliminación completa de sus datos.

Spotify se reserva el derecho de revisar las aplicaciones que acceden a su API para garantizar que cumplan con los términos establecidos. En caso de detectar alguna infracción, como el incumplimiento de la normativa descrita, pueden limitar el acceso, revocar permisos o incluso suspender completamente la funcionalidad de la aplicación. Por lo tanto, el diseño y desarrollo de la aplicación deben incorporar estas consideraciones desde el principio, para evitar cualquier tipo de conflicto con la política de *Spotify*.

4.2. Requisitos Funcionales

Los requisitos funcionales describen las funcionalidades esenciales que debe cumplir la aplicación para satisfacer las necesidades del usuario. En el caso de esta aplicación, todos los requisitos funcionales están orientados al cumplimiento del objetivo principal: permitir a los usuarios acceder al análisis de sus datos musicales cuando lo deseen. Estas funcionalidades están organizadas en diferentes categorías según su ámbito.

4.2.1. Autenticación y Sesión

- El usuario debe poder iniciar sesión utilizando su cuenta de *Spotify* mediante OAuth 2.0.
- El usuario debe poder cerrar sesión desde cualquier página.
- El cierre de sesión debe eliminar cualquier dato asociado al usuario.
- La aplicación debe obtener y gestionar adecuadamente e token de acceso para interactuar con la API de *Spotify*.
- La sesión del usuario debe permanecer activa mientras este interactúe con la aplicación web. En caso de que el tiempo de validez del token expire, el sistema debe renovar el token de forma transparente para el usuario.

4.2.2. Navegación

- La aplicación debe tener una barra de navegación con acceso a las secciones:
  - **Home:** Vista general de las estadísticas generales.
  - **Stats:** Estadísticas más avanzadas y originales.
  - **Panel de Usuario:** Panel desplegable con la opción de cerrar sesión.

4.2.3. Estadísiticas Generales (Home)

- El usuario debe ver una página inicial con estadísticas generales relacionadas con su cuenta, siendo las siguientes:
  - **Top Tracks:** Muestra los top 5 canciones del usuario.
  - **Top Artists:** Muestra los top 5 artistas del usuario.
  - **Top Genres:** Muestra los top 5 géneros del usuario,
  - **Recently Played:** Muestra una lista reducida de las últimas 10 canciones escuchadas, en orden cronológico. Si el usuario quiere, podrá ver la lista completa de las 50 canciones pulsando un botón.
- El usuario podrá cambiar el periodo de tiempo en el que se basan los datos de los tres tops mediante un menú desplegable.

4.2.4. Estadísticas Avanzadas (Stats)

En esta página, el usuario podrá explorar una gama de seis estadísticas más avanzadas. Cada estadística ofrece funcionalidades específicas, descritas a continuación:

Hall Of Fame

- La estadística mostrará las **top 16 canciones** del usuario en formato de cuadrícula (4x4), utilizando las portadas de los álbumes de cada canción como elemento visual.
- Al pasar el ratón por encima de cada una de las portadas, se mostrará el nombre de la canción y el artista.
- El usuario, mediante un botón, podrá crear una playlist de manera automática en su cuenta:
  - Se generará una nueva playlist.
  - Se añadirán las canciones del top 16.
  - Se actualizará la imagen de la playlist con la representación visual de la cuadrícula de portadas generada para esta estadística.

Huella Del Día

- El usuario verá un gráfico de líneas, donde el eje horizontal representa las horas del día y el eje vertical los minutos de música escuchados (correspondientes a 1 hora).
- Al pasar el ratón por encima de un nodo de la gráfica, se mostrará la cantidad exacta de minutos escuchados en esa hora.
- Se indica la hora del día con más minutos de escucha.

Estaciones Musicales

- Se mostrará un gráfico en forma de anillo dividido en cuatro segmentos, cada uno representando una estación del año: invierno, primavera, verano y otoño.
- Al pasar el ratón por encima de un segmento del gráfico, se abrirá un panel que mostrará, basado en la actividad del usuario, el artista y el género musical destacados durante ese periodo.
- La información en el panel se actualiza dinámicamente mientras el usuario pasa el ratón por las distintas secciones del gráfico.

Tus Décadas

- Se presentará un histograma que muestra el número de álbumes correspondientes a cada año, agrupados por décadas. La cantidad de álbumes se muestra de manera visual, mediante imágenes de portadas apiladas en columnas. Los álbumes están asociados a las canciones favoritas del usuario, mostrando una única repetición de cada álbum, independientemente de la cantidad de canciones guardadas de ese álbum.

- El gráfico es explorable, permitiendo al usuario desplazarse horizontal y verticalmente.
- El usuario podrá hacer *zoom in* y *zoom out* para ajustar el nivel de detalle, permitiéndole observar las portadas de los álbumes con mayor claridad.

La Bitácora

El usuario podrá explorar detalladamente el historial completo de sus canciones guardadas en favoritos, organizadas por fechas. Se estructura de la siguiente manera:

- Se mostrará la visualización principal; un gráfico de barras con el número total de canciones guardadas, agrupadas por años, desde la creación de la cuenta hasta la fecha actual.
- Al hacer clic en una barra que represente un año, se profundizará en el gráfico para mostrar la distribución de canciones guardadas por meses dentro de ese año.
- De forma similar, al hacer clic en una barra que represente un mes, se desglosará la información para mostrar las canciones guardadas por días dentro de ese mes específico.
- El usuario podrá navegar libremente entre los niveles (años, meses y días), pudiendo retroceder hacia niveles superiores en cualquier momento.
- Al pasar el ratón por encima de cualquier barra, se mostrará el número de canciones guardadas para ese período.
- En el nivel de días, al pasar el ratón sobre una barra, se mostrarán además los nombres y artistas de las canciones guardadas en esa fecha específica.

Índice de Resonancia

Se presentará al usuario, de forma visual, dos métricas relacionadas con sus preferencias musicales: la popularidad media de sus canciones guardadas en favoritos y la popularidad media de las últimas 50 canciones escuchadas. Estas métricas se implementarán a través de las siguientes funcionalidades:

- Cada valor estará asociado a una onda sinusoidal animada, cuya frecuencia se ajustará proporcionalmente al valor numérico correspondiente.
- El usuario podrá pulsar un botón que generará una nueva onda. Esta nueva onda se calcula mediante la suma matemática de las frecuencias de las dos ondas originales, mostrando la interferencia entre las dos. Además, junto a esta nueva onda, se mostrará la diferencia numérica entre los dos valores originales.
- El usuario podrá pasar el ratón sobre la tarjeta de cualquiera de los valores para resaltar visualmente la onda correspondiente, facilitando la identidicación.
- Los cálculos y la generación de datos necesarios para esta estadística se realizarán principalmente en el servidor.

4.3. Requisitos No Funcionales

Los requisitos no funcionales de la aplicación son fundamentales para garantizar no solo su funcionalidad, sino también su rendimiento, seguridad, escalabilidad y usabilidad. A continuación, se detallan los aspectos clave que la aplicación debe cumplir para asegurar una experiencia de usuario satisfactoria y el cumplimiento de los estándares de calidad.

Rendimiento y Escalabilidad

La aplicación debe ofrecer tiempos de respuesta aceptables en las operaciones críticas, como la carga inicial del *Home* o la presentación de las estadísticas. El tiempo de respuesta debe ser inferior a **1 s** bajo condiciones normales de uso, con un máximo de **3 s** bajo picos de carga. Este requisito se medirá utilizando herramientas de pruebas de carga como K6, simulando hasta **50 usuarios simultáneos** realizando operaciones comunes. El objetivo es garantizar que el rendimiento del sistema satisfactorio para los usuarios, incluso en picos de carga inusuales.

Seguridad

Es esencial garantizar la protección de los datos del usuario y las comunicaciones entre el frontend y el backend. Para ello, todas las conexiones deben utilizar **HTTPS**, y los tokens de acceso deben almacenarse de forma segura para prevenir ataques comunes como *Cross-Site Scripting* (XSS) y *Man-in-the-Middle* (MITM). Además, es necesario implementar medidas de seguridad en procesos críticos, como la autenticación de usuarios, incluyendo protección frente a ataques de tipo *Cross-Site Request Forgery* (CSRF).

Testabilidad

Para garantizar la calidad del sistema, se deben implementar pruebas automatizadas que cubran las funcionalidades clave de la aplicación, priorizando las partes críticas del backend y las interacciones más relevantes del frontend. Esto incluirá pruebas unitarias y de integración. La cobertura mínima objetivo será del **60 % del código**, con un enfoque en las secciones esenciales. Las pruebas se ejecutarán regularmente mediante integración continua (CI) utilizando GitHub Actions.

Usabilidad

La aplicación debe ser intuitiva y fácil de usar para el público general. Todas las estadísticas y funcionalidades deben presentarse de manera clara. Este requisito se evaluará mediante pruebas de usabilidad con al menos **5 usuarios representativos**, asegurando que puedan completar tareas concretas sin dificultad ni confusión.

4.4. Casos de Uso

En esta sección se describen los principales casos de uso de la aplicación, relacionados con los requisitos funcionales mencionados. Cada caso de uso se enfoca en un objetivo específico que un usuario puede alcanzar utilizando las funcionalidades del sistema.

4.4.1. Actores

En este sistema, interactúan tres actores diferentes con la aplicación: el usuario anónimo, el usuario autenticado y la API de *Spotify*. A continuación, se describen los roles de cada uno:

- **Usuario Anónimo:** Este actor representa a los usuarios que no han iniciado sesión en la aplicación. Su única interacción con el sistema es el de inicio de sesión.
- **Usuario Autenticado:** Este actor es el usuario que ha iniciado sesión correctamente en la aplicación. Tiene acceso a todas las funcionalidades. Es considerado como el actor principal.
- **Web API de Spotify:** Este actor es el servicio de *Spotify* que proporciona un sistema de autenticación y los datos necesarios para generar las estadísticas. Interactúa principalmente con el backend de la aplicación.

4.4.2. Modelo de Casos de Uso

En el modelo presentado en la figura 4.13, se destacan las funcionalidades principales de la aplicación y su relación con los actores: **Usuario Anónimo** y **Usuario Autenticado**, que interactúan directamente con el sistema, y la **API de Spotify**, que actúa como proveedor de datos.

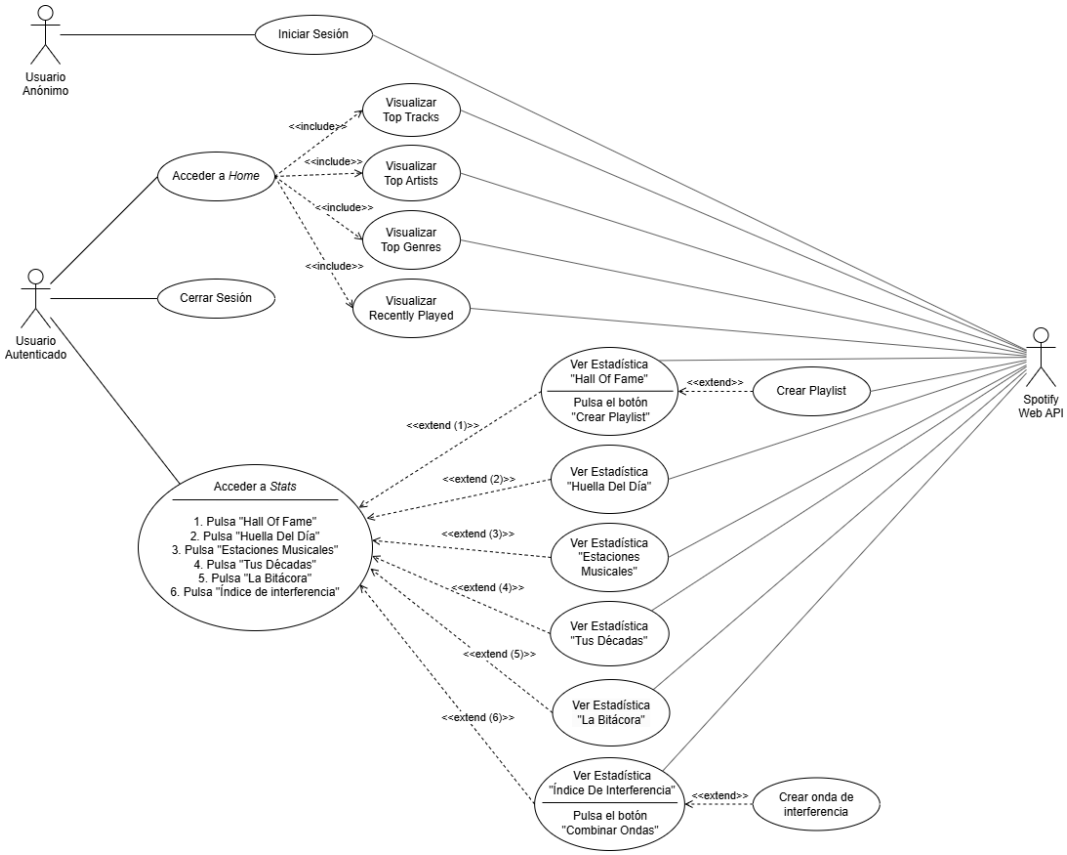

Figura 4.13: Modelo de casos de uso del sistema.

Iniciar Sesión

Permite al usuario anónimo iniciar sesión en la aplicación utilizando sus credenciales de *Spotify* a través del sistema de autenticación OAuth 2.0.

- Flujo principal (Inicio de Sesión):
  1. El usuario anónimo selecciona la opción de "Iniciar Sesión" en la pantalla principal.
  2. El sistema redirige al usuario a la página de inicio de sesión de *Spotify*.
  3. El usuario introduce sus credenciales de *Spotify* en el formulario proporcionado por *Spotify*.
  4. *Spotify* verifica las credenciales y, si son correctas, muestra la pantalla de solicitud de autorización.
  5. El usuario autoriza a la aplicación a acceder a sus datos de *Spotify* indicados.
  6. *Spotify* genera un token de acceso y redirige al usuario de vuelta a la aplicación.
  7. El sistema recibe el token de acceso y lo almacena de forma segura.
  8. El sistema redirige al usuario autenticado a la pantalla principal de la aplicación (*Home*).
- Flujo alternativo (Error de Autenticación):
  1. El usuario introduce credenciales incorrectas en la página de inicio de sesión de *Spotify*.
  2. *Spotify* rechaza las credenciales y muestra un mensaje de error al usuario.
  3. El sistema redirige al usuario de vuelta a la pantalla principal, donde podrá volver a intentar iniciar sesión.
- Flujo alternativo (Denegación de Autorización):
  1. *Spotify* verifica las credenciales del usuario y muestra la pantalla de solicitud de autorización.
  2. El usuario decide no conceder la autorización y selecciona la opción de "Cancelar".
  3. *Spotify* informa al sistema de que la autorización ha sido rechazada.
  4. El sistema redirige al usuario de vuelta a la pantalla principal.

Acceder a Home

Permite al usuario autenticado visualizar la pantalla principal de la aplicación, donde se muestran las estadísticas básicas de *Top Tracks*, *Top Artists*, *Top Genres* y *Recently Played*.

- Flujo principal (Visualización de Home):
  1. El usuario autenticado accede a la aplicación tras iniciar sesión correctamente.
  2. El sistema solicita los datos relevantes de la API de *Spotify* para generar las estadísticas.
  3. La API de *Spotify* responde con los datos necesarios para cada estadística.
  4. El sistema procesa y organiza los datos para presentarlos en el frontend.
  5. El sistema muestra al usuario las siguientes secciones en la pantalla principal:
     - *Top Tracks*: Canciones más escuchadas por el usuario.
     - *Top Artists*: Artistas más escuchados.
     - *Top Genres*: Géneros favoritos.
     - *Recently Played*: Lista de reproducción reciente.
  6. El usuario puede cambiar el periodo de tiempo de las estadísticas de los tres *Tops* seleccionando entre los últimos 30 días, 6 meses o 1 año.
  7. El usuario puede expandir la lista de *Recently Played* para ver hasta 50 canciones en lugar de las 10 iniciales, pulsando un botón de "Ver más".
  8. El usuario puede contraer nuevamente la lista de *Recently Played* para reducirla a 10 canciones, pulsando un botón de "Ver menos".
- Flujo alternativo (Error al obtener datos de la API):
  1. El sistema solicita datos a la API de *Spotify*, pero ocurre un error en la comunicación.
  2. El sistema muestra un mensaje de error al usuario, indicando que no se pudieron cargar algunas estadísticas.
  3. El usuario puede intentar recargar la página o esperar a que el sistema reintente la solicitud.

Cerrar Sesión

Permite al usuario autenticado cerrar su sesión en la aplicación, eliminando cualquier dato relacionado con su sesión activa.

- Flujo principal (Cerrar Sesión):
  1. El usuario autenticado selecciona la opción de “Cerrar Sesión” en la interfaz de la aplicación.
  2. El sistema elimina todas las cookies relacionadas con la sesión, incluido el *access_token* del usuario.
  3. El sistema borra cualquier dato del usuario almacenado temporalmente en caché.
  4. El sistema redirige al usuario a la página de inicio de sesión.

Acceder a Stats

Permite al usuario autenticado acceder a la sección de estadísticas avanzadas de la aplicación, donde puede interactuar con diferentes visualizaciones de sus datos musicales.

- Flujo principal (Acceso a Stats):
  1. El usuario autenticado selecciona la opción de "Stats" en la barra de navegación.
  2. El sistema muestra una pantalla con las opciones de estadísticas disponibles:
     - **Hall of Fame.**
     - **Huella del Día.**
     - **Estaciones Musicales.**
     - **Tus Décadas.**
     - **La Bitácora.**
     - **Índice de Interferencia.**
  3. El usuario selecciona una estadística específica para interactuar con ella.
  4. El sistema carga la visualización correspondiente.

Para todas las siguientes estadísticas avanzadas, se sigue un flujo alternativo cuando ocurre algún error en la carga. Para evitar repetir la misma descripción, a continuación se mencionará el flujo alternativo. En cada estadística, solo se describirá el flujo principal.

- Flujo alternativo (Error en la carga de datos de una estadística avanzada):
  1. El sistema solicita a la API de *Spotify* los datos necesarios para generar la estadística.
  2. La API responde con un error, ya sea por una conexión fallida, un token de acceso inválido o una respuesta incompleta.
  3. El sistema muestra un mensaje de error en la pantalla, indicando que no se pudo cargar la estadística.

Ver Estadística: Hall of Fame

Permite al usuario visualizar las 16 canciones más destacadas según su actividad, representadas en forma de cuadrícula con las portadas de los álbumes correspondientes.

- Flujo principal (Visualización de Hall of Fame):
  1. El usuario selecciona la opción "Hall of Fame" en la pantalla de *Stats*.
  2. El sistema solicita a la API de *Spotify* las 16 canciones más destacadas del usuario.
  3. La API responde con los datos necesarios, incluyendo nombres, artistas y portadas de los álbumes.
  4. El sistema genera una cuadrícula con las portadas de los álbumes correspondientes.

5. El usuario puede pasar el ratón sobre una portada para visualizar el nombre de la canción y el artista.
6. Si el usuario pulsar el botón "Crear Playlist", se envía la portada generada y los datos de las canciones a la API de *Spotify* para crear una nueva playlist.

Ver Estadística: Huella del Día

Permite al usuario visualizar un gráfico de línea que representa los minutos escuchados por cada hora del día, mostrando las horas de mayor actividad musical.

- Flujo principal (Visualización de Huella del Día):
  1. El usuario selecciona la opción "Huella del Día" en la pantalla de *Stats*.
  2. El sistema solicita a la API de *Spotify* los datos de escucha del usuario necesarios para calcular los minutos por hora.
  3. La API responde con los datos correspondientes.
  4. El sistema genera el gráfico de línea.
  5. El usuario puede pasar el ratón sobre los puntos de la línea para ver los minutos escuchados en una hora específica.
  6. El sistema destaca visualmente la hora con más minutos escuchados.

Ver Estadística: Estaciones Musicales

Permite al usuario visualizar, de forma gráfica, el artista y género más representativo de cada estación del año según su actividad musical.

- Flujo principal (Visualización de Estaciones Musicales):
  1. El usuario selecciona la opción "Estaciones Musicales" en la pantalla de *Stats*.
  2. El sistema solicita a la API de *Spotify* los datos necesarios para calcular el artista y género destacados de cada estación.
  3. La API responde con los datos correspondientes.
  4. El sistema calcula los datos y los agrupa por estaciones.
  5. El sistema genera un gráfico de tipo anillo dividido en cuatro segmentos, representando cada estación del año.
  6. El usuario puede pasar el ratón sobre uno de los segmentos del gráfico para que el sistema muestre, en un panel adicional, el artista y género destacados de la estación seleccionada.
  7. El sistema actualiza el panel dinámicamente según el segmento que el usuario seleccione con el ratón.

Ver Estadística: Tus Décadas

Permite al usuario visualizar en formato de histograma los álbumes de sus canciones favoritas organizados por décadas, destacando las épocas más representativas de su actividad musical.

- Flujo principal (Visualización de Tus Décadas):
  1. El usuario selecciona la opción "Tus Décadas" en la pantalla de *Stats*.
  2. El sistema solicita a la API de *Spotify* los datos de las canciones guardadas por el usuario en su biblioteca.
  3. La API responde con los datos, incluyendo las fechas de lanzamiento de los álbumes asociados a las canciones.
  4. El sistema organiza los datos por décadas y genera el histograma correspondiente.
  5. El usuario puede desplazarse sobre el gráfico y explorar con mayor detalle las portadas haciendo clic sobre el gráfico para ampliar o reducir el aumento.

Ver Estadística: La Bitácora

Permite al usuario explorar el historial completo de sus canciones guardadas en favoritos, organizadas cronológicamente.

- Flujo principal (Visualización de La Bitácora):
  1. El usuario selecciona la opción "La Bitácora" en la pantalla de *Stats*.
  2. El sistema solicita a la API de *Spotify* los datos de las canciones guardadas en favoritos, incluyendo la fecha en que fueron añadidas.
  3. La API responde con los datos correspondientes.
  4. El sistema genera un gráfico de barras que muestra, inicialmente, el número de canciones guardadas por año.
  5. El usuario puede interactuar con el gráfico para explorar diferentes niveles de detalle:
     - Al hacer clic en una barra que representa un año, el sistema actualiza el gráfico para mostrar los meses dentro de ese año.
     - Al hacer clic en una barra que representa un mes, el sistema desglosa la información para mostrar los días dentro de ese mes.
  6. Al pasar el ratón sobre cualquier barra, el sistema muestra un tooltip con el número de canciones guardadas en ese periodo.
  7. En el nivel de días, al pasar el ratón sobre una barra, el sistema muestra los nombres y artistas de las canciones guardadas en esa fecha específica.
  8. El usuario puede navegar libremente entre los niveles (años, meses y días) y retroceder en cualquier momento para cambiar de periodo.

Ver Estadística: Índice de Interferencia

Permite al usuario comparar el cambio en sus gustos sobre la popularidad de las canciones que escucha.

- Flujo principal (Visualización de Índice de Interferencia):
  1. El usuario selecciona la opción "Índice de Interferencia" en la pantalla de Stats.
  2. El sistema solicita a la API de *Spotify* los datos necesarios.
  3. La API responde con los datos correspondientes.
  4. El sistema hace el cálculo de las dos popularidades medias y los presenta visualmente en forma de ondas sinusoidales animadas.
  5. Si el usuario pulsa un botón de "Combinar Ondas", el sistema genera una nueva onda, resultado de la suma matemática de las frecuencias de las dos ondas originales (interferencia).
  6. El sistema muestra esta nueva onda junto a la diferencia numérica entre las dos popularidades.

Capítulo 5

Diseño

5.1. Arquitectura del Sistema

La arquitectura del sistema se basa por completo en el uso de *Next.js*, un framework de *React* que permite la creación de aplicaciones web, con la funcionalidad clave de un backend integrado. Desde la versión 13 de *Next.js* se introdujo el concepto de **App Router**, que crea las rutas de la web en base a la organización de las carpetas dentro del proyecto. Junto a esto, se introdujeron los **Route Handlers**, que permiten la creación de endpoints API REST de la misma manera, creando un backend integrado que hace de intermediario entre el cliente y los servicios externos. De esta manera, se consigue una arquitectura notablemente más simplificada, además de segura, ya que se puede controlar la exposición de datos sensibles al cliente.

Otra caracterísitica muy útil de *Next.js*, son los **Server Components**, que permiten renderizar componentes en el servidor en lugar del cliente. Solo aquellos componentes que sean necesarios serán enviados y ejecutados en el cliente, mejorando el rendimiento y la seguridad. Estos componentes pueden realizar llamadas a los endpoints proporcionados por los *Route Handlers*, recreando los roles de un sistema cliente-servidor tradicional. En el diagrama 5.1 se pueden ver representadas estas interacciones.

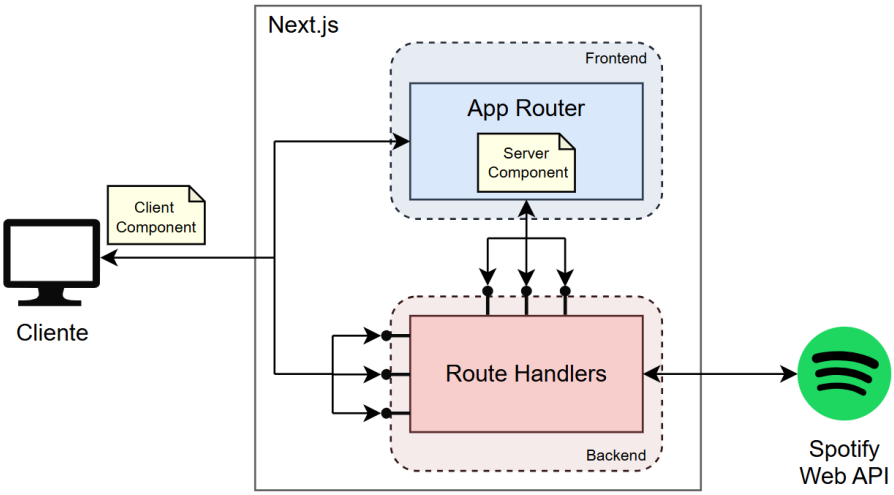

Figura 5.1: Diagrama de la arquitectura del sistema haciendo uso de *Next.js*.

5.1.1. Rutas del Frontend

Usando el *App Router*, las rutas del frontend se organizan en la carpeta `app/`, donde cada subcarpeta representa una ruta específica en la web. Dentro de cada carpeta se pueden encontrar archivos específicos que siguen una convención de nombres, y que definen su función. Las más importantes son:

- `layout.tsx`: Define el diseño compartido de los componentes que se encuentran anidados en las subcarpetas interiores.
- `page.tsx`: Contiene el contenido principal de una ruta específica. Representa la página renderizada cuando el usuario accede a esa ruta. **Para que una página sea accesible, debe existir un archivo `page.tsx` en la carpeta correspondiente**.
- `loading.tsx`: Muestra un indicador de carga mientras se obtienen datos o se renderizan componentes en una página.
- `error.tsx`: Gestiona errores específicos de una página, mostrando mensajes o interfaces para el usuario en caso de fallos.

En el caso de este proyecto, la jerarquía de carpetas que genera las páginas routeables de la web es la siguiente:

```
app/
├── home/
│   └── page.tsx (Ruta: /home)
├── stats/
│   └── page.tsx (Ruta: /stats)
└── page.tsx (Ruta: /)
```

La ruta `/` (raíz) es la página principal, en donde el usuario puede iniciar sesión. Las rutas `/home` y `/stats` representan las páginas de estadísticas básicas y estadísticas avanzadas, respectivamente. En cada caso, el archivo `page.tsx` define el contenido principal de la página y es posible añadir otros archivos como `layout.tsx`, `loading.tsx` o `error.tsx` para mejorar la experiencia del usuario.

5.1.2. Endpoints del Backend

En el caso de la creación de endpoints mediante los *Route Handlers*, la estructura de carpetas es similar a la de las rutas del frontend, pero con la diferencia de que cada subcarpeta dentro de `app/api/` contiene un archivo `route.ts` que define el comportamiento del endpoint asociado. **Para que un endpoint sea accesible, debe existir un archivo `route.ts` en la carpeta correspondiente**.

La estructura de carpetas para los endpoints de este proyecto es la siguiente:

```
app/
└── api/
    ├── auth/
    │   ├── callback/route.ts
    │   ├── login/route.ts
    │   └── logout/route.ts
    ├── home/
    │   ├── recently-played/route.ts
    │   ├── top-artists/route.ts
    │   ├── top-genres/route.ts
    │   ├── top-tracks/route.ts
    │   └── user-profile/route.ts
    └── stats/
        ├── estaciones-musicales/route.ts
        ├── hall-of-fame/route.ts
        ├── huella-del-dia/route.ts
        ├── indice-de-resonancia/route.ts
        ├── la-bitacora/route.ts
        └── tus-decadas/route.ts
```

Los endpoints en `/api/auth/` gestionan el incio de sesión del usuario, mientras que los de `/api/home/` y `/api/stats/` se encargan de obtener y procesar los datos necesarios para las estadísiticas. El contenido del fichero `route.ts` sigue una convención concreta que *Next.js* reconoce y utiliza para gestionar las peticiones, la cual se explicará con más detalle en el capítulo de Implementación.

5.2. Diagrama de Componentes de React

Los componentes en *React* son las piezas fundamentales para construir interfaces de usuario. Cada componente puede ser reutilizable, contener su propio estado y lógica, y ser combinado con otros para formar estructuras más complejas. Para describir estas interfaces de manera declarativa, *React* utiliza **JSX** (JavaScript XML), una extensión de sintaxis de JavaScript, que permite escribir el código de los componentes con una sintaxis similar a *HTML*. Aunque no sea obligatorio, su uso es ampliamente adoptado en proyectos *React*.

Las interfaces se suelen diseñar en forma de jerarquías de componentes, en las que los componentes "padre" organizan y controlan el comportamiento y la presentación de los componentes "hijo". Esta organización jerárquica facilita el mantenimiento y la escalabilidad del código, ya que cada componente tiene una responsabilidad definida. En este proyecto, se ha creado una jerarquía de componentes (diagrama 5.2) que refleja las diferentes funcionalidades de la aplicación. A la hora de construir la web, *Next.js* y *React* se encargan de anidar los componentes necesarios y renderizarlos, o enviarlos al cliente, según sea necesario.

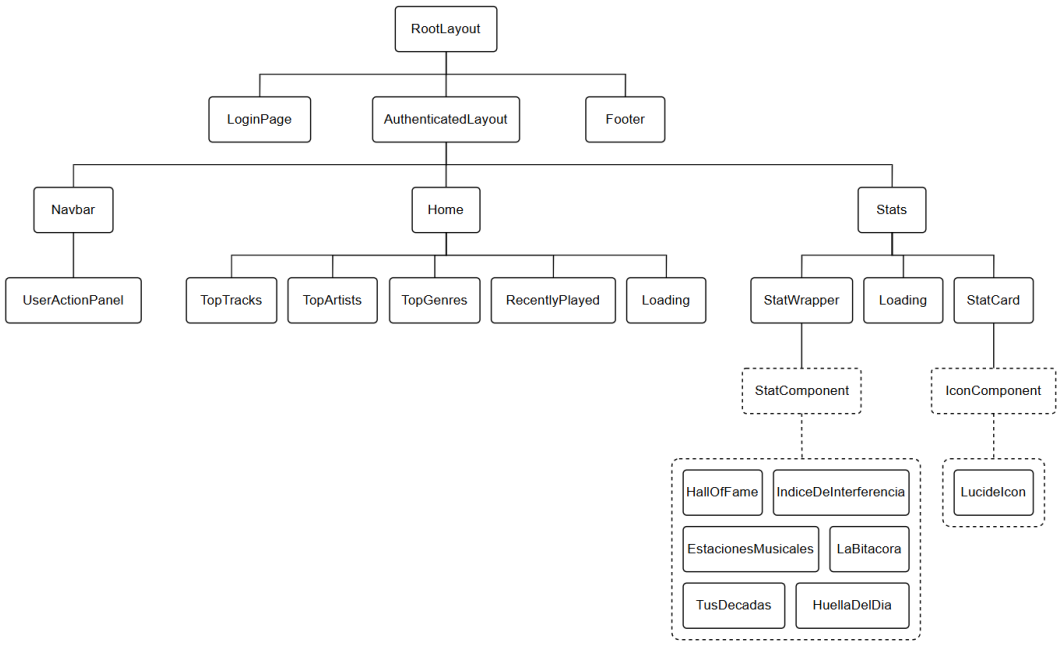

Figura 5.2: Diagrama de la jerarquía de componentes de React creados en el proyecto.

La mayoría de los componentes están definidos de manera explícita y son renderizados cuando se llegue a una ruta en concreto, pero hay otros componentes que se cargan dinámicamente, es decir, el contenido que aparece en su lugar puede variar en función de las acciones del usuario o del contexto del componente. Se podría considerar que el componente hijo temporal, se "transforma" en otro que toma su lugar. En este proyecto, se han definido dos componentes que se comportan así:

- **StatComponent**: Se utiliza para mostrar una de las seis estadísticas disponibles, cada una siendo un componente autocontenido. Dependiendo de la selección del usuario, se cargará uno de estos subcomponentes en el lugar correspondiente. También se puede inferir que solo se podrá mostrar una estadística a la vez.
- **IconComponent**: Está asociado con los iconos representativos de cada estadística, que se muestran en las tarjetas (`StatCard`) que el usuario puede clicar para verla. Utiliza el paquete `LucideIcon` para cargar diferentes iconos, ya que, en este paquete, cada icono es representado como un componente de *React*.

Con este diseño, en especial con el componente `StatComponent`, se reduce la cantidad de código a renderizar y enviar al cliente, ya que solamente se cargará la estadística necesaria y cuando el usuario lo solicite. Además, en el desarrollo se puede encapsular toda la lógica necesaria para cada estadística en un único componente, facilitando la incorporación de nuevas estadísticas, si hace falta.

5.3. Interfaz de Usuario

La interfaz de usuario es uno de los aspectos más flexibles dentro del desarrollo de software, ya que permite una gran libertad creativa y de implementación. En este proyecto, se plantearon varias opciones: utilizar una base de diseño preexistente, como guías visuales o sistemas de diseño consolidados, o desarrollar una interfaz completamente personalizada. Debido a que este trabajo pertenece al ámbito de la informática y no del diseño gráfico, muchas decisiones se orientaron hacia la practicidad, priorizando herramientas y enfoques que permiten simplificar el desarrollo, sin sacrificar la experiencia de usuario. A continuación, se describen las decisiones tomadas al respecto.

5.3.1. Principios de Diseño

Como *Spotify* es una plataforma ampliamente conocida y tiene una identidad visual muy reconocible, se decidió inspirar la interfaz en su diseño. La plataforma propociona una guía para los desarrolladores que incluye recomendaciones muy útiles [1]. De entre ellas, se han seleccionado aquellos elementos que resultan relevantes para el uso y alcance de esta aplicación, que se sintetizan en los siguientes puntos:

Colores

El principal identificador visual de *Spotify* es su color verde, llamado **Spotify Green**. Este color es el que se ha utilizado como el color primario para la página web. Además, se han incorporado el blanco y negro oficiales y también algunos colores complementarios, como el zul y el rojo, para aportar variedad en las situaciones en las que se necesite.

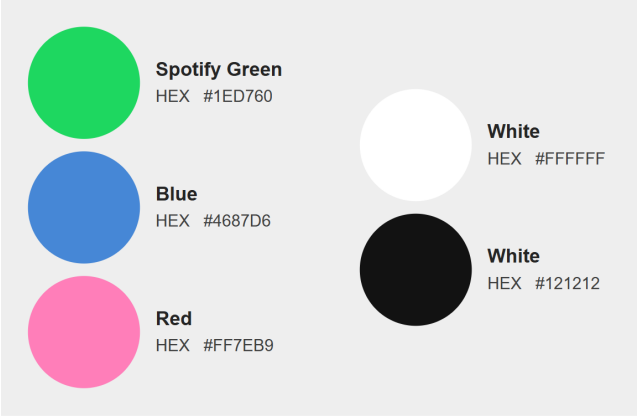

Figura 5.3: Colores seleccionados para la paleta de colores de la aplicación.

Estas combinaciones respetan las recomendaciones de la plataforma, que desaconseja el uso de colores sobresaturados, y también se ha ajustado claridad de los colores en ciertas ocasiones para mejorar el contraste en la lectura de texto.

Tipografía

Aunque *Spotify* no proporciona acceso público a su tipografía ofical, sugieren usar fuentes sans-serif como *Helvetica* o *Arial*. En el caso de este proyecto, se ha optado por hacer uso de la tipografía **Inter** (figura 5.4), ya que se asemeja más a la original que las recomendadas. Podemos encontrarla en *Google Fonts*, pero *Next.js* nos evita la necesidad de tener que descargarla, ya que está incluida de forma predeterminada en el framework.

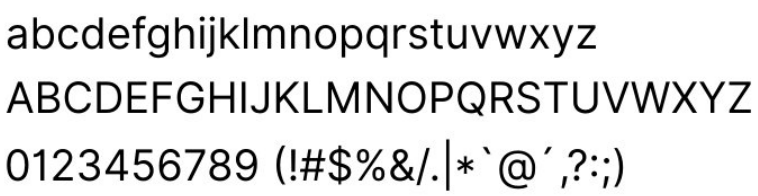

Figura 5.4: Muestra de caracteres de la tipografía *Inter*.

5.3.2. Tailwind CSS

Para el estilizado de los elementos de la interfaz se ha utilizado *Tailwind CSS*, una librería de utilidades que permite desarrollar estilos de manera rápida y eficiente. En vez de crear un estilo agrupado que se asocie a cada elemento, el enfoque de *Tailwind* se basa en agregar pequeñas clases utilitarias predefinidas a cada elemento mediante el atributo `className` en JSX (figura 5.1), aplicandole así el resultado de la combinación de todos esos estilos.

```
1    <main className="flex flex-col items-center justify-center">
2    <h1 className="text-2xl font-bold mb-4">Bienvenido</h1>
3    <a href="/api/auth/login"
4        className="px-4 py-2 bg-green-500 text-white rounded">
5    Sign In </a>
6    </main>
```

Algoritmo 5.1: Ejemplo de aplicación de estilos a un componente JSX usando *Tailwind CSS*.

De esta manera se evita la necesidad de definir manualmente grandes cantidades de estilos en archivos CSS. Otra ventaja que aporta *Tailwind* es el diseño responsivo, que viene integrado de serie en las clases definidas. Esto facilita la creación de interfaces adaptables a distintos tamaños de pantalla, sin necesidad de escribir reglas adicionales.

Otra razón por la que se ha decidido utilizar este sistema, es su integración con *React* y *Next.js*. Es un stack de tecnologías frontend muy popular, por lo que existe mucha documentación al respecto y se garantiza una gran compatibilidad. Además, *Tailwind* es muy flexible y permite personalizar los estilos de manera sencilla para ajustarlos a cualquier tipo de proyecto.

5.3.3. Páginas y Gráficos

Por completar.

5.4. Diagramas de Secuencia

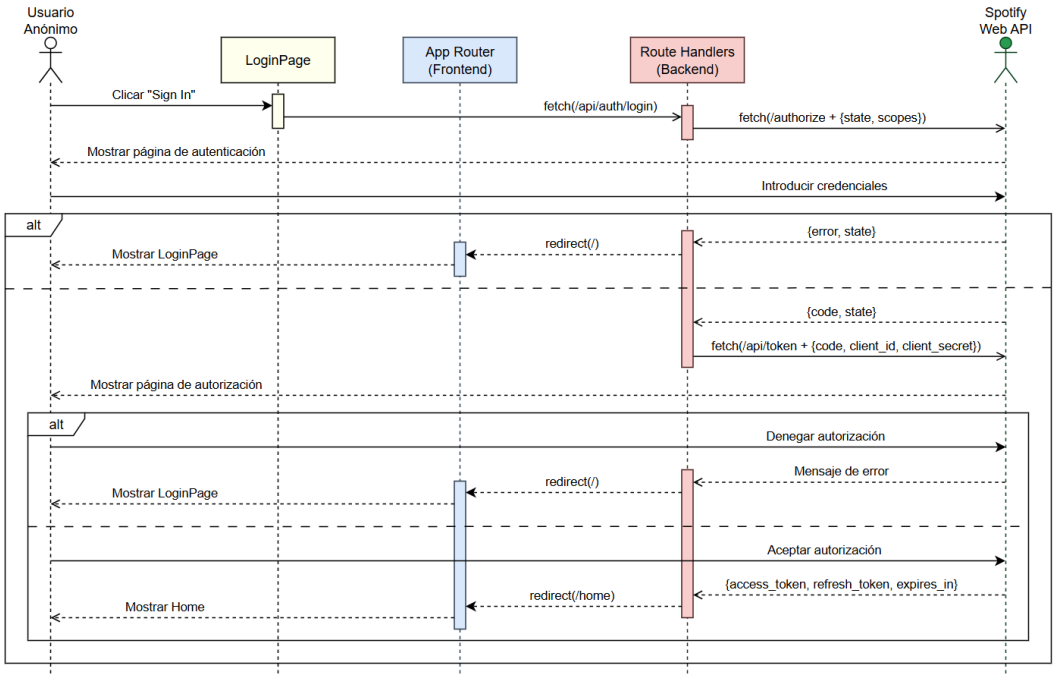

Figura 5.5: Diagrama de secuencia: Iniciar Sesión.

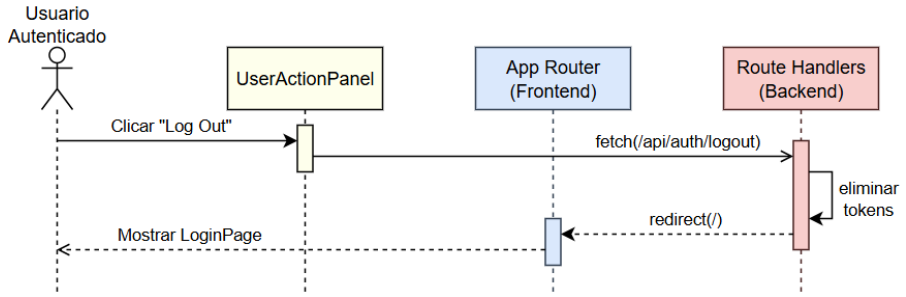

Figura 5.6: Diagrama de secuencia: Cerrar Sesión.

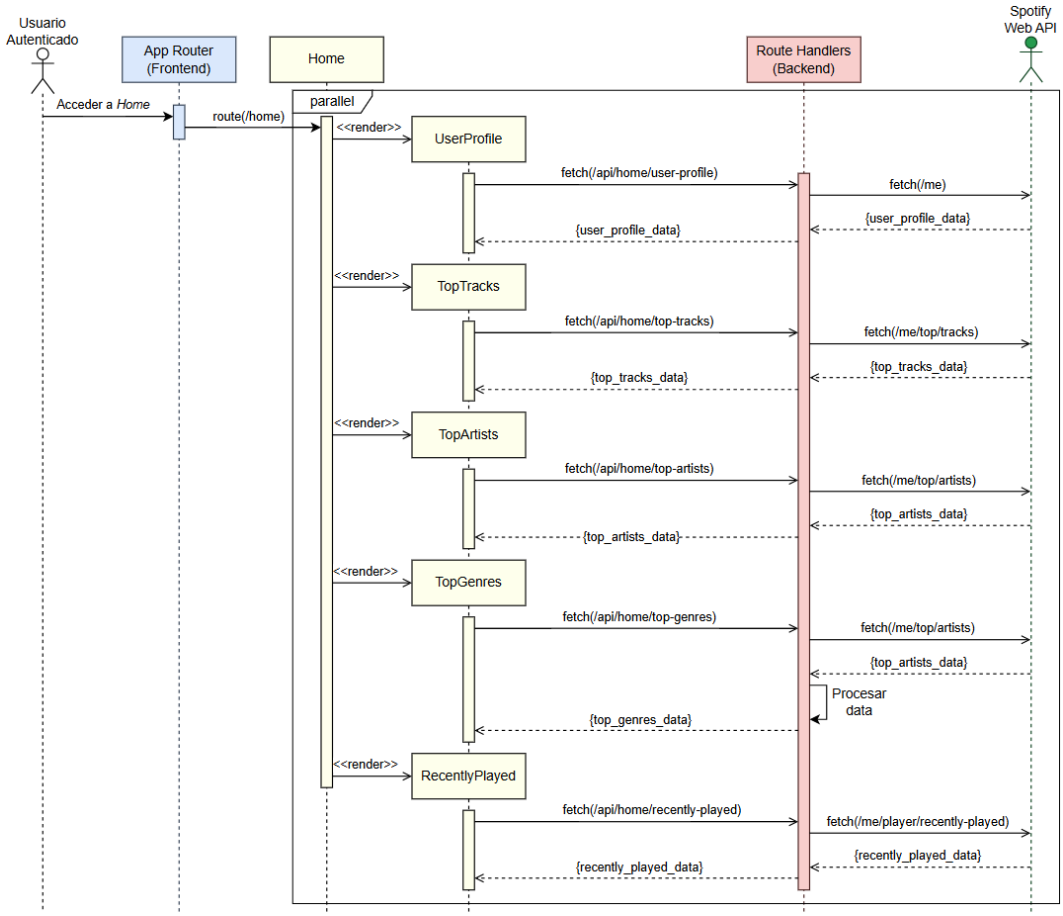

Figura 5.7: Diagrama de secuencia: Acceder a *Home*.

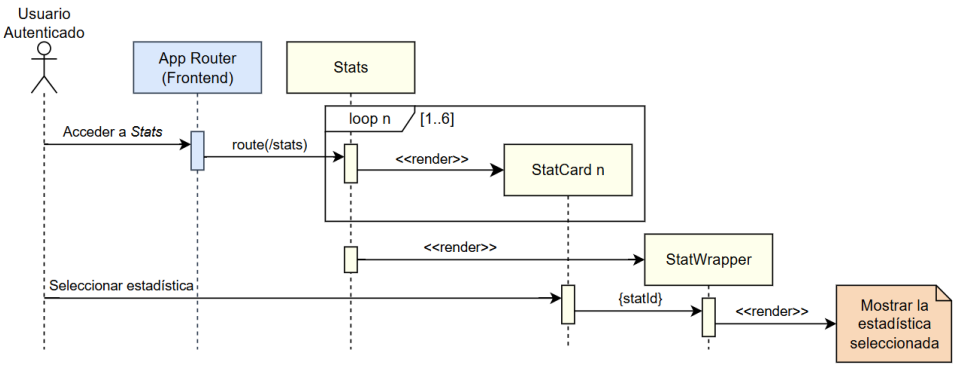

Figura 5.8: Diagrama de secuencia: Acceder a *Stats*.

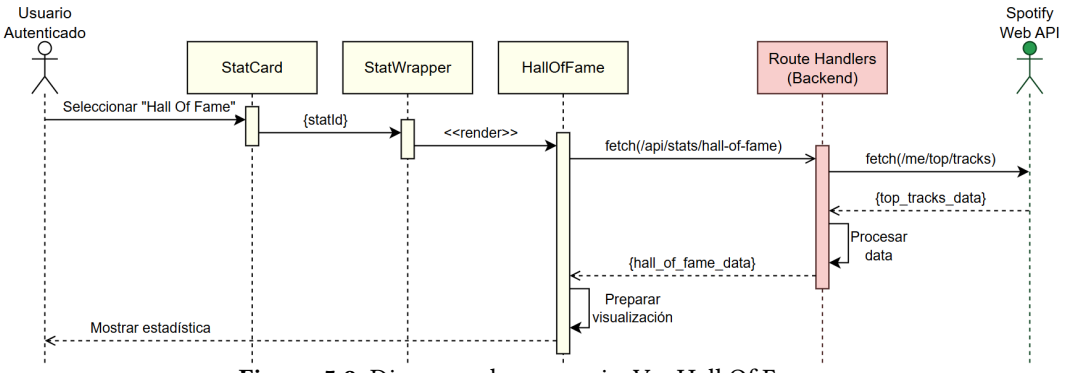

Figura 5.9: Diagrama de secuencia: Ver Hall Of Fame.

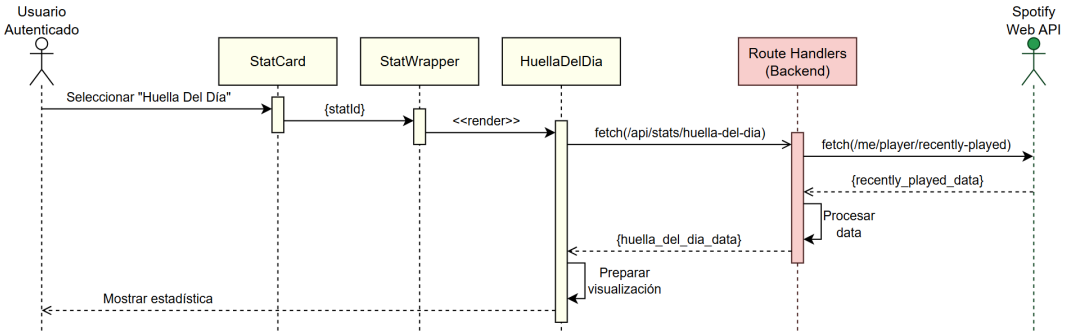

Figura 5.10: Diagrama de secuencia: Ver Huella Del Día.

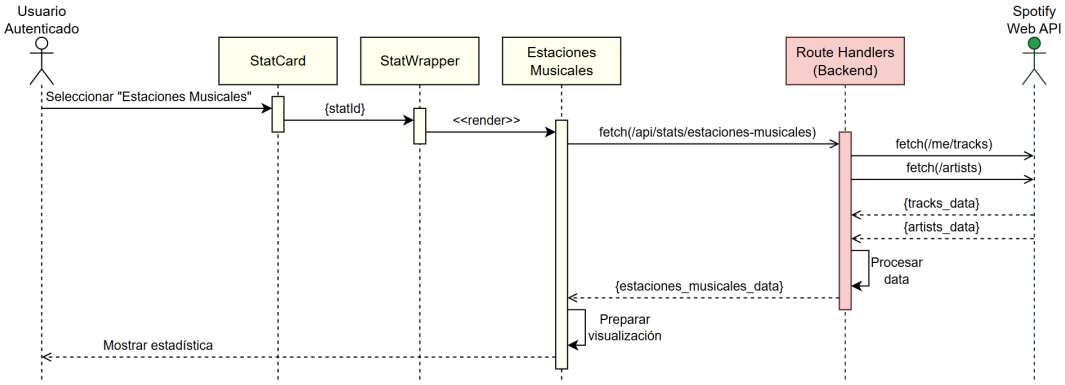

Figura 5.11: Diagrama de secuencia: Ver Estaciones Musicales.

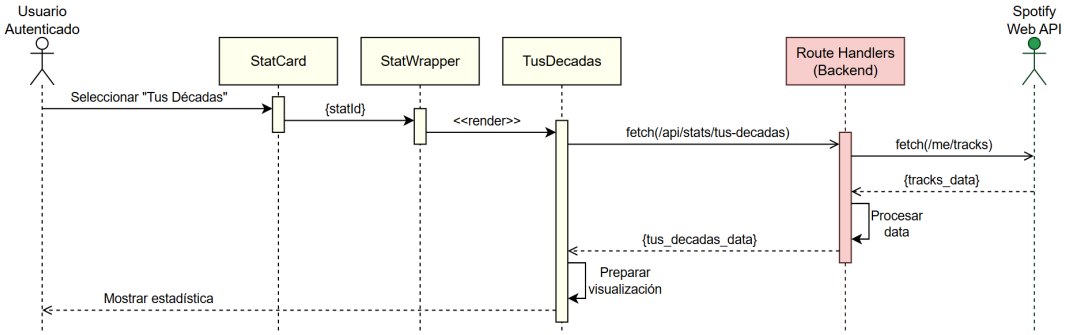

Figura 5.12: Diagrama de secuencia: Ver Tus Décadas.

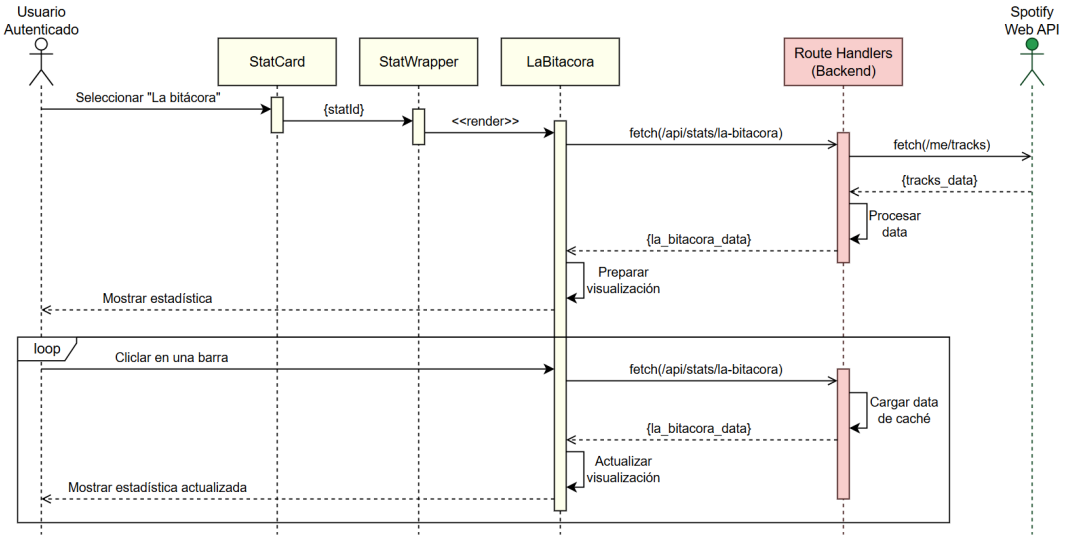

Figura 5.13: Diagrama de secuencia: Ver La Bitácora.

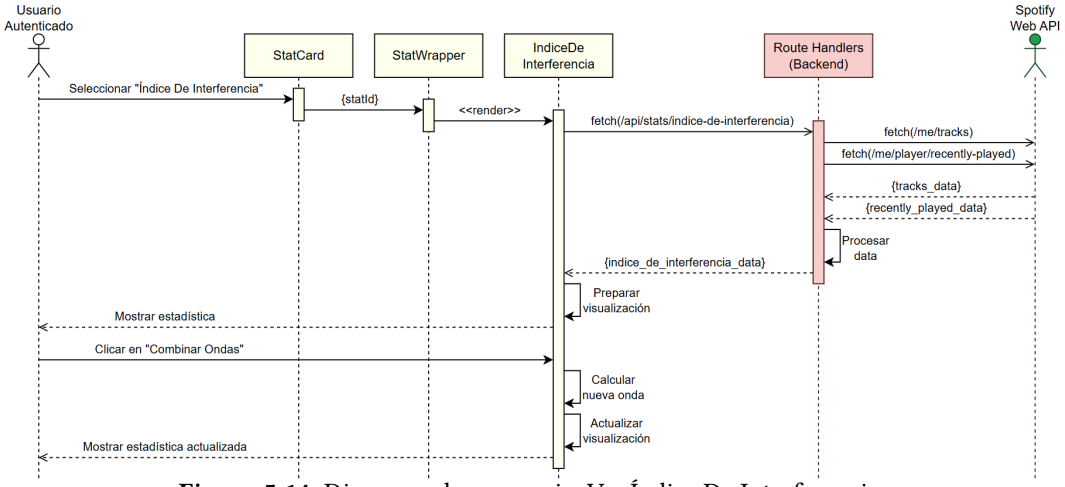

Figura 5.14: Diagrama de secuencia: Ver Índice De Interferencia.

5.5. Seguridad

Como esta aplicación maneja información privada de los usuarios, un aspecto crítico en el diseño y desarrollo de la misma es la seguridad. En esta sección se describen las medidas implementadas para proteger los datos de posibles ataques comunes y reducir las vulnerabilidades del sistema.

5.5.1. Gestión de Credenciales

La mala gestión de credenciales es un punto de vulnerabilidad común en aplicaciones web. En el caso del proyecto, se manejan varias credenciales sensibles, especialmente durante el flujo de inicio de sesión. A continuación se detallan las medidas aplicadas a cada una.

Client ID y Client Secret

En la creación de una aplicación que hace uso de la API de *Spotify*, se obtiene un **client_id** y un **client_secret** que son necesarios para la autenticación con la API. Estas credenciales pertenecen al desarrollador y deben ser protegidas adecuadamente para evitar su exposición a terceros no autorizados. Se almacenan exclusivamente en el servidor, en un archivo de variables de entorno (`.env.local`), que no se incluye en el control de versiones. En el momento del despliegue, estas variables se cargan en el entorno de producción, que han de ser anteriormente indicadas de manera manual a través de un panel de configuración de *Vercel*. Se puede acceder a estas variables en el lógica del servidor mediante el obtejo `process.env`.

Prevención de CSRF y Scopes

Para prevenir ataques de *Cross-Site Request Forgery* (CSRF), cada solicitud de inicio de sesión incluye un valor de estado (**state**) generado de forma aleatoria. Este valor es único para cada sesión y se valida al recibir la respuesta del servidor de autenticación. Este mecanismo, recomendado explícitamente en la documentación de *Spotify* [2], garantiza que las solicitudes sean legítimas y originadas únicamente desde el cliente autorizado, evitando que actores maliciosos puedan ejecutar acciones en nombre del usuario.

Por otro lado, durante el paso de la autorización en el proceso de inicio de sesión, se respeta el principio de mínimos privilegios, donde solo se solicitan los permisos necesarios y ninguno más. Los usuarios son informados claramente sobre los **scopes** solicitados y su propósito antes de otorgar los permisos. De esta manera se consigue reducir el riesgo de exposición innecesaria de datos.

Access Token y Refresh Token

Una vez que el usuario ha iniciado sesión y la aplicación ha obtenido un **access_token** y un **refresh_token**, estos se almacenan en unas cookies correspondiente. La configuración de estas cookies es muy importante para protegerlas frente a ataques comunes como *Cross-Site Scripting* (XSS) o *Man-In-The-Middle* (MITM). Las siguientes marcas (*flags*) permiten configurar la seguridad necesaria:

- **`httpOnly`:** Indicando esta opción como "`true`", asegura que las cookies no puedan ser accedidas por código JavaScript en el navegador, protegiéndolas contra ataques de XSS.
- **`secure`:** Indicando esta opción como "`true`", las cookies solo pueden ser transmitidas a través de conexiones HTTPS, protegiéndolas contra ataques de MITM. Esta opción solo es necesaria en un entorno de producción, es posible desactivarlo durante el desarrollo.
- **`maxAge`** Estableciendo un tiempo de vida limitado, se garantiza que se las cookies se eliminen automáticamente una vez cumplido el periodo concretado, minimizando el impacto de un posible compromiso.

5.5.2. Routing y Conexiones Seguras

Next.js proporciona un sistema de **middleware** que permite ejecutar procesos antes de que se manejen las solicitudes de las rutas. En este proyecto, se ha implementado un middleware que verifica la validez de las cookies de sesión antes de permitir el acceso a las rutas protegidas. Si no existe una cookie con un access token, el usuario es redirigido a la página de inicio de sesión, grantizando que solo los usuarios autenticados puedan acceder a las estadísticas y los datos personales.

Para la comunicación entre el cliente y la aplicación, al desplegatse en *Vercel*, éste proporciona automáticamente **conexiones HTTPS** para todas las solicitudes. Esto garantiza la confidencialidad de los datos transmitidos entre el cliente y el servidor, protegiendo contra ataques de intercepción como el MITM.

5.5.3. Otras Medidas

Del mismo modo que se han implementado mecanismos explícitos para prevenir ataques comunes, la seguridad de una aplicación no solo depende de configuraciones específicas, sino también del entorno de desarrollo y de las prácticas adoptadas a lo largo del ciclo de vida del software. La seguridad implícita, aquella que surge como resultado de buenas prácticas y herramientas adecuadas, desempeña un gran papel.

- **Análisis estático del código con ESLint:** Un linter como *ESLint* permite detectar errores, malas prácticas y patrones potencialmente inseguros en el código TypeScript antes de que lleguen a producción. Integrarlo en el flujo de desarrollo facilita la identificación temprana de vulnerabilidades y la posibilidad de corregirlos antes de que se conviertan en problemas de seguridad.
- **Dependencias actualizadas:** La aplicación se construye sobre versiones recientes y estables de *Next.js*, *React* y *Node.js LTS*, lo que permite beneficiarse de mejoras en seguridad y reducir la exposición a vulnerabilidades conocidas; un riesgo muy común derivado del uso de software obsoleto.
- **Ejecución de pruebas:** Realizar pruebas sistemáticas permite identificar fallos inesperados, comportamientos anómalos o configuraciones incorrectas en la aplicación.
- **Prácticas recomendadas:** Seguir las guías oficiales de seguridad, tanto de la API de *Spotify* como de *Next.js*, garantiza que se cumplen los estándares más recientes.

5.6. Diseño de Pruebas

Capítulo 6

Implementación

Capítulo 7

Pruebas

Capítulo 8

Despliegue

8.1. Vercel

8.2. CD/CI

Capítulo 9

Conclusiones

Cita falsa [3]

Apéndice

Apéndices

Bibliografía

[1] Spotify, “Spotify Design Guidelines,” 2025, Último acceso: 2025-01-31. [Online]. Available: https://developer.spotify.com/documentation/design Ver página 49.

[2] ——, “Authorization Code Flow,” 2025, Último acceso: 2025-01-30. [Online]. Available: https://developer.spotify.com/documentation/web-api/tutorials/code-flow Ver página 55.

[3] A. Falso, “Título ficticio para pruebas,” 2024, cita ficticia para evitar errores de compilación en LaTeX. Ver página 61.