Universitat Oberta de Catalunya

uoc.edu

Desarrollo de una aplicación híbrida para la creación de *playlists* de Spotify desde *setlists* de conciertos utilizando *Ionic* y *Angular*: ***LiveList***

Víctor Gómez Ramos
Plan de Estudios: Grado de Ingeniería Informática. Curso 2023/2024.
Área del trabajo final: Desarrollo multiplataforma de aplicaciones móviles.

Consultores: Carles Sànchez Rosa, Jordi Almirall López.
Profesor responsable de la asignatura: Carles Garrigues Olivella.

Fecha Entrega: 09/01/2024

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Desarrollo de una aplicación híbrida para la creación de playlists de Spotify desde setlists de conciertos utilizando Ionic y Angular:* ***LiveList***
Nombre del autor:	*Víctor Gómez Ramos*
Nombre del consultor/a:	*Carles Sànchez Rosa, Jordi Almirall López*
Nombre del PRA:	*Carles Garrigues Olivella*
Fecha de entrega (mm/aaaa):	01/2024
Titulación:	*Grado en Ingeniería Informática.* *Curso 2023/24*
Área del Trabajo Final:	*TFG-Desarrollo multiplataforma de aplicaciones móviles*
Idioma del trabajo:	*Castellano*
Palabras clave	*App, híbrida, multiplataforma*

Resumen del Trabajo (máximo 250 palabras): *Con la finalidad, contexto de aplicación, metodología, resultados i conclusiones del trabajo.*

Este proyecto final presenta el desarrollo de una aplicación móvil dirigida a los entusiastas de la música, enfocada en la gestión de listas de canciones y conciertos. El proyecto abarca las fases de diseño, desarrollo, pruebas y documentación, siguiendo un enfoque estructurado para garantizar la creación de una aplicación de alta calidad y fácil de usar.

La fase de diseño de la interfaz de usuario implica la creación de *wireframes* y mockups, asegurando una interfaz intuitiva y visualmente atractiva. El desarrollo de la aplicación utiliza la tecnología *Ionic*, permitiendo la creación de aplicaciones móviles híbridas para múltiples plataformas. La integración con las *APIs* de setlist.fm y Spotify es un aspecto clave del desarrollo, proporcionando funcionalidades para la consulta de *setlists* y la creación de listas de reproducción.

La fase de pruebas incluye pruebas exhaustivas para identificar y solucionar cualquier problema funcional, de rendimiento o de seguridad. Además, el proyecto incluye la creación de documentación técnica detallada que describe la funcionalidad de la aplicación, las *APIs* utilizadas y los procesos de desarrollo.

Las tecnologías clave utilizadas en el proyecto incluyen el *Framework Ionic*, *Capacitor* para el acceso a la funcionalidad nativa y *Google Cloud* para alojar *APIs* intermedias. La implementación de pruebas unitarias se lleva a cabo utilizando *Karma* y *Jasmine*, asegurando la confiabilidad e integridad del código base.

Este proyecto tiene como objetivo ofrecer una aplicación móvil sólida y rica en funciones para los entusiastas de la música, proporcionando un acceso sin problemas a la gestión de *setlists* y la creación de listas de reproducción, siguiendo las mejores prácticas en el desarrollo de software y el diseño de experiencia de usuario.

Abstract (in English, 250 words or less):

This final project presents the development of a mobile application aimed at music enthusiasts, focusing on the management of setlists and playlists. The project encompasses the design, development, testing, and documentation phases, following a structured approach to ensure the creation of a high-quality and user-friendly application.

The user interface design phase involves the creation of wireframes and mockups, ensuring an intuitive and visually appealing interface. The application development utilizes Ionic technology, allowing for the creation of hybrid mobile applications for multiple platforms. Integration with *setlist.fm* and *Spotify APIs* is a key aspect of the development, providing functionalities for setlist querying and playlist creation.

The testing phase includes comprehensive testing to identify and address any functional, performance, or security issues. Additionally, the project includes the creation of detailed technical documentation describing the application's functionality, the *APIs* used, and the development processes.

Key technologies utilized in the project include *Ionic Framework*, *Capacitor* for native functionality access, and *Google Cloud* for hosting intermediary *APIs* (*middleware*). The implementation of unit testing is carried out using *Karma* and *Jasmine*, ensuring the reliability and integrity of the codebase.

This project aims to deliver a robust and feature-rich mobile application for music enthusiasts, providing seamless access to setlist management and playlist creation while adhering to best practices in software development and user experience design.

Índice

Lista de figuras

1. Introducción

1.1. Contexto y justificación del Trabajo

La música es una parte esencial de la vida de muchas personas alrededor del mundo. Los conciertos en vivo son un modo emocionante de experimentar la música en su forma más pura y auténtica. Los amantes de la música suelen asistir a conciertos de sus artistas favoritos para disfrutar de actuaciones en vivo. Sin embargo, planificar y disfrutar de un concierto muchas veces implica algo más que comprar una entrada. Es posible que los asistentes quieran conocer la lista de canciones (*setlist*) del concierto antes o después del evento y también quieran crear una lista de reproducción personalizada para revivir la experiencia o compartirla con amigos.

En este contexto surge la necesidad de una aplicación que brinde una experiencia completa a los amantes de la música. Actualmente existen sitios web y aplicaciones que brindan información sobre listas de conciertos, pero a menudo están fragmentados y carecen de integración directa con los servicios de *streaming*. Los amantes de la música se ven obligados a consultar múltiples fuentes de información sobre conciertos y luego crear manualmente listas de reproducción (*playlists*) en sus plataformas de música favoritas. Esta falta de integración y conveniencia crea una oportunidad para desarrollar una aplicación que resuelva este problema.

El motivo de este trabajo radica en la creación de una aplicación móvil que satisfaga las necesidades de los aficionados a la música en general, y en particular a aquellos que disfrutan de sus artistas preferidos en directo. Los siguientes puntos resaltan por qué este proyecto es relevante y necesario:

- Mejora de la experiencia del usuario: Al brindarles a los usuarios la posibilidad de ver listas de conciertos y crear listas de reproducción fácil y rápidamente dentro de una aplicación, la experiencia del usuario mejora significativamente. Esto elimina la necesidad de cambiar entre múltiples aplicaciones y sitios web.

- Integración con servicios populares: Por ejemplo, la integración con la API de *Spotify*, una de las principales plataformas de transmisión de música, permitirá a los usuarios acceder a su música favorita y crear listas de reproducción de conciertos directamente desde la aplicación.

- Exploración de tecnologías actuales: Este proyecto daría la oportunidad de explorar y aplicar tecnologías actuales como *Ionic* para desarrollar aplicaciones móviles híbridas y API de terceros para recopilar y manipular datos.

Por todo lo descrito anteriormente, la principal aportación de este trabajo será el desarrollo de una aplicación móvil híbrida utilizando *Ionic*, que permitirá a los usuarios ver listas de conciertos desde la API setlist.fm y crear listas de reproducción directamente a través de los servicios de *streaming* de música antes mencionados (Spotify) a través de su API.

Además, si los usuarios de la aplicación están registrados en setlist.fm, también podrán realizar un seguimiento de los conciertos a los que han asistido. Esto otorgará una experiencia completa para planificar actividades musicales y mantenerse conectado con nuestra música favorita.

1.2. Objetivos del Trabajo

La aplicación resultante debe garantizar la usabilidad y la experiencia del usuario en las plataformas móviles Android y iOS. Además, debe cumplir con estándares de seguridad y privacidad en la manipulación de datos de usuarios y autenticación.
En cuanto a los requerimientos, tendremos los siguientes:

- Requerimientos Funcionales:

    1. Consulta de *Setlists*: La aplicación permitirá a los usuarios buscar y consultar *setlists* de conciertos de artistas y eventos específicos utilizando la API de setlist.fm.
    2. Consulta de conciertos a los que se ha asistido: Se podrá obtener una lista de conciertos a los que un usuario registrado en Setlist.fm haya asistido.
    3. Integración con Spotify: Los usuarios podrán autenticarse en sus cuentas de Spotify y crear listas de reproducción directamente desde la aplicación, agregando las canciones de los *setlists* consultados.

- Requerimientos No Funcionales:

    1. Compatibilidad Multiplataforma: La aplicación será compatible con múltiples plataformas móviles, incluyendo Android y iOS, para llegar a la mayor cantidad de usuarios posible.
    2. Seguridad: Se implementarán medidas de seguridad adecuadas para proteger la información de los usuarios, especialmente en lo que respecta a la autenticación de cuentas de Spotify.
    3. Rendimiento: La aplicación deberá ser eficiente en términos de rendimiento, garantizando una respuesta rápida a las solicitudes de los usuarios y una carga ágil de los datos.

4. Documentación Técnica: Se proporcionará documentación técnica completa que describa el funcionamiento de la aplicación, las API utilizadas y las consideraciones de desarrollo.
5. Experiencia del Usuario: La aplicación se centrará en ofrecer una experiencia de usuario positiva, con una interfaz atractiva, navegación intuitiva y tiempos de respuesta rápidos.

1.3. Enfoque y método seguido

El enfoque seleccionado para este proyecto es el desarrollo de un producto nuevo, una aplicación móvil híbrida que combina la funcionalidad de consulta de *setlists* de conciertos y la creación de *playlists* en Spotify en una sola aplicación. A continuación, se explican las razones detrás de esta elección y cómo se llevará a cabo:

- Desarrollo de un Producto Nuevo:

La elección de desarrollar un producto nuevo se justifica por varias razones:

1. Falta de Solución Integral: Se ha detectado la falta de soluciones integrales para consultar setlists y crear playlists de manera conjunta. Esto representa una oportunidad para crear una aplicación única y completa que satisfaga las necesidades de los usuarios.

2. Innovación y Diferenciación: El desarrollo de un producto nuevo permite la innovación y la posibilidad de diferenciarse de las posibles aplicaciones existentes. Esto atraerá a los usuarios que buscan una experiencia más completa.

- Método de Desarrollo:

La metodología que se seguirá será una metodología en cascada (también llamada *Waterfall*). Como su nombre indica, el método en cascada es un método de gestión de proyectos basado en el desarrollo secuencial de etapas que ocurren como una cascada (de ahí su nombre). Es decir, divide el proyecto en diferentes fases secuenciales, donde cada nueva fase comienza sólo cuando se ha completado la fase anterior.

El método de desarrollo elegido para este proyecto se basará en las siguientes etapas:

1. Investigación y Planificación: Se realizará una investigación exhaustiva de las API de setlist.fm y Spotify para comprender sus

capacidades y limitaciones. También se planificarán las funcionalidades, la arquitectura y la tecnología a utilizar.

2. Diseño de la Interfaz de Usuario: Se diseñará la interfaz de usuario de la aplicación, asegurándose de que sea intuitiva y atractiva para los usuarios. Esto incluirá la creación de *wireframes* y *mockups*.

3. Desarrollo de la Aplicación: Se llevará a cabo el desarrollo de la aplicación utilizando la tecnología *Ionic*, que permite el desarrollo de aplicaciones móviles híbridas para múltiples plataformas. Se implementarán las funcionalidades de consulta de *setlists* y creación de *playlists*, así como la integración con las API de setlist.fm y Spotify.

4. Pruebas y Depuración: Se realizarán pruebas exhaustivas para identificar y corregir errores o problemas de funcionamiento. Esto incluirá pruebas de funcionamiento, pruebas de rendimiento y pruebas de seguridad.

5. Documentación: Se creará documentación técnica completa que describa el funcionamiento de la aplicación, las API utilizadas y los procesos de desarrollo.

1.4. Planificación del Trabajo

El trabajo está pensado para que se realicen durante el semestre las siguientes entregas, con sus respectivas tareas:

- **PEC1** (Entrega: 10 de octubre de 2023):

Investigación inicial sobre las APIs de setlist.fm y Spotify.
Definición detallada de los requerimientos funcionales y no funcionales.
Establecimiento de la arquitectura inicial de la aplicación.
Investigación de las mejores prácticas en diseño de interfaz de usuario (UI/UX) para aplicaciones móviles.

- **PEC2** (Entrega: 7 de noviembre de 2023):

Diseño de la interfaz de usuario de la aplicación.
Diseño de la arquitectura de la aplicación, incluyendo la estructura de carpetas y componentes.
Creación de *wireframes* y *mockups* de la aplicación.
Elección de colores, fuentes y elementos de diseño.
Desarrollo de una versión preliminar del *frontend* de la aplicación (puede ser en blanco y negro).

- **PEC3** (Entrega: 19 de diciembre de 2023):

Desarrollo de la lógica de la aplicación.
Implementación de la integración con la API de setlist.fm para la consulta de *setlists*.
Implementación de la integración con la API de Spotify para la creación de *playlists*.
Implementación de la autenticación de usuarios en Spotify.
Pruebas preliminares de la funcionalidad de consulta de *setlists*.
Pruebas de funcionamiento básicas del *frontend* y la navegación.
Preparación de la documentación técnica inicial.

- **Entrega Final** (Entrega: 9 de enero de 2024):

Finalización del desarrollo de la aplicación.
Pruebas exhaustivas, incluyendo pruebas de rendimiento y seguridad.
Corrección de errores y mejoras de usabilidad basadas en pruebas de usuario.
Preparación y revisión de la memoria del proyecto.
Creación de la presentación para la defensa del TFG.
Empaquetamiento y preparación del producto final para su entrega.

Para calcular el coste en horas de cada tarea se han tenido en cuenta 4 horas disponibles de media en días no festivos. Se dispone de 69 días no festivos hasta la entrega, por lo que el total de horas disponibles es de 276.
De este total de horas se han distribuido entre las tareas 250, dejando las 26 horas restantes para posibles imprevistos.
El cómputo de las horas asignadas a cada tarea queda del siguiente modo:

Actividad	Duración Estimada (En horas)
PEC1: Investigación Inicial	15
PEC1: Definición de Requerimientos	10
PEC1: Establecimiento de Arquitectura	10
PEC1: Investigación de Diseño UI/UX	5
PEC2: Diseño de Interfaz de Usuario	15
PEC2: Diseño de Arquitectura	15
PEC2: Creación de *Wireframes* y *Mockups*	15
PEC2: Diseño de Colores y Fuentes	5
PEC2: Desarrollo Preliminar *Frontend*	15
PEC3: Desarrollo de Lógica	25
PEC3: Integración con setlist.fm	15
PEC3: Integración con Spotify	15
PEC3: Pruebas Preliminares	15
PEC3: Pruebas de Funcionamiento	15
PEC3: Documentación Técnica	10

Entrega Final: Finalización	10
Entrega Final: Implementación Adicional	10
Entrega Final: Pruebas Exhaustivas	10
Entrega Final: Corrección de Errores	5
Entrega Final: Preparación de Memoria	5
Entrega Final: Creación de Presentación	5
Entrega Final: Empaquetamiento y Entrega	5

Se ha utilizado la herramienta *GanttProject* para realizar un diagrama de Gantt de la planificación anterior. Esta herramienta no permite el cómputo de las tareas por horas, solamente lo hace por días, por lo que se ha tenido que ajustar para jornadas de 4 horas. Finalmente se ha obtenido el siguiente diagrama:

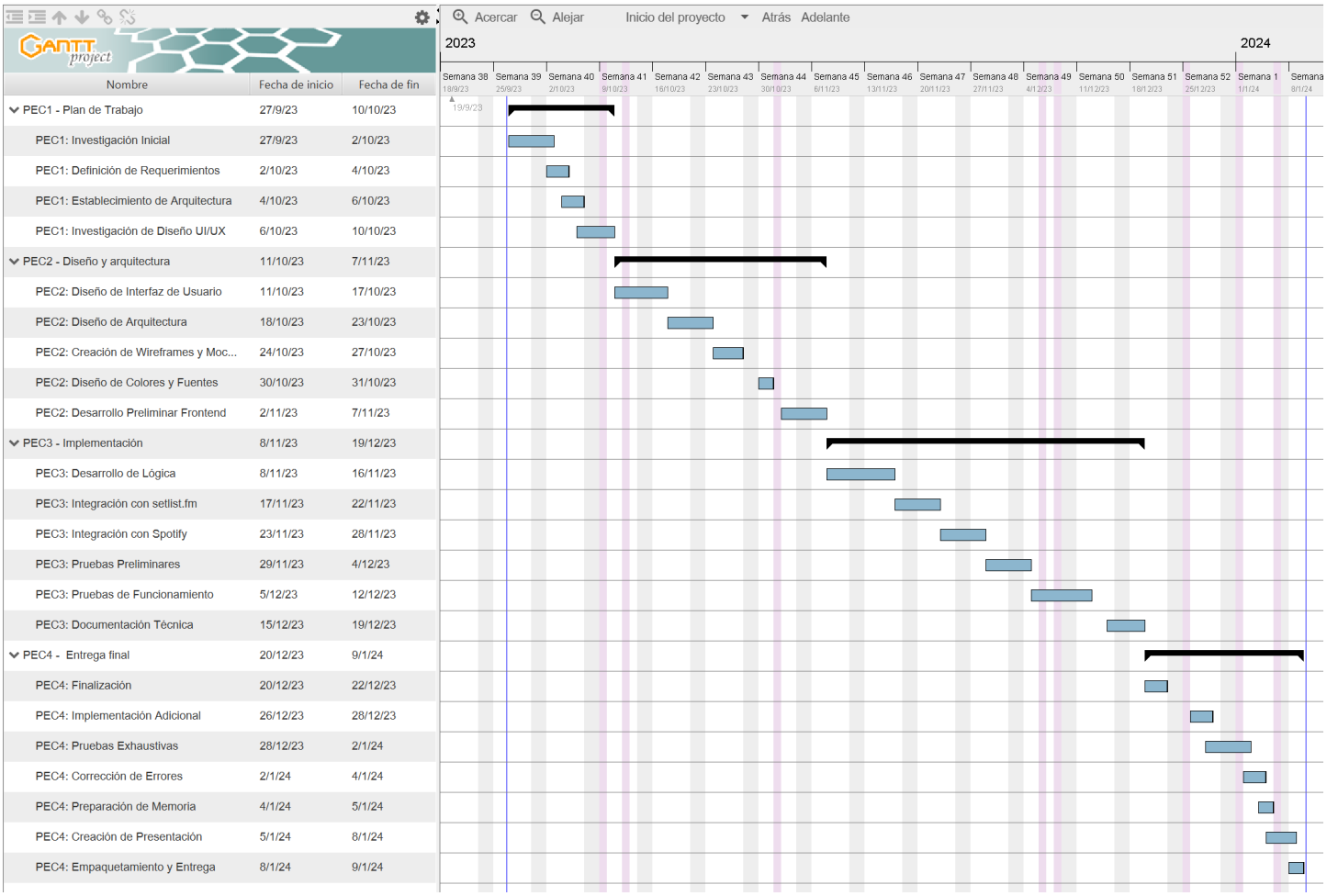


Ilustración 1: Planificación

Para terminar este apartado de planificación, y con el cómputo de horas realizado, se procederá a realizar una valoración del coste económico.

Para la valoración de los costes económicos se tendrá en cuenta que se mantendrá a un solo trabajador, durante el tiempo estimado (unas 250h), trabajando en el proyecto.

Adicionalmente, también se tendrán en cuenta otros gastos, como el material necesario para la realización del proyecto, en este caso se tratará del equipo informático.

Por tanto, el desglose quedará de la siguiente manera:

Descripción	Precio
Salario del desarrollador (56€/hora)	50€ x 250 horas = 12500€
Precio del material (equipo informático)	1500€
TOTAL	**14000€**

1.5. Breve sumario de productos obtenidos

Los productos obtenidos al finalizar el proyecto incluyen:

- **Aplicación Móvil Híbrida Funcional**: El producto principal será una aplicación móvil híbrida que permitirá a los usuarios consultar *setlists* de conciertos y crear *playlists* en Spotify desde una interfaz intuitiva y atractiva.

- **Memoria descriptiva**: Se adjuntará esta memoria descriptiva, en la que quedarán recogidos los resultados, así como también todo el proceso de desarrollo de este.

- **Presentación del proyecto en formato de vídeo**: Se entregará una presentación en formato vídeo donde se dará explicación al proceso realizado para la obtención del producto final.

- **Documentación Técnica Detallada**: Se proporcionará documentación técnica completa que describirá el funcionamiento de la aplicación, la arquitectura utilizada, las API integradas y otros detalles técnicos importantes.

- **Informe de Pruebas**: Se generará un informe de pruebas detallado que documentará las pruebas realizadas en la aplicación.

- **Documentación de Usuario**: Se creará documentación de usuario que guiará a los usuarios finales sobre cómo utilizar la aplicación de manera efectiva, incluyendo instrucciones detalladas y capturas de pantalla.

1.6. Breve descripción de los otros capítulos de la memoria

El resto de los capítulos serán los siguientes:

2 Análisis, diseño y arquitectura.
En este apartado se analizarán los contextos de uso, el diseño conceptual, el prototipado de la aplicación. En resumen, todo el proceso necesario antes de pasar a la implementación.

3 Implementación.

En este capítulo, se detallará el proceso de desarrollo de la aplicación. Se describen las tecnologías y herramientas utilizadas, así como los desafíos y soluciones encontrados durante la implementación. Podrían incluirse fragmentos de código relevante y ejemplos para ilustrar el proceso de desarrollo.

4 Conclusiones

En este capítulo, se presentarán las conclusiones generales del trabajo. Se resumen los resultados alcanzados, se evalúa si se cumplieron los objetivos planteados al inicio del proyecto y se discuten las lecciones aprendidas durante el proceso.

5 Glosario

Aquí se proporcionará un glosario de términos técnicos y conceptos utilizados a lo largo del trabajo. Esto facilita la comprensión de la terminología específica relacionada con el proyecto y permite a los lectores consultar definiciones cuando sea necesario.

6 Bibliografía

Este capítulo enumera todas las fuentes de información utilizadas en el trabajo, como libros, artículos, sitios web y documentos relevantes. Se presentan de acuerdo con un formato de cita bibliográfica estándar.

7 Anexos

Por último, en este capítulo se incluyen elementos adicionales que complementan el trabajo principal.

2. Análisis, diseño y arquitectura.

2.1. Diseño centrado en el usuario.

2.1.1. Usuarios.

2.1.1.1. Métodos de indagación.

Los métodos de indagación ayudan a estudiar las necesidades de los usuarios y se llevan a cabo en las etapas de definición del contexto de uso y de los propios requisitos, involucrando a los usuarios en las distintas actividades a realizar. Estos métodos tienen como objetivo obtener la información necesaria para definir y desarrollar un producto con una alta usabilidad, en este caso una aplicación.

Para llegar a conocer las características de los usuarios de la aplicación *LiveList* y sus necesidades y objetivos, se elegirán dos métodos, el método de *entrevistas en profundidad* y el método de *encuestas*.

- *Entrevistas en profundidad*:

Permiten obtener información de tipo cualitativo.
La elección de utilizar entrevistas en profundidad se justifica por su capacidad para proporcionar información detallada y rica sobre las necesidades y expectativas de los usuarios.
Para la realización de las entrevistas, se ha utilizado una estructura de preguntas abiertas para fomentar la discusión. Se alentó al entrevistado a compartir experiencias personales y opiniones, y se hicieron preguntas de seguimiento para profundizar en respuestas específicas.
Los entrevistados serán 5 hombres y 5 mujeres de diferentes perfiles y edades.
La identidad de los entrevistados será anónima.

El guion seguido para las entrevistas ha sido el siguiente:

- Preguntas Introductorias:

¿Con qué frecuencia asistes a actuaciones en directo?
¿Con qué frecuencia utilizas aplicaciones de música para escuchar música?
¿Tienes alguna experiencia previa con aplicaciones que ofrecen características similares a la que estamos desarrollando?

- Experiencia y Necesidades de Usuario:

¿Qué te motiva a asistir a conciertos en vivo?
¿Qué esperas de esas experiencias?

¿Cómo te gustaría utilizar una aplicación que convierte *setlists* de conciertos en listas de reproducción de música?
¿Qué características o funcionalidades serían más valiosas para ti en esta aplicación?

- Integración con Plataformas de Música:

¿En qué plataforma de música prefieres escuchar música (por ejemplo, *Spotify*, *Apple Music*, *Deezer*, etc.)?
¿Te gustaría que nuestra aplicación se integre con tu plataforma de música favorita? ¿Por qué?

- Usabilidad y Experiencia de Usuario:

¿Qué características de diseño o funcionalidades consideras esenciales para que una aplicación sea fácil de usar en este contexto?

- Análisis de Competidores:

¿Has utilizado aplicaciones similares en el pasado? ¿Cuál fue tu experiencia con ellas?

Durante las entrevistas, se ha podido explorar a fondo las experiencias del usuario, sus desafíos y sus preferencias. Esto nos permitirá diseñar la aplicación de manera más precisa y alineada con las necesidades reales de los usuarios.

Después de realizar las entrevistas, se obtienen los resultados siguientes:

- Los entrevistados destacaron la necesidad de recordar y recrear la experiencia de conciertos a través de listas de reproducción.
- La mayoría de entrevistados expresó un fuerte interés en la integración con Spotify, esto demuestra que la integración con Spotify es una característica importante para los usuarios.
- La recreación de la experiencia del concierto es una necesidad clave.
- La mayoría de los usuarios coinciden en que las aplicaciones que más les gustan son las aplicaciones simples, sin muchas opciones en pantalla.
- La compartición de listas de reproducción es una característica social valorada por los usuarios.

• *Encuestas*:

Las encuestas permiten obtener información de tipo cuantitativo acerca de los usuarios.
La elección de utilizar una encuesta se justifica por su eficiencia en la recopilación de datos cuantitativos de manera rápida y con un alcance amplio.

A través de preguntas estructuradas, hemos obtenido información sobre las necesidades y preferencias del usuario, lo que nos permitirá tomar decisiones basadas en datos para el desarrollo de la aplicación.

La encuesta realizada y los datos resultantes de la misma se pueden consultar en el Anexo 1.

De toda la información recopilada mediante las dos técnicas de indagación utilizadas, se puede concluir que:

- La aplicación más utilizada para escuchar música es Spotify.

- Los usuarios quieren buscar información de conciertos a los que han asistido como espectadores.

- Los usuarios quieren poder crear *playlists* en sus aplicaciones de música en *streaming* de los conciertos a los que han asistido.

- Existe también un número menor de usuarios a los que les gustaría poder crear *playlists* de manera personalizada.

- Por los comentarios libres, también se concluye que los usuarios prefieren un interfaz fácil de usar e intuitivo. Por lo que se intentará que el diseño sea lo más simple posible para facilitar el uso de la aplicación.

2.1.1.2. Fichas de perfil de usuario.

Con los datos obtenidos mediante las técnicas de indagación anteriores, y teniendo en cuenta que la aplicación que se desarrollará tendrá unas funcionalidades muy concretas, se han detectado los siguientes 4 perfiles de usuario:

Perfil de Usuario 1: Registrado en *Spotify* y en *Setlist.fm*

- Nombre del Perfil: Laura
- Lista de Comportamientos:
  Utiliza activamente *Spotify* para escuchar música.
  Asiste a conciertos en vivo con regularidad.
  Registra *setlists* de conciertos en *Setlist.fm*.
  Sigue a sus bandas y artistas favoritos en ambas plataformas.
  Comparte listas de reproducción con amigos y en redes sociales.
- Características Demográficas:
  Edad: 30 años
  Género: Femenino
  Ubicación: Ciudad urbana
  Profesión: Diseñadora gráfica

Perfil de Usuario 2: Registrado en *Spotify*, pero no en *Setlist.fm*

- Nombre del Perfil: Carlos
- Lista de Comportamientos:
  Utiliza *Spotify* como su plataforma principal para escuchar música.
  Asiste ocasionalmente a conciertos en vivo en eventos importantes.
  Disfruta de la creación de listas de reproducción temáticas.
  Comparte música con amigos a través de *Spotify*.
- Características Demográficas:
  Edad: 35 años
  Género: Masculino
  Ubicación: Suburbio
  Profesión: Médico

Perfil de Usuario 3: Registrado en *Setlist.fm*, pero no en *Spotify*

- Nombre del Perfil: Diego
- Lista de Comportamientos:
  Asiste regularmente a conciertos en vivo y registra *setlists* en *Setlist.fm*.
  Explora bandas emergentes y música independiente.
  Crea listas de reproducción temáticas personalizadas basadas en *setlists*.
  Comparte activamente música y experiencias musicales con amigos y en redes sociales.
- Características Demográficas:
  Edad: 25 años
  Género: Masculino
  Ubicación: Ciudad cosmopolita
  Profesión: Estudiante de música

Perfil de Usuario 4: No registrado en *Spotify* ni en *Setlist.fm*

- Nombre del Perfil: María
- Lista de Comportamientos:
  Disfruta de la música, pero no utiliza plataformas de *streaming* ni registra *setlists*.
  Asiste ocasionalmente a eventos musicales en su área.
  Prefiere aplicaciones de música de uso sencillo y sin necesidad de registro.
  No comparte música de manera activa en línea.
- Características Demográficas:
  Edad: 40 años
  Género: Femenino
  Ubicación: Pueblo rural
  Profesión: Profesora

2.1.2. Diseño conceptual.

2.1.2.1. *Point of view statements*.

Con la información recopilada en la fase anterior se han elaborado los siguientes *Point of view statements* para cada uno de los perfiles de usuario definidos:

Perfil de Usuario 1: Registrado en *Spotify* y en *Setlist.fm*

Point of View Statement 1:

> "Las personas registradas tanto en *Spotify* como en *Setlist.fm*, necesitan una forma de integrar de manera perfecta sus cuentas, permitiéndoles crear listas de reproducción personalizadas a partir de *setlists* de conciertos porque quieren disfrutar de la música en vivo en su plataforma de música favorita con facilidad."

Point of View Statement 2:

> "Las personas registradas tanto en *Spotify* como en *Setlist.fm*, necesitan una forma de llevar el control de los conciertos a los que han asistido en directo porque quieren tener un listado con sus conciertos."

Perfil de Usuario 2: Registrado en *Spotify*, pero no en *Setlist.fm*

Point of View Statement 1:

> "Las personas registradas en *Spotify*, pero no en *Setlist.fm*, necesitan una forma de crear listas de reproducción a partir de *setlists* de conciertos porque quieren poder descubrir nueva música y disfrutar de listas de reproducción personalizadas de manera eficiente."

Point of View Statement 2:

> "Las personas registradas en *Spotify*, pero no en *Setlist.fm*, necesitan una forma de explorar canciones y artistas relacionados con los conciertos a los que han asistido
> porque quieren consultar que canciones de sus artistas favoritos han visto en directo."

Perfil de Usuario 3: Registrado en *Setlist.fm*, pero no en *Spotify*

Point of View Statement 1:

> "Las personas registradas en *Setlist.fm*, pero no en *Spotify*, necesitan una forma de llevar el control de los conciertos a los que han asistido en directo porque quieren tener un listado con sus conciertos."

Point of View Statement 2:

> "Las personas registradas en *Setlist.fm*, pero no en *Spotify*, necesitan una forma de consultar las canciones que han visto en directo porque quieren saber a qué álbum pertenecen las canciones que han escuchado en directo."

Perfil de Usuario 4: No registrado en *Spotify* ni en *Setlist.fm*

Point of View Statement 1:

"Las personas no registradas en *Spotify* ni en *Setlist.fm*, necesitan una forma de consultar los conciertos de sus bandas favoritas porque quieren saber en qué recintos han actuado. "

Point of View Statement 2:

" Las personas no registradas en *Spotify* ni en *Setlist.fm*, necesitan una forma de consultar las canciones que sus bandas favoritas han tocado en directo porque quieren saber cuáles de estas canciones son las que más veces suelen tocar. "

2.1.2.2. Flujos de interacción.

A continuación, se muestran 4 flujos de interacción. Un flujo por cada uno de los perfiles de usuario vistos anteriormente:

Perfil de Usuario 1: Registrado en *Spotify* y en *Setlist.fm*

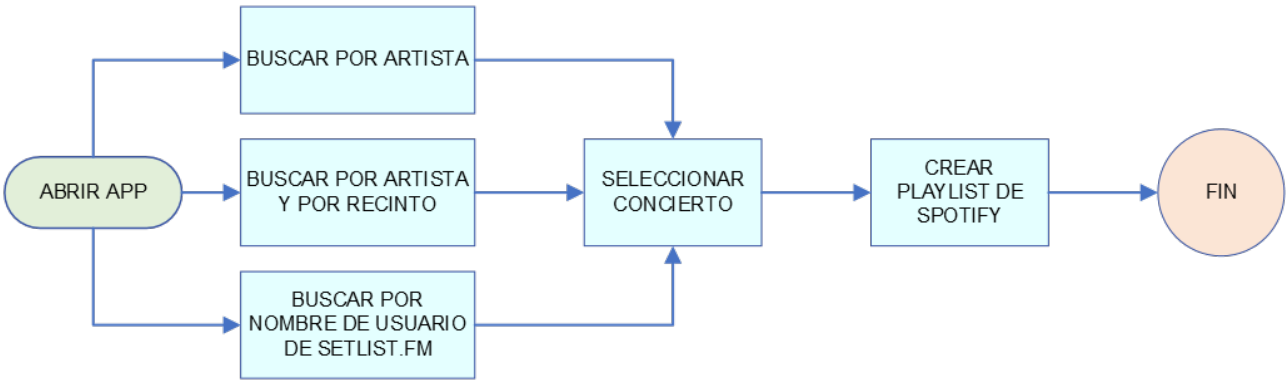


Ilustración 2: Perfil de usuario 1

Perfil de Usuario 2: Registrado en *Spotify*, pero no en *Setlist.fm*

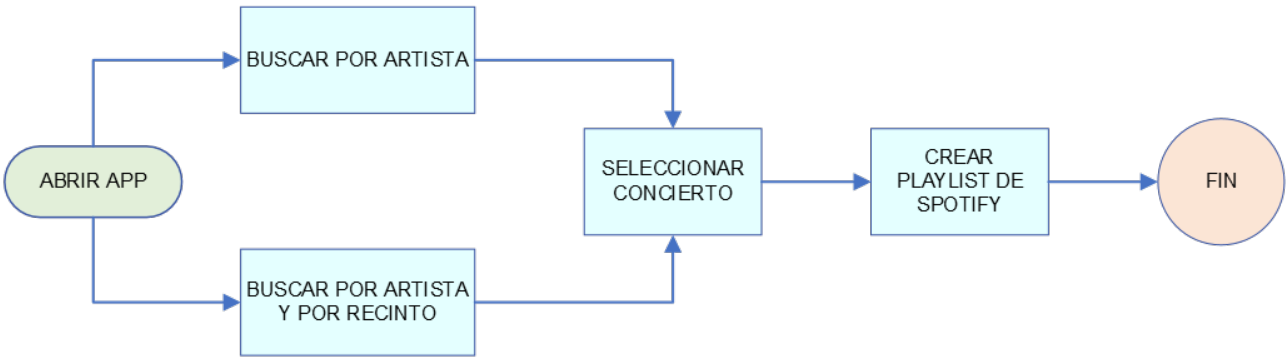


Ilustración 3: Perfil de usuario 2

Perfil de Usuario 3: Registrado en *Setlist.fm*, pero no en *Spotify*

Ilustración 4: Perfil de usuario 3

Perfil de Usuario 4: No registrado en *Spotify* ni en *Setlist.fm*

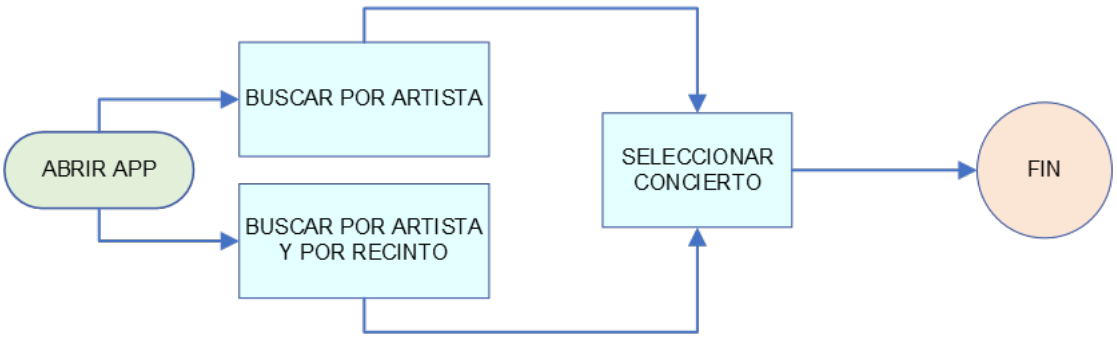


Ilustración 5: Perfil de usuario 4

2.1.3. Prototipado.

2.1.3.1. *Sketches*.

En la primera imagen se puede ver lo que será la pantalla principal, a la que se accederá una vez ejecutada la aplicación.

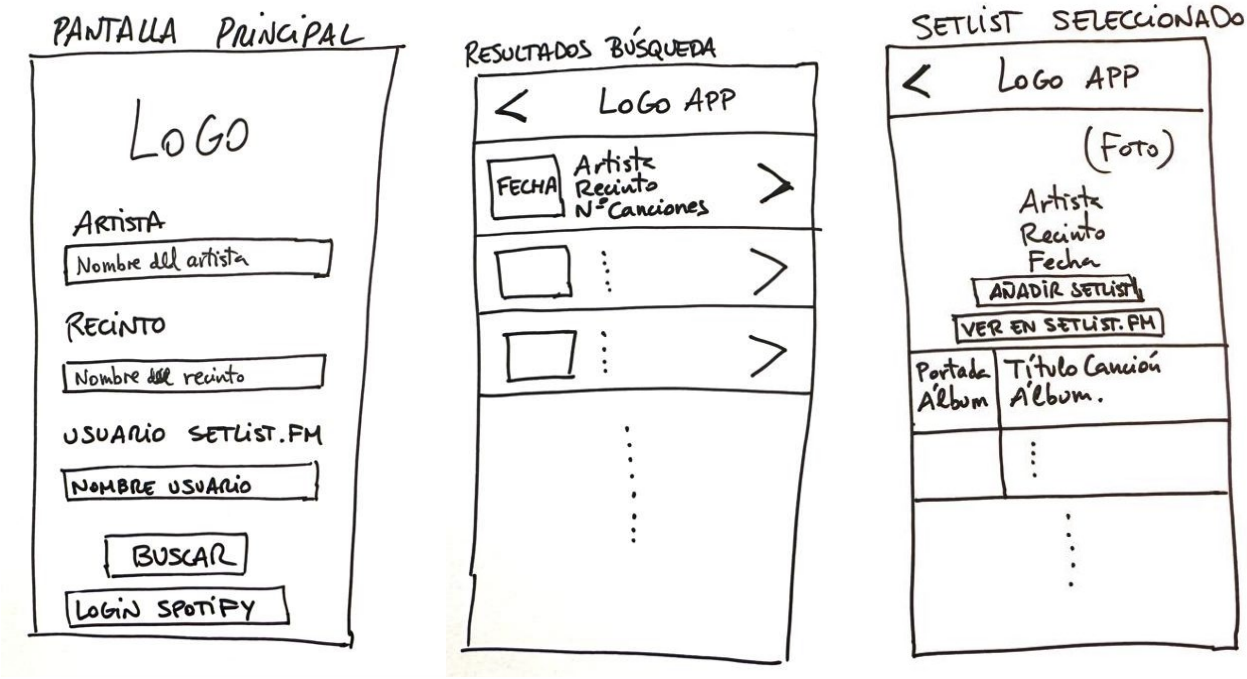


Ilustración 6: Sketches de la aplicación.

En la siguiente imagen se ve cómo será la pantalla de resultados de la búsqueda, y en la tercera captura se aprecia la pantalla del *setlist* seleccionado por cualquier método de búsqueda.

2.1.3.2. Prototipo de alta fidelidad.

A continuación, se pueden ver las tres pantallas del prototipo de alta fidelidad, y haciendo *click* en el siguiente enlace se puede navegar por el mismo.

Prototipo LiveList

Para poder abrir el prototipo hay que utilizar la contraseña **livelist**.

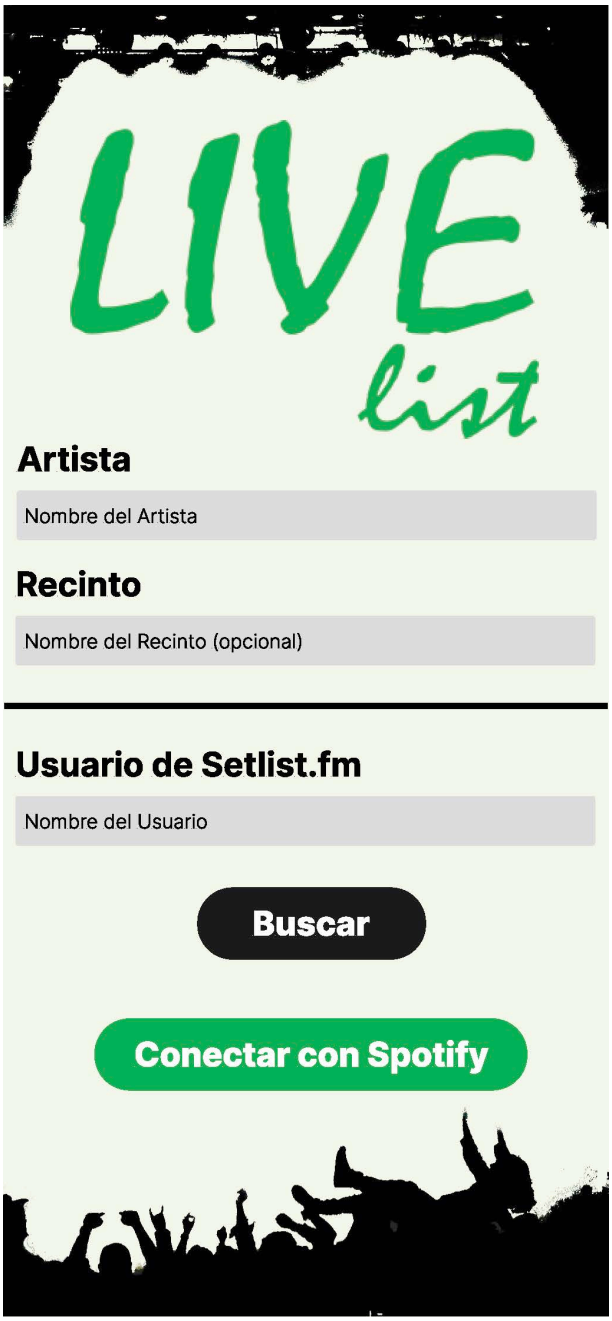


Ilustración 7: Prototipo 1

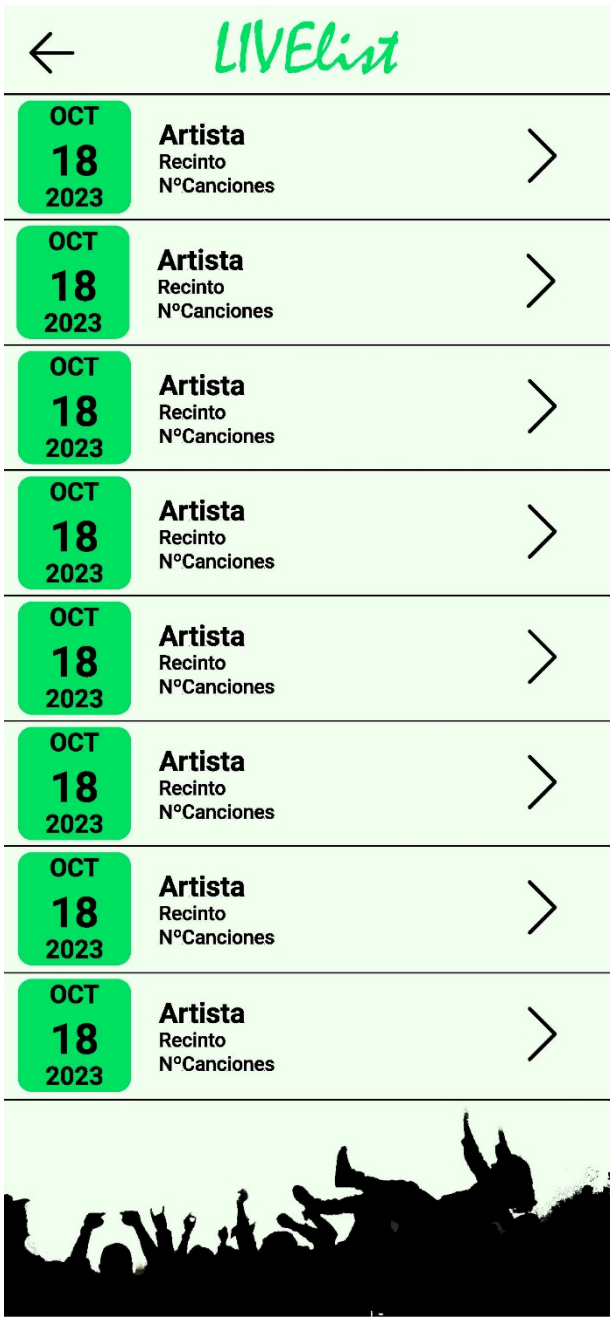


Ilustración 8: Prototipo 2

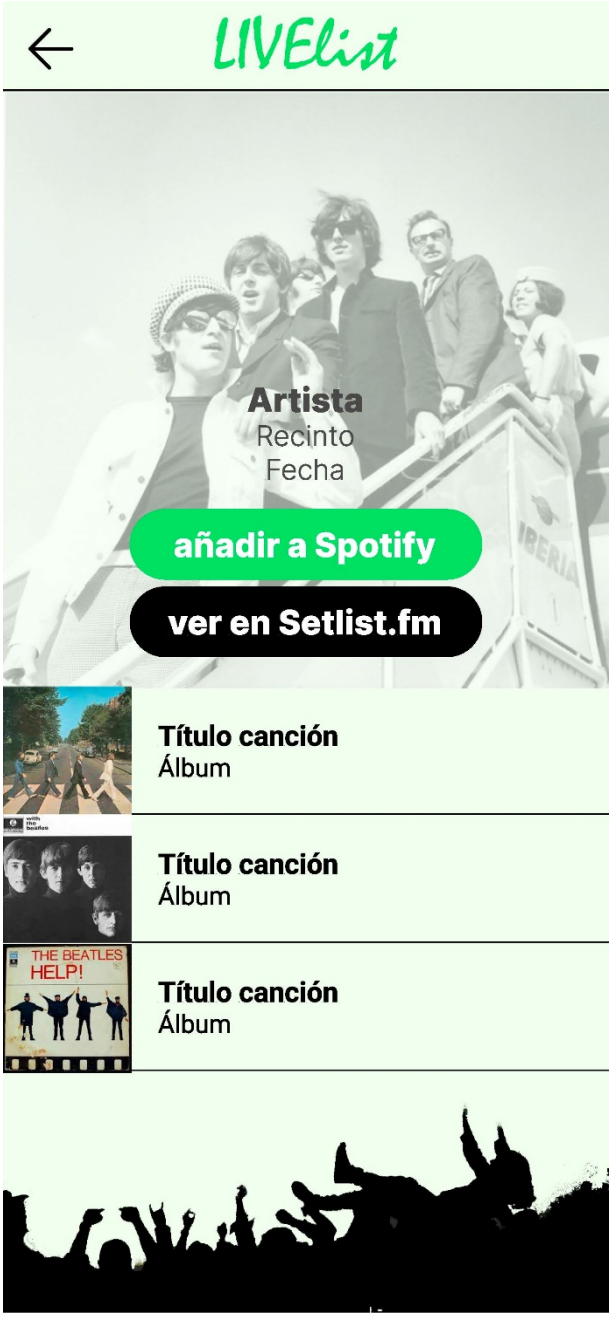


Ilustración 9: Prototipo 3

2.1.4. Evaluación.

2.1.4.1. *Tests* con usuarios.

Para evaluar el prototipo de la aplicación, se llevará a cabo una prueba con usuarios. A continuación, se proporciona la estructura del proceso de evaluación, incluyendo las preguntas de información sobre los usuarios, las tareas que deben realizar y las preguntas relacionadas con esas tareas:

- Preguntas de Información sobre el Usuario:
  - Nombre del usuario:
  - Edad:
  - ¿Con qué frecuencia asistes a conciertos en vivo?
  - ¿Con qué frecuencia utilizas aplicaciones de música en *streaming*?
  - ¿Estás registrado en plataformas como Spotify y Setlist.fm? (Especificar cuáles)
  - ¿Has utilizado aplicaciones similares a la que estamos evaluando en el pasado? (Especificar)

- Tareas para el *Test* con Usuarios:
  - Tarea 1: Inicio y exploración de eventos:
    - Inicia la aplicación y busca un evento/concierto de tu elección.
    - Visualiza los detalles del evento, incluyendo el *setlist* si está disponible.
  - Tarea 2: Integración con Plataforma de Música (*Spotify*):
    - Integra la aplicación con tu cuenta de la plataforma de música.
  - Tarea 3: Creación de una Lista de Reproducción:
    - Busca un evento y crear una lista de reproducción del evento en *Spotify* a partir del *setlist*.
  - Tarea 4: Consulta de eventos a los que se ha asistido (para usuarios de Setlist.fm):
    - Haz una búsqueda por tu nombre de usuario de *Setlist.fm*.
    - Navega por tus eventos y exporta alguno de ellos a *Spotify* (si dispones de una cuenta en *Spotify*).

- Preguntas relacionadas con las Tareas:

  - ¿Pudiste completar las tareas de manera exitosa? ¿Encontraste dificultades?
  - ¿Cómo calificarías la facilidad de uso de la aplicación en general?
  - ¿Hubo alguna característica que no encontraste en la aplicación y esperabas encontrar?
  - ¿La integración con la plataforma de música *Spotify* fue sencilla?

2.1.4.2. *Feedbak* obtenido.

Una vez realizadas las pruebas a dos usuarios, se obtienen los siguientes resultados:

	Usuario 1: Aficionado a la música en vivo, registrado en Spotify y Setlist.fm.	
	Inputs positivos	Inputs negativos
Concepto general	La idea de la aplicación es genial.	La interfaz de usuario inicial es demasiado simple.
Onboarding	El proceso de inicio de la aplicación fue rápido y sencillo.	La información sobre la integración con Spotify fue limitada.
Funcionalidad de búsqueda de un evento por artista y/o recinto.	No hubo problemas al consultar la lista de eventos por artista y lugar.	Sería útil una opción de búsqueda rápida de eventos cercanos.
Funcionalidad de búsqueda de un evento por usuario de *Setlist.fm*.	No hubo problemas al consultar la lista de eventos asistidos.	Sería útil poder buscar conciertos a los que se ha asistido en un periodo determinado.
Funcionalidad de conexión con *Spotify*.	La integración con Spotify funcionó sin problemas.	Al conectar con el usuario de *Spotify*, no quedó claro si se conectó con éxito.
Funcionalidad creación de *playlist* en *Spotify*.	No hubo problemas al crear el *playlist* en Spotify.	Al crear un *playlist* en *Spotify*, no quedó claro si se creó con éxito.

	Usuario 2: Melómano ocasional, registrado en Spotify.	
	Inputs positivos	Inputs negativos
Concepto general	La aplicación ofrece una forma diferente de disfrutar de la música en vivo.	Necesita un mínimo de orientación sobre cómo utilizarla.
Onboarding	El proceso de inicio de la aplicación fue rápido y sencillo.	Falta información sobre la integración con *Setlist.fm*.
Funcionalidad de búsqueda de un evento por artista y/o recinto.	No hubo problemas al consultar la lista de eventos por artista y lugar.	Sería útil una opción de búsqueda rápida de eventos cercanos.
Funcionalidad de búsqueda de un evento por usuario de *Setlist.fm*.	N/A	N/A
Funcionalidad de conexión con *Spotify*.	No hubo problemas al iniciar sesión con Spotify.	Al conectar con el usuario de *Spotify*, no quedó claro si se conectó con éxito.
Funcionalidad creación de *playlist* en *Spotify*.	No hubo problemas al crear el *playlist* en Spotify.	Al crear un *playlist* en *Spotify*, no quedó claro si se creó con éxito.

2.1.4.3. Puntos de mejora.

Se han detectado varios puntos de mejora. A continuación, se presenta un listado argumentado de estos puntos de mejora, junto con la decisión sobre si se incluirán en el producto final o no:

- Puntos de Mejora Detectados:
  - **Claridad en el *Onboarding***: Los usuarios expresaron confusión durante el proceso de inicio de la aplicación en cuanto a que no quedaba claro el tipo de integración con *Spotify* y con *Setlist.fm*. Se necesita un *onboarding* más claro y descriptivo.
    Se incluirá en el producto final la mínima información necesaria para que quede claro este punto.

  - **Búsqueda Rápida de Eventos Cercanos**: Se sugirió la inclusión de una opción de búsqueda rápida de eventos cercanos para ayudar a los usuarios a encontrar conciertos en su área.
    Esta mejora no se incluirá en el producto final, ya que queda fuera del alcance de este proyecto. Se podría dejar pendiente para una posible actualización en un futuro.

- **Claridad en la conexión a usuario de Spotify**: Los usuarios señalaron que la aplicación no proporciona una confirmación clara después de conectarse a su usuario de *Spotify*.
  Se incluirá en el producto final una manera sencilla de saber si el usuario está conectado a Spotify. Por ejemplo, cambiando un botón de color o similar.

- **Claridad en la confirmación de la creación de *playlists***: Los usuarios señalaron que la aplicación no proporciona una confirmación clara después de crear una lista de reproducción en *Spotify*.
  Igual como en punto anterior, se incluirá en el producto final una manera sencilla de saber si el usuario ha creado satisfactoriamente un *playlist* en *Spotify*. Por ejemplo, cambiando un botón de color o similar.

- **Búsqueda de eventos concretos para usuarios de Setlist.fm**: Un usuario de *Setlist.fm* indicó que no pudo encontrar una forma de ver eventos pasados en un periodo determinado, ya que los resultados que se obtienen están ordenados cronológicamente de manera descendente y es complicado acceder a *setlists* antiguos si existen muchos eventos en la lista.
  Si el tiempo del proyecto lo permite se incluirá en el producto final. Se habilitará una función para filtrar eventos pasados.

Todos estos puntos de mejora se abordarán en la fase de implementación de la aplicación y contribuirán a una experiencia de usuario más satisfactoria y completa.

2.2. Diseño técnico.

2.2.1. Definición de los casos de uso.

2.2.1.1. Diagrama UML.

El diagrama de casos de uso UML representa gráficamente los actores y el flujo de los casos de uso mencionados en apartados anteriores.

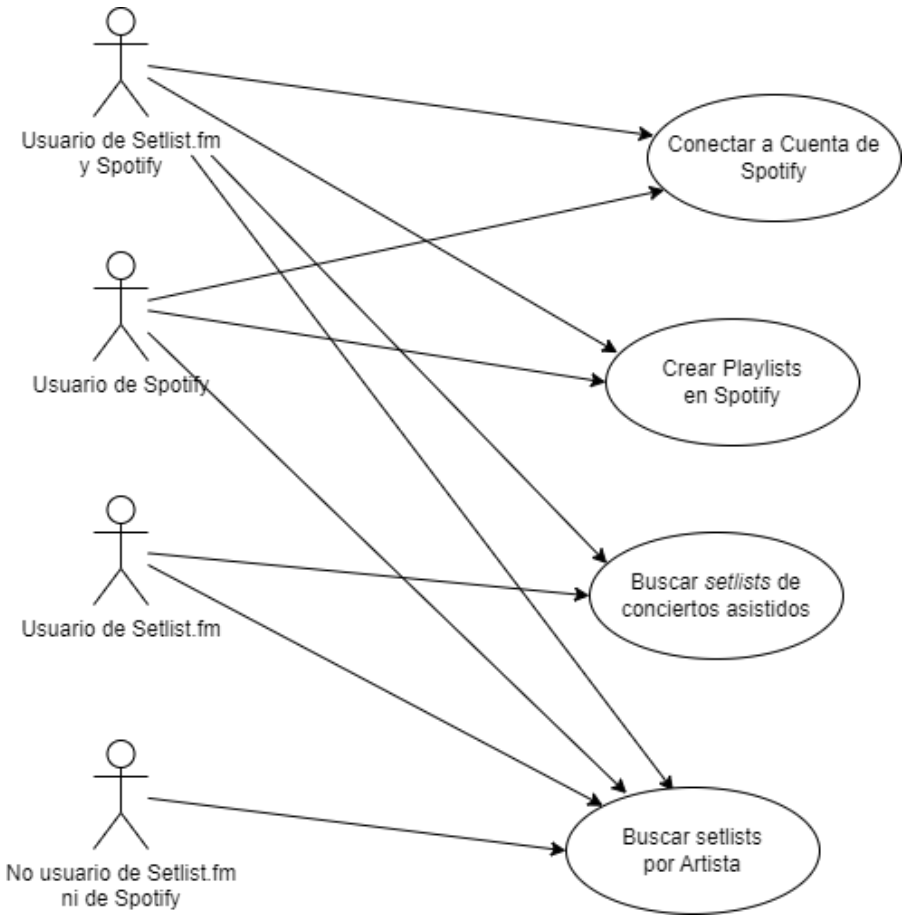


Ilustración 10: Diagrama UML Casos de uso

Los actores se representan en la parte izquierda.
Los casos de uso se muestran en la parte derecha.
Las flechas conectan los actores con los casos de uso en los que participan.

Se incluyen cuatro casos de uso:

- El proceso de conexión a *Spotify*
- La creación de una lista de reproducción en *Spotify* a partir del *setlist* de un concierto.
- La búsqueda de los eventos a los que ha asistido el usuario registrado en *Setlist.fm*.
- La búsqueda de eventos por artista. Pudiendo filtrar por recinto donde ha tenido lugar el concierto.

2.2.1.2. Casos de uso.

Los cuatro casos de uso representados en el apartado anterior son:

Identificador	CU-001
Nombre	Conexión a cuenta de *Spotify.*
Prioridad	Normal.
Descripción	Un usuario de la aplicación, que además está registrado en *Spotify*, puede integrar su cuenta para poder crear *playlists* directamente en su perfil de *Spotify*.
Actores	- Usuario de Setlist.fm y Spotify. - Usuario de Spotify.
Precondiciones	El usuario todavía no ha conectado su cuenta de *Spotify* con la aplicación.
Iniciado por	Usuario.
Flujo	1- El usuario abre la aplicación. 2- Pincha en el botón correspondiente a la conexión con *Spotify*. 3- Acepta los términos de *Spotify* y su cuenta queda vinculada.
Postcondiciones	El usuario ha finalizado el proceso de conexión con *Spotify* y ya puede crear *playlists*.
Notas	

Identificador	CU-002
Nombre	Creación de *playlist* en *Spotify.*
Prioridad	Normal.
Descripción	Un usuario de la aplicación, que además está registrado en *Spotify*, puede crear *playlists* directamente en su perfil de *Spotify*.
Actores	- Usuario de Setlist.fm y Spotify. - Usuario de Spotify.
Precondiciones	El usuario ya ha conectado su cuenta de *Spotify* con la aplicación.
Iniciado por	Usuario.
Flujo	1- El usuario abre la aplicación. 2- Realiza una búsqueda. 3- Escoge un *setlist* de los que aparecen en la lista y accede a él. 4- Pincha en el botón de crear *playlist*.
Postcondiciones	El usuario ya dispone del *playlist* creado en su cuenta de Spotify.
Notas	

Identificador	CU-003
Nombre	Búsqueda de *setlists* de eventos a los que se ha asistido.
Prioridad	Normal.
Descripción	Un usuario de la aplicación, que además está registrado en *Setlist.fm*, puede ver los conciertos a los que ha asistido
Actores	- Usuario de Setlist.fm y Spotify. - Usuario de Setlist.fm.
Precondiciones	El usuario no necesita conectar su cuenta de *Setlist.fm*. Solamente necesita su nombre de usuario de *Setlist.fm*.
Iniciado por	Usuario.
Flujo	1- El usuario abre la aplicación. 2- Realiza una búsqueda por nombre de usuario de Setlist.fm. 3- Escoge un *setlist* de los que aparecen en la lista y accede a él.
Postcondiciones	El usuario puede consultar las canciones del *setlist* escogido.
Notas	

Identificador	CU-004
Nombre	Búsqueda de *setlists* por artista.
Prioridad	Baja.
Descripción	Un usuario de la aplicación puede ver los conciertos realizados por un artista determinado.
Actores	- Usuario de Setlist.fm y Spotify. - Usuario de Setlist.fm. - Usuario de Spotify. - No usuario de Setlist.fm y Spotify.
Precondiciones	El usuario no necesita conectar su cuenta de *Setlist.fm* ni de *Spotify*.
Iniciado por	Usuario.
Flujo	1- El usuario abre la aplicación. 2- Realiza una búsqueda por nombre del artista. 3- Puede filtrar la búsqueda por recinto. 4- Escoge un *setlist* de los que aparecen en la lista y accede a él.
Postcondiciones	El usuario puede consultar las canciones del *setlist* escogido.
Notas	

2.2.2. Diseño de la arquitectura.

2.2.2.1. Diagrama UML del diseño de la base de datos.

En esta arquitectura, no se mantendrá una base de datos persistente, ya que los datos provienen de servicios externos. Se gestionarán los datos temporales en la memoria durante la sesión del usuario.

Ilustración 11: Diseño de la base de datos.

Se realizarán solicitudes a las *APIs* de *Spotify* y *setlist.fm*, que devolverán datos temporales, que luego serán procesados y presentados.

Este enfoque permite una arquitectura flexible y escalable, ya que la aplicación se integrará con servicios externos y no requerirá una base de datos propia. La comunicación con las *APIs* se realizará a través de solicitudes HTTP, y los datos se almacenarán temporalmente en la memoria del servidor.

Los únicos datos persistentes que se almacenarán serán los relativos a las claves (*tokens*) del usuario que conecte su cuenta de Spotify. En este caso, se hará uso de lo que se conoce como *LocalStorage.*

LocalStorage se refiere a un espacio de almacenamiento en el navegador que permite a las aplicaciones web guardar datos de forma persistente en el dispositivo del usuario. Es una manera de almacenar información localmente en la máquina del cliente, lo que significa que los datos se conservan incluso después de que el usuario cierre la aplicación o apague su dispositivo.

En *Ionic*, el *LocalStorage* se puede utilizar para guardar y recuperar datos clave-valor en forma de pares de datos (normalmente en formato de cadena).

2.2.2.2. Diagrama UML del diseño de las entidades y clases.

A continuación, se representa lo que será el diagrama básico de las clases y servicios que tendrá la aplicación.

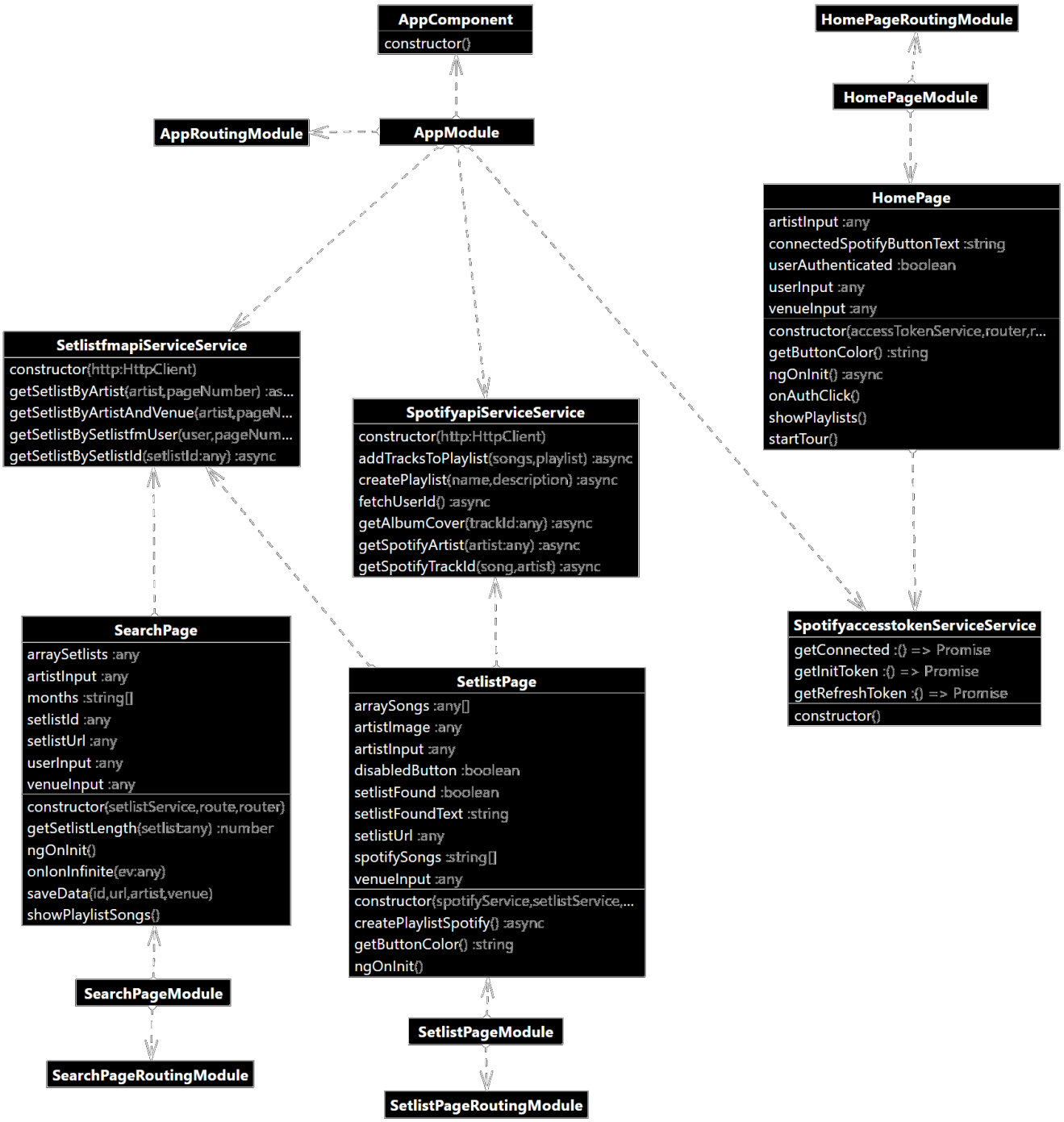


Ilustración 12: Diagrama UML del diseño de las entidades y clases.

2.2.2.3. Diagrama de la arquitectura del sistema.

La implementación de la aplicación se realizará en *Ionic Framework*, que es un marco que nos permite desarrollar aplicaciones nativas para *iOS*, *Android* y la web, desde una única base de código. Además, se integra perfectamente con los mejores marcos *frontend*, incluidos *Angular*, *React* o *Vue*. Este marco utiliza un patrón de diseño MVC (*Model-View-Controller*) de tres capas.

Esquemáticamente, aplicando el modelo anterior al sistema, se establece el siguiente flujo de información y procesamiento:

1. El usuario comienza a interactuar con el sistema, a través de la vista (interfaz de usuario), solicitando que realice una acción. Esta acción desencadena el inicio de un proceso (controlador), el cuál puede recibir información necesaria a partir de la vista.

2. El controlador toma entonces el control e invoca los métodos que resulten necesarios del modelo.

3. La capa del modelo es el encargado de recopilar la información necesaria para llevar a cabo el proceso. En caso necesario, se comunicará con la base de datos para recuperar los datos necesarios para ello y enviará dichos datos al controlador.

4. El controlador procesa entonces la información y envía los resultados a la vista.

5. La vista es en última instancia la encargada de, en caso necesario, dar formato a dichos datos y presentarlos al usuario.

El siguiente diagrama representa gráficamente este flujo a través de la arquitectura del sistema.

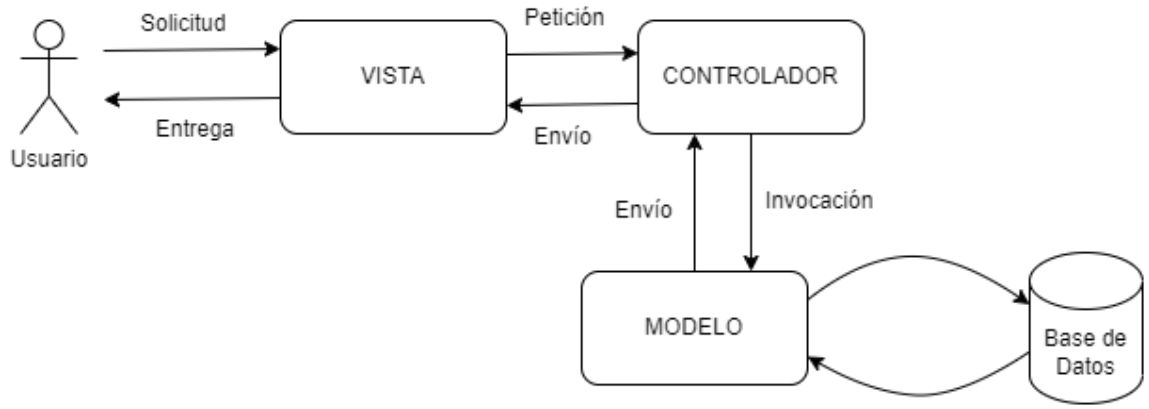


Ilustración 13: Diagrama de la arquitectura del sistema

3. Implementación.

3.1. Desarrollo.

3.1.1. Herramientas utilizadas para el desarrollo.

3.1.1.1. *Ionic framework*.

Se elige *Ionic Framework* como marco de desarrollo para aplicaciones híbridas.

Ilustración 14: Logo de Ionic Framework.

Ionic permite crear aplicaciones móviles utilizando tecnologías web estándar como *HTML*, *CSS* y *JavaScript*.

Se utiliza *Angular* como marco principal, proporcionando un conjunto de componentes reutilizables y una estructura para el desarrollo eficiente de aplicaciones.

Ilustración 15: Logo de Angular Framework.

Funcionalidades Principales:

- Creación de interfaces de usuario atractivas y responsivas.
- Acceso a funcionalidades nativas de dispositivos móviles a través de *Capacitor*.
- Integración fluida con Angular para una estructura de desarrollo consistente.

3.1.1.2. Capacitor.

Capacitor es otro *framework* mantenido por el equipo detrás de *Ionic Framework* y se utiliza en conjunción con éste para desarrollar aplicaciones multiplataforma.

Ilustración 16: Logo de Capacitor.

Algunas de las características clave de *Capacitor* incluyen:

- Acceso a Funcionalidades Nativas.
- Compatibilidad Multiplataforma.
- Estructura de Proyecto Clara.
- Integración con *Ionic*.
- Soporte para *PWA* (Aplicaciones Web Progresivas).

En definitiva, *Capacitor* actúa como un puente entre el código web y las capacidades nativas de los dispositivos móviles, proporcionando una manera eficiente de construir aplicaciones móviles híbridas con tecnologías web familiares.

3.1.1.3. Google Cloud.

Google Cloud ofrece una amplia gama de servicios que van desde bases de datos hasta servicios de aprendizaje automático.

Google Cloud se selecciona para la gestión de servicios en la nube. En este caso en concreto se alojarán gratuitamente dos servicios realizados con *Node.JS* y *Express* para no interactuar directamente con las *API's* de *Spotify* y *Setlist.fm* desde la aplicación.

Ilustración 17: Logo de Google Cloud.

Como resultado final se tendrá dos nuevas *API's*, desplegadas en Google Cloud, que actuaran de intermediarias (*Middleware*) entre la aplicación y las *API's* de *Spotify* y *Setlist.fm*.

En los siguientes enlaces se puede acceder a ambas *API's*:

- my-spotify-api
- my-setlistfm-api

3.1.1.4. Node.JS.

Node.js es un entorno de ejecución de código abierto que permite ejecutar *JavaScript* en el lado del servidor. Fue creado por Ryan Dahl en 2009 y está diseñado para facilitar el desarrollo de aplicaciones web escalables y de alto rendimiento. *Node.js* utiliza el motor V8 de *Google Chrome* para ejecutar *JavaScript* de manera eficiente.

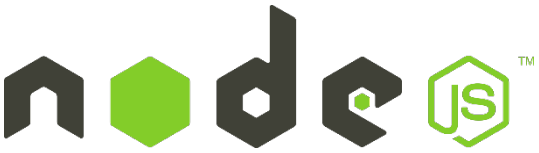


Ilustración 18: Logo de Node.JS.

Algunos puntos clave sobre Node.js:

- JavaScript en el Servidor: JavaScript era principalmente conocido por ejecutarse en el navegador del cliente. *Node.js* permite utilizar *JavaScript* para escribir scripts del lado del servidor.

- *Event-Driven* y Asíncrono: Se basa en un modelo de programación no bloqueante y basado en eventos. Las operaciones de entrada/salida (E/S), como la lectura de archivos o las consultas a bases de datos, se realizan de manera asíncrona, lo que permite manejar un gran número de conexiones concurrentes sin bloquear el hilo de ejecución.

- Módulos y *NPM*: Utiliza un sistema de módulos que permite organizar el código en archivos separados y reutilizables. Además, *Node.js* viene con *npm* (*Node Package Manager*), una herramienta que facilita la instalación, gestión y compartición de bibliotecas y paquetes de código.

- Amplia Comunidad y Ecosistema: *Node.js* ha ganado una amplia adopción y cuenta con una comunidad activa. Existe un amplio ecosistema de módulos y bibliotecas disponibles a través de *npm*.

- Escalabilidad y Rendimiento: Es conocido por su capacidad para manejar un gran número de conexiones simultáneas con eficiencia, lo que lo hace adecuado para aplicaciones escalables en tiempo real.

- Versatilidad: *Node.js* es versátil y se utiliza para desarrollar una variedad de aplicaciones, desde servidores web y servicios *RESTful* hasta herramientas de línea de comandos y scripts del lado del servidor.

3.1.1.5. Express.JS.

Express.js, comúnmente conocido como *Express*, es un *framework* web para *Node.js* que simplifica y agiliza el desarrollo de aplicaciones web y servicios *RESTful*. Es minimalista, flexible y ofrece un conjunto de características que facilitan la creación de servidores web robustos y escalables en *Node.js*.

Ilustración 19: Logo de Express.JS.

Algunas características clave de Express incluyen:

- *Routing*: *Express* facilita la definición de rutas para manejar diferentes solicitudes *HTTP*.

- *Middlewares*: Los *middlewares* son funciones que tienen acceso al objeto de solicitud (*req*), al objeto de respuesta (*res*) y a la función siguiente en el ciclo de solicitud-respuesta del servidor. *Express* utiliza *middlewares* para realizar diversas tareas, como el análisis del cuerpo de la solicitud, la autenticación, la gestión de sesiones, entre otras.

- *Templates*: *Express* facilita la renderización de vistas mediante el soporte para diferentes motores de plantillas. Esto permite generar respuestas HTML dinámicas basadas en datos.

- *Manejo de Errores*: *Express* permite capturar y manejar errores de manera centralizada. Esto mejora la legibilidad del código y facilita la gestión de errores en toda la aplicación.

- *Static Files*: *Express* simplifica la entrega de archivos estáticos, como imágenes, hojas de estilo y archivos *JavaScript*.

- *RESTful Routing*: *Express* proporciona un enfoque sencillo y efectivo para la creación de servicios web *RESTful*. Las rutas y los controladores pueden organizarse de manera lógica para manejar las operaciones *CRUD* (Crear, Leer, Actualizar, Eliminar).

- *Extensible*: Aunque *Express* en sí mismo es liviano, es extensible mediante el uso de *middleware* y módulos adicionales. Esto permite personalizar y agregar funcionalidades según las necesidades las aplicaciones.

3.1.1.6. Spotify API.

La *API* de *Spotify* se utiliza para interactuar con la plataforma de música *Spotify*. Proporciona funciones para buscar canciones, acceder a listas de reproducción y gestionar la biblioteca de un usuario.

Ilustración 20: Logo de Spotify for Developers.

En el siguiente enlace se puede encontrar la documentación asociada a la *API*:

- developer.spotify.com

3.1.1.7. Setlist.fm API.

La *API* de *setlist.fm* se utiliza para obtener información sobre *setlists* de conciertos. Proporciona datos detallados sobre las canciones interpretadas en eventos en vivo.

Ilustración 21: Logo de Setlist.FM.

En el siguiente enlace se puede encontrar la documentación asociada a la *API*:

- api.setlist.fm

3.1.1.8. IDE: Visual Studio Code.

Visual Studio Code (*VS Code*) se elige como entorno de desarrollo integrado (*IDE*) principal.

Ilustración 22: Logo de Visual Studio Code.

Es una herramienta ligera pero poderosa que admite una variedad de lenguajes de programación y ofrece numerosas extensiones para facilitar el desarrollo.

Funcionalidades Principales:

- Resaltado de sintaxis y sugerencias inteligentes.
- Integración con *Git* para el control de versiones.
- Amplia comunidad y soporte de extensiones.

3.1.1.9. Git.

Git es un sistema de control de versiones distribuido (*DVCS*, por sus siglas en inglés) diseñado para rastrear cambios en el código fuente durante el desarrollo de software. Fue creado por Linus Torvalds en 2005 y se ha convertido en una herramienta fundamental en el desarrollo colaborativo de proyectos de software.

Ilustración 23: Logo de Git.

Algunos conceptos clave de *Git* incluyen:

- Control de Versiones: *Git* permite realizar un seguimiento de los cambios en los archivos a lo largo del tiempo.

- Distribuido: *Git* es distribuido, lo que significa que cada colaborador tiene una copia completa del repositorio, incluyendo todo el historial de cambios. Esto facilita la colaboración y permite trabajar localmente sin necesidad de estar constantemente conectado a un servidor central.

- Ramas (*Branches*): Git utiliza un modelo de ramificación que permite trabajar en funcionalidades o correcciones de errores de manera independiente sin afectar la rama principal del código (normalmente llamada *master* o *main*). Las ramas se pueden fusionar cuando las características están completas y probadas.

- Repositorios: *Git* almacena la información en repositorios. Un repositorio *Git* puede vivir localmente en una máquina o puede estar almacenado en un servidor remoto. Los repositorios remotos permiten la colaboración y el intercambio de código entre desarrolladores.

- Commit: Un *commit* en *Git* representa un conjunto de cambios en el código. Los *commits* son versiones atómicas que pueden revertirse o revisarse individualmente. Cada *commit* tiene un mensaje descriptivo que proporciona información sobre los cambios realizados.

3.1.1.10. GitHub.

GitHub es una plataforma de desarrollo colaborativo basada en *Git*, el sistema de control de versiones distribuido. Fundada en 2008, *GitHub* proporciona herramientas y servicios que facilitan la colaboración en proyectos de software y la gestión eficiente del código fuente.

Ilustración 24: Logo de GitHub.

3.1.2. Justificación de la elección del entorno de desarrollo.

La elección del entorno de desarrollo es una decisión crítica en cualquier proyecto de ingeniería informática, ya que influye directamente en la eficiencia del desarrollo, la calidad del código y la escalabilidad del sistema. En el caso de nuestro proyecto de desarrollo de una aplicación híbrida para obtener *setlists* de la *API setlist.fm* y convertirlos en *playlists* de *Spotify*, la elección del entorno de desarrollo se basa en criterios fundamentales para garantizar el éxito y la sostenibilidad del proyecto.

- *Ionic Framework y Angular*:

Ionic Framework junto con *Angular* proporciona un marco de desarrollo eficiente que permite un desarrollo rápido y estructurado. *Angular*, como un marco *MVC* robusto, facilita la creación de interfaces de usuario atractivas y funcionales.

Esta combinación permite la creación de aplicaciones que son compatibles con múltiples plataformas, incluyendo iOS, Android y la web. Esto asegura que la aplicación sea accesible para un amplio espectro de usuarios.

- *Google Cloud*:

Google Cloud fue seleccionado por su capacidad de escalabilidad y la variedad de servicios que ofrece. Pero principalmente porque su versión gratuita permite desplegar el middleware necesario para el correcto funcionamiento de la aplicación.

- *Spotify API y Setlist.fm API*:

La elección de las *APIs* de *Spotify* y *setlist.fm* se basa en la necesidad de acceder a datos específicos para cumplir con los objetivos del proyecto. Estas *APIs* proporcionan funciones esenciales para obtener información sobre canciones, álbumes y *setlists* de conciertos.

- *Visual Studio Code*:

Visual Studio Code se seleccionó como el entorno de desarrollo principal debido a su ligereza y potencia. Ofrece funciones como resaltado de sintaxis, sugerencias inteligentes y una integración fluida con Git, lo que mejora la productividad y la calidad del código.

3.1.3. Análisis del estado del proyecto.

Llegado este punto de implementación, el proyecto avanza dentro de los márgenes de tiempo establecidos al inicio de este. La aplicación se encuentra en un estado totalmente funcional, solamente a falta de las correspondientes pruebas.

El estado actual del producto es:

• *Onboarding*: Se ha implementado una pequeña guía de para que sirve y como utilizar la aplicación. Se accede a ella desde la página de inicio mediante un pequeño icono en forma de interrogante situado en la parte superior derecha de la pantalla.

• Página inicial (*Home*). Punto de entrada a la aplicación. Se han incluido los campos de texto necesarios para poder realizar las búsquedas de *setlists*.

• Autenticación en la plataforma de *Spotify*: Totalmente implementado. Se dispone ya de la interfaz para poder conectar la cuenda de *Spotify* del usuario.

• Página de resultados de la búsqueda (*Search*). En esta página se pueden ver mediante la funcionalidad *Infinite scroll*, todos los *setlists* que se ajustan a los parámetros de la búsqueda ordenados decrecientemente por fecha del evento. Se incluye un icono para volver siempre a la página anterior, y un icono en cada registro obtenido para poder acceder al detalle del *setlist* escogido. Con esto se pretende facilitar la navegación entre páginas.

• Página de detalle del *setlist* escogido (*Setlist*). En esta página se ven las canciones interpretadas en el evento escogido. Así mismo se incluye un botón para crear el playlist con dichas canciones en la cuenda de Spotify del usuario, siempre que éste disponga de una. También se incluye un icono para volver a la página anterior.

En lo que respecta a las dos *API's* que actuarán de *middleware*, se han realizado los *endpoints* necesarios para el alcance del proyecto.

Uno de los aspectos más problemáticos ha sido el hecho de que la *API* de *Spotify*, mientras la aplicación a implementar consta en su plataforma como en modo desarrollo, no permite utilizar su *API* desde la aplicación a ninguna cuenta de *Spotify* que no esté autorizada en el portal para desarrolladores. Por este motivo, y para facilitar las pruebas de personas ajenas al proyecto, se ha creado una cuenta de *Spotify* de prueba que será deshabilitada una vez finalice el periodo de pruebas.

Usuario:	vicgora@me.com
Password:	TestLiveList!!

Como punto final, se ha conseguido plasmar casi por completo el prototipo presentado, algunos detalles no son exactamente iguales, pero en la gran mayoría de páginas el diseño es idéntico.

3.2. Pruebas

3.2.1. Pruebas con usuarios reales.

Para la prueba y posterior optimización de la aplicación se hará uso de familiares y/o amigos cercanos, los cuales probarán la aplicación mediante dispositivos *Android*, el navegador *Google Chrome* y con el simulador de *Android Studio*. También se intentará hacer pruebas en un dispositivo iOS, aunque este caso es más difícil por el hecho de que para poder simular la aplicación en el entorno *Apple* hay que pagar una suscripción.

A los probadores se les ha pedido que testeen la aplicación en los diferentes escenarios de uso presentados con anterioridad, así quedarán probadas situaciones similares a la realidad.

• *Onboarding*: La funcionalidad implementada como *onboarding* funciona correctamente. Al pinchar sobre el icono en forma de interrogante aparecen una cadena de mensajes, que, junto con una serie de ayudas visuales, facilitan que el usuario comprenda el funcionamiento de la aplicación.

• Menús: La navegación en la aplicación es correcta. Los diferentes usuarios han navegado por la aplicación sin ningún tipo de problema.

• Conexión a Spotify: La conexión a Spotify funciona correctamente, aunque se ha detectado que el usuario solo aparece como conectado la segunda vez que se realiza la conexión. Se intentará modificar esta anomalía.

• Búsqueda de un *setlist*: Esta prueba se ha realizado con éxito en el 100% de los casos. Tanto en las búsquedas por artista y/o recinto, como en las búsquedas por nombre de usuario de *Setlist.fm*. En todos los casos, si existe algún *setlist* se muestra una lista como resultado, y en caso de no haber ninguno, se muestra una alerta y se vuelve a la página principal.

• Creación *playlist* en Spotify: También ha funcionado correctamente. Hay que tener en cuenta que en este caso solo se añaden al *playlist* las canciones que son del artista. Las que no pertenecen al artista, como versiones, solos, etc., no se añadirán al *playlist*, ya que esto hacía más laboriosa la implementación.

3.2.2. Pruebas unitarias.

Las pruebas unitarias desempeñan un papel fundamental en el desarrollo de software, ya que permiten verificar el comportamiento individual de componentes o unidades de código. En el contexto de la aplicación móvil desarrollada, las pruebas unitarias se centran en evaluar el funcionamiento a nivel de código de módulos específicos, funciones y clases.

El objetivo principal de las pruebas unitarias es garantizar que cada unidad de código funcione como se espera, identificando posibles errores y asegurando la calidad y fiabilidad del software. Al realizar pruebas unitarias, se busca validar el comportamiento de las funciones y métodos en diferentes escenarios, incluyendo casos de uso esperados e inesperados.

Para llevar a cabo las pruebas unitarias en la aplicación móvil desarrollada, se utilizarán las herramientas *Karma* y *Jasmine*.

Karma es un entorno de pruebas que permite ejecutar pruebas unitarias en diferentes navegadores y plataformas. *Karma* se integra con *Jasmine*, un *framework* de pruebas unitarias para *JavaScript* que proporciona una sintaxis clara y fácil de entender para escribir pruebas.

Jasmine se basa en la definición de "especificaciones" que describen el comportamiento esperado de una unidad de código. Cada especificación se compone de una o varias "expectativas" que definen las condiciones que deben cumplirse para que la especificación sea considerada exitosa.

Para escribir pruebas unitarias con *Jasmine*, se definen *suites* que agrupan las especificaciones relacionadas con una unidad de código específica. Dentro de cada *suite*, se definen las especificaciones utilizando la sintaxis de *Jasmine*, que incluye funciones como *describe*, *it*, *expect*, entre otras.

Karma se encarga de ejecutar las pruebas definidas en *Jasmine* en diferentes navegadores y plataformas, proporcionando información detallada sobre el resultado de cada prueba. Además, *Karma* permite automatizar el proceso de ejecución de pruebas, lo que facilita la integración de las pruebas unitarias en el proceso de desarrollo.

Para ejecutar las pruebas unitarias utilizando *Karma* y *Jasmine*, se deben seguir los siguientes pasos:

- Abrir una terminal en el directorio raíz del proyecto.
- Ejecutar el comando “*npm install”* para instalar las dependencias necesarias.
- Ejecutar el comando “*npm run test”* para iniciar la ejecución de las pruebas unitarias.
- *Karma* abrirá automáticamente un navegador y ejecutará las pruebas definidas en *Jasmine*.
- Una vez finalizada la ejecución de las pruebas, *Karma* mostrará un resumen de los resultados en la terminal.

En total, en este caso se han realizado 22 pruebas unitarias, que consiguen comprobar desde el correcto funcionamiento de los componentes y servicios, hasta alguna de las funcionalidades de cada apartado.

En la siguiente imagen se puede ver como las 22 pruebas unitarias han finalizado satisfactoriamente.

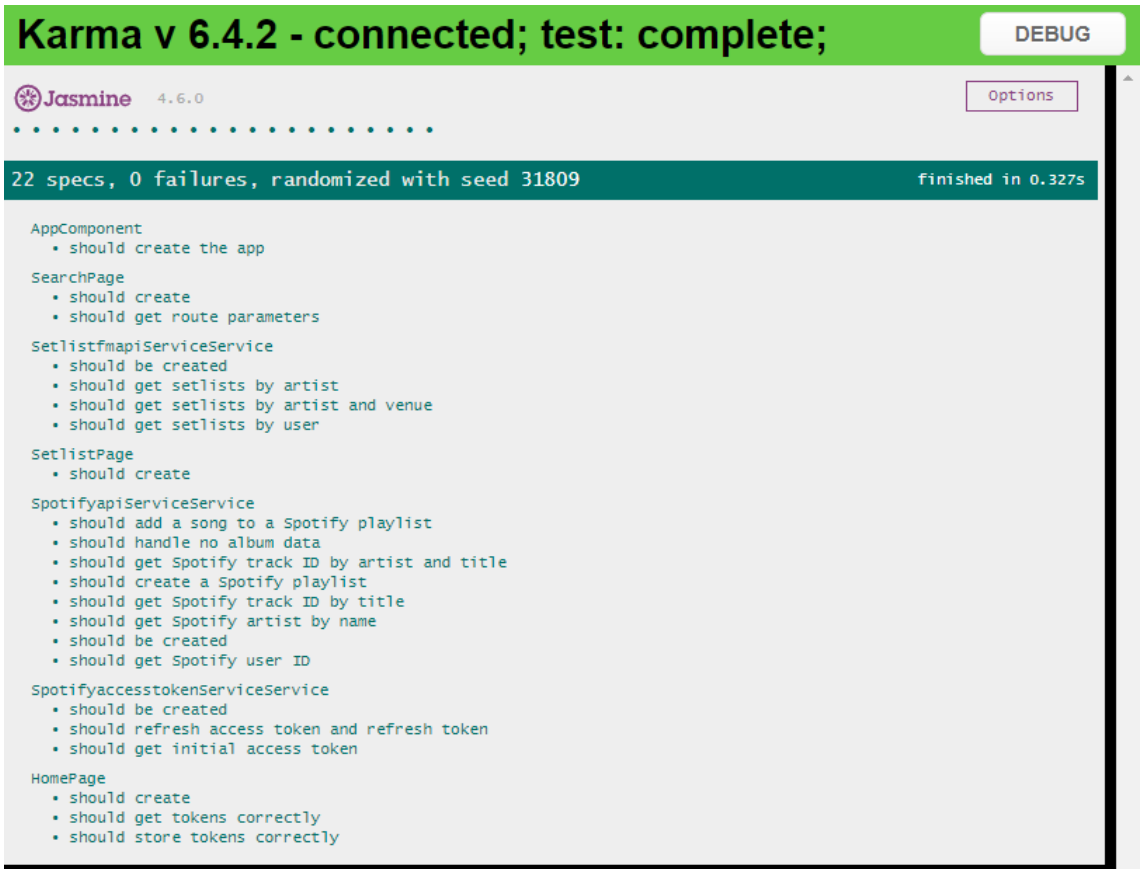


Ilustración 25: Pruebas unitarias.

Seguramente serían necesarias más pruebas unitarias para poder cubrir al 100% el alcance del proyecto ejecutado.

4. Conclusiones

El desarrollo de *LiveList* ha sido una experiencia muy enriquecedora para mí. A través de este proyecto, he aprendido mucho sobre el desarrollo de aplicaciones móviles híbridas, la integración de *APIs* y la importancia del diseño centrado en el usuario. Aunque la aplicación no ha sido llevada a producción, se ha conseguido completar a nivel funcional y se podría identificar como una fase *Beta* con ciertos aspectos que encajarían en una fase *Alpha*.

El trabajo ha sido satisfactorio y ha demostrado mis capacidades para el desarrollo de aplicaciones. Además, los conocimientos adquiridos durante los años de estudio del grado de ingeniería informática en la UOC han sido fundamentales para el desarrollo de este proyecto. La metodología en cascada ha sido muy útil para dividir el proyecto en diferentes fases secuenciales y asegurarme de que cada fase se completara antes de pasar a la siguiente. La capacidad de aprender de manera autónoma y la habilidad para resolver problemas han sido también habilidades clave para el éxito de este proyecto.

De cara al futuro, se espera completar los objetivos planteados y conseguir un sistema de datos capaz de ser utilizado para el beneficio de los usuarios de *LiveList*. Además, se espera añadir mayor seguridad a las *APIs* y hacer mejoras a nivel estructural.

Para resumir, el desarrollo de *LiveList* ha sido una experiencia muy gratificante y ha demostrado la importancia de la formación continua y la capacidad de adaptación en el campo de la informática. Estoy muy agradecido por la oportunidad de poder llevar a cabo este proyecto y espero seguir desarrollando mis habilidades en el futuro.

5. Glosario

- **API**: Interfaz de Programación de Aplicaciones. Conjunto de reglas y especificaciones que las aplicaciones pueden seguir para comunicarse entre sí.
- **Setlist**: Lista de canciones interpretadas en un concierto o evento musical.
- **Playlist**: Lista de reproducción de audio que contiene una secuencia de canciones.
- **Framework**: Entorno de trabajo que proporciona una estructura y conjunto de herramientas para el desarrollo de software.
- **Frontend**: Parte de una aplicación o sistema que interactúa directamente con los usuarios.
- **Backend**: Parte de una aplicación o sistema que no es visible para los usuarios y que gestiona la lógica y la manipulación de datos.
- **Spotify**: Plataforma de streaming de música.
- **Setlist.fm**: Base de datos colaborativa de setlists de conciertos.
- **Tecnologías web**: Conjunto de tecnologías utilizadas para el desarrollo de aplicaciones web, como HTML, CSS, JavaScript, etc.
- **JavaScript**: Lenguaje de programación ampliamente utilizado para el desarrollo web.
- **TypeScript**: Lenguaje de programación de código abierto desarrollado y mantenido por Microsoft.
- **HTML**: Lenguaje de marcado utilizado para el desarrollo de páginas web.
- **CSS**: Lenguaje utilizado para definir la presentación de un documento escrito en HTML.
- **Angular**: Framework de desarrollo de aplicaciones web con JavaScript/TypeScript, mantenido por Google.
- **Ionic**: Framework de desarrollo de aplicaciones móviles híbridas, que utiliza tecnologías web como HTML, CSS y JavaScript.
- **Testing**: Proceso de evaluación de un sistema o componente para determinar si cumple con los requisitos especificados.
- **Depuración**: Proceso de identificación y corrección de errores o defectos en un software.

- **Desarrollo multiplataforma**: Creación de aplicaciones que pueden ser implementadas en múltiples plataformas, como iOS, Android, etc.
- **Repositorio**: Espacio de almacenamiento donde se guarda y gestiona el código fuente de un proyecto.

6. Bibliografía

Ionic Company (2023). *Ionic Docs*. Disponible en: https://ionicframework.com/docs (Consultado: 11 de diciembre 2023).

Google (2023). *Introduction to the Angular Docs*. Disponible en: https://angular.io/docs (Consultado: 11 de diciembre 2023).

Spotify (2023). *Spotify for Developers*. Disponible en: https://developer.spotify.com/ (Consultado: 11 de diciembre 2023).

Setlist.fm (2023). *Setlist.fm API*.Disponible en: https://api.setlist.fm/docs/1.0/index.html (Consultado: 11 de diciembre 2023).

Ravulavaru, A. (2017). *Learning ionic: build hybrid mobile applications with HTML5, SCSS, and angular.* (Second edition.). Packt.

Griffith, C. J. (Christopher J. (2017). *Mobile app development with ionic: cross-platform apps with ionic, angular, and cordova.* (Revised edition.). O'Reilly.

Khanna, R., Yusuf, S., & Phan, H. (2017). *Ionic: hybrid mobile app development: create cutting-edge, hybrid mobile applications using the ionic framework: a course in three modules.* (1st edition). Packt Publishing.

Boada Oriols, M., Gómez Gutiérrez J.A. (2019). *El gran libro de Angular*. (Primera edición). Alfaomega.

Freeman, A. (2022). *Pro Angular: build powerful and dynamic web apps.* (Fifth edition.). Apress L. P. https://doi.org/10.1007/978-1-4842-8176-5

Jasmine (2023). *Jasmine. Simple Javascript testing*. Disponible en: https://jasmine.github.io/ (Consultado: 11 de diciembre 2023)

Karma (2023). *Karma. Installation*. Disponible en: https://karma-runner.github.io/6.4/index.html (Consultado: 11 de diciembre 2023)

7. Anexos

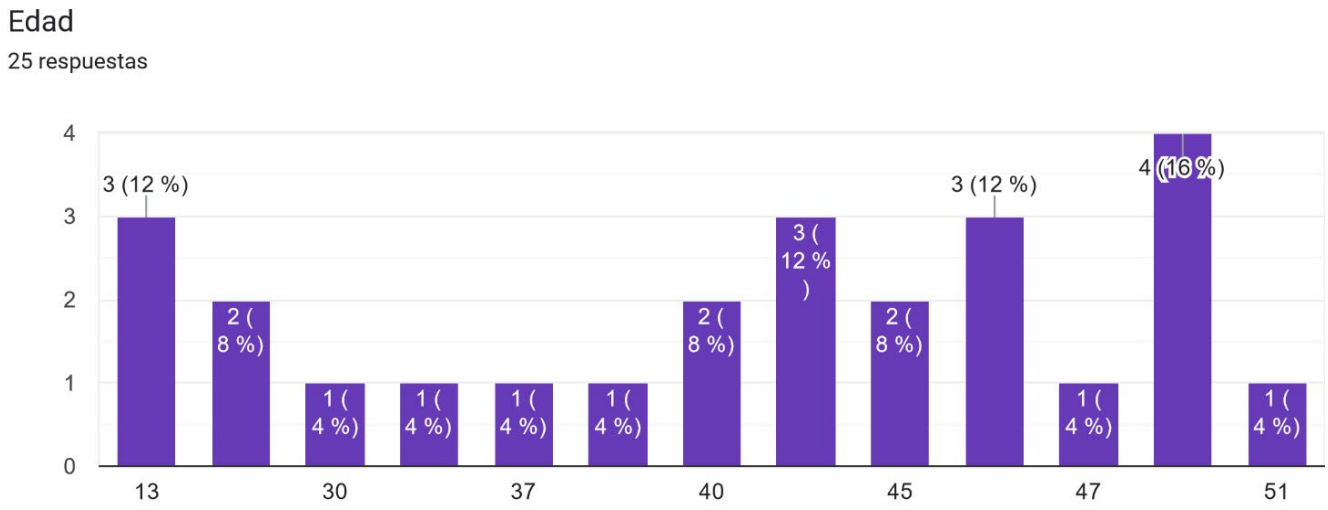
7.1. Anexo 1: Resultados de las encuestas a usuarios.

Las encuestas se realizarán mediante la aplicación *Google Forms*. Para el diseño de la encuesta se han tenido en cuenta las entrevistas realizadas anteriormente.

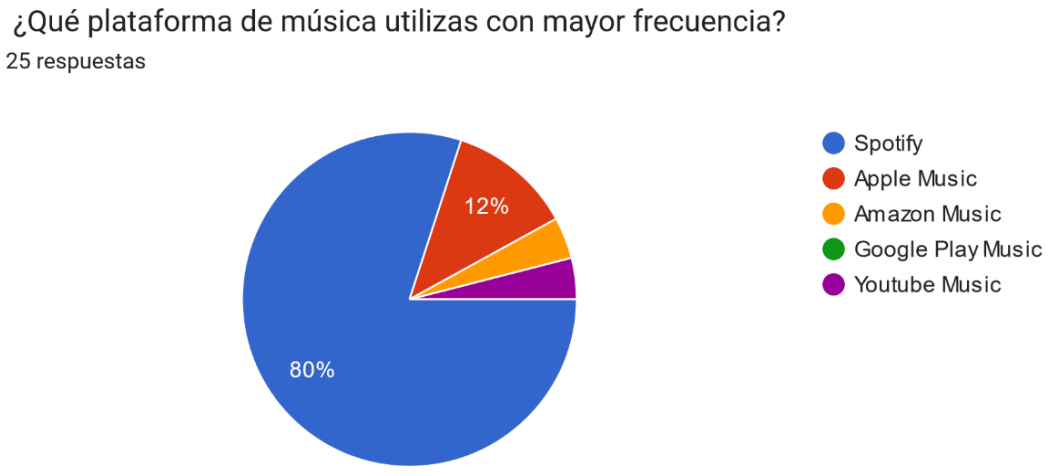
Se podrá acceder a la encuesta mediante el siguiente enlace:

Encuesta LiveList

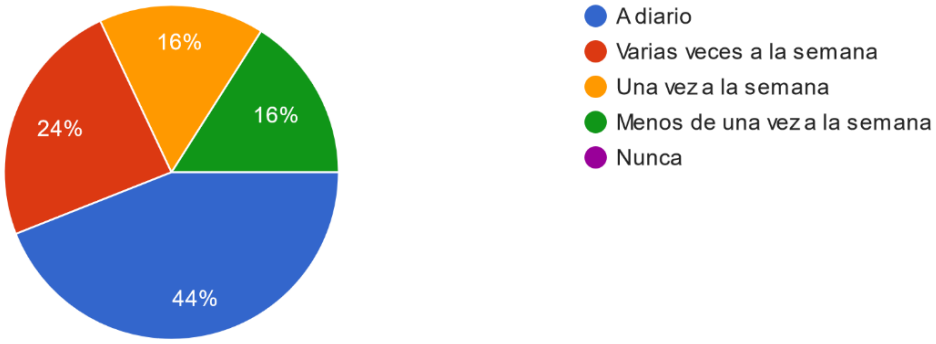
Una vez realizada la encuesta a un total de 25 personas, se obtienen los resultados siguientes:



La media de edad es de 37,48 años. El rango de edades ha sido de 13 a 51 años.



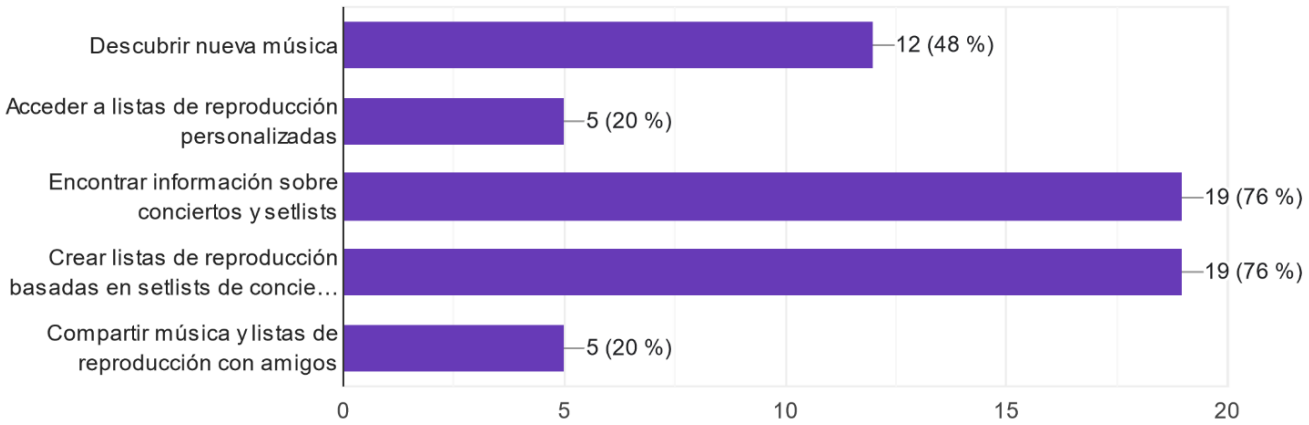
¿Con qué frecuencia utilizas aplicaciones de música como Spotify, Apple Music u otras?
25 respuestas



Se puede apreciar que todos los encuestados utilizan en mayor o menor medida servicios de música en *streaming*, en su mayoría utilizan concretamente *Spotify*. Un 68% de los encuestados utiliza con bastante frecuencia aplicaciones para escuchar música.

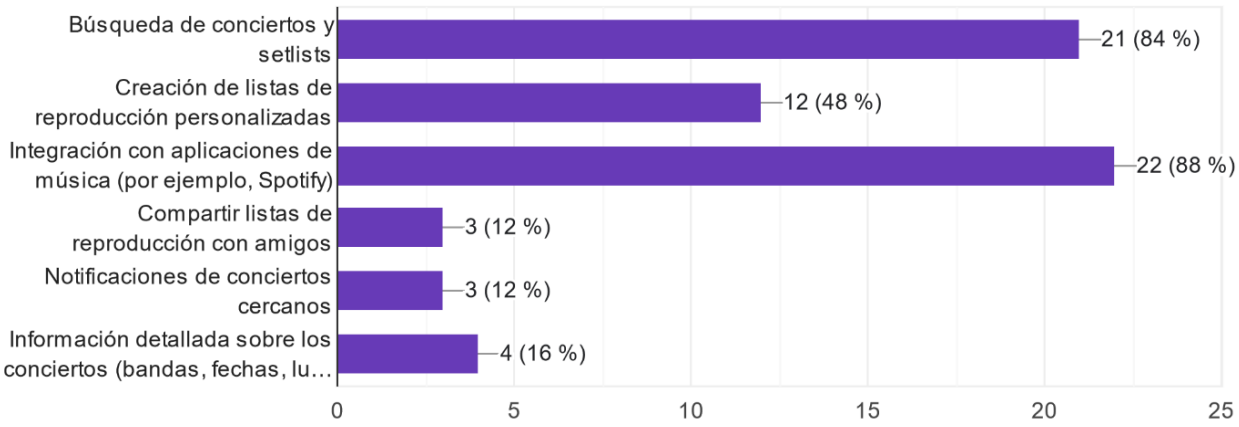
En cuanto a las necesidades y características necesarias según las encuestas, tenemos:

¿Cuáles son tus principales necesidades o desafíos relacionados con la música y los conciertos? (Selecciona todas las que correspondan)
25 respuestas



¿Qué características te gustaría ver en una aplicación que convierte setlists de conciertos en listas de reproducción de música? (Selecciona todas las que correspondan)

25 respuestas



Ha habido dos encuestados que han contestado a la pregunta libre de si tenían alguna sugerencia:

¿Tienes alguna otra sugerencia o comentario sobre lo que esperas de una aplicación como LiveList?

2 respuestas

Que sea sencilla de utilizar

Que tenga un interfaz amigable.

7.2. Anexo 2: Ejecución de una Aplicación *Ionic* en un navegador

Para ejecutar la aplicación desarrollada con *Ionic* en un navegador y poder probarla, se deben seguir los siguientes pasos:

- Instalación de Ionic y Node.js:

Es necesario tener instalado *Node.js* en el sistema. Si no está instalado, se puede descargar desde el sitio web oficial de *Node.js*. Una vez instalado *Node.js*, se puede instalar *Ionic* globalmente utilizando el siguiente comando en la terminal o línea de comandos:

```
npm install -g @ionic/cli
```

- Creación de un Nuevo Proyecto:

Si el proyecto aún no ha sido creado, se puede crear un nuevo proyecto de *Ionic* utilizando el siguiente comando en la terminal o línea de comandos:

```
ionic start nombre_del_proyecto
```

- Navegación al Directorio del Proyecto:

Una vez creado el proyecto, se debe navegar al directorio del proyecto utilizando el comando ***cd*** en la terminal o línea de comandos:

```
cd nombre_del_proyecto
```

- Ejecución en un Navegador:

Para ejecutar la aplicación en un navegador, se puede utilizar el siguiente comando en la terminal o línea de comandos:

```
ionic serve
```

Este comando iniciará un servidor de desarrollo y abrirá la aplicación en un navegador web predeterminado.

NOTA: En el caso de *Livelist,* se ha utilizado la librería ***InAppBrowser***, que los navegadores no son capaces de gestionar, por lo que al intentar la autenticación en Spotify se obtiene un error. Pero se puede apreciar en la url devuelta que la autenticación es exitosa.

- Prueba:

Una vez que la aplicación se esté ejecutando en el navegador, se puede interactuar con ella para probar su funcionalidad.

7.3. Anexo 3: Compilación de la Aplicación en Android.

- Software necesario para ejecutar la aplicación.

Para poder compilar y ejecutar la aplicación serán necesarios los siguientes programas:

- *Google Chrome*
- *Visual Studio Code* (o similar).
- *Git*
- *Node.js*
- *Android Studio*
- *Angular*
- *Ionic Framework*

- Abrir el proyecto.

Una vez instaladas las herramientas mencionadas, se descomprimirá el archivo *Livelist-master.zip* en el disco local.

Mediante el programa *Visual Studio Code* se podrá abrir la carpeta *Livelist-master* y navegar a través de los archivos del proyecto.

- Compilación y ejecución en Android Studio

Con el proyecto abierto en *Visual Studio Code* se abrirá una terminal

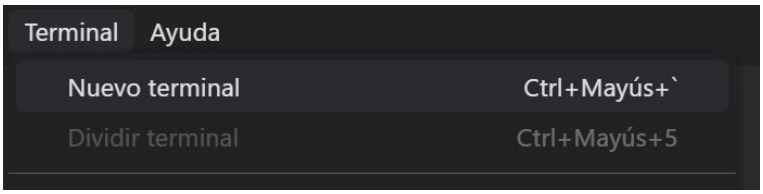


Ilustración 26: Abrir terminal en Visual Studio Code.

y se instalarán las dependencias del proyecto con el siguiente comando:

npm install

A continuación, para generar el proyecto Android a partir del código base de *Ionic*, se ejecutarán los comandos:

ionic build
ionic capacitor add android
ionic capacitor copy android

ionic capacitor build android

Con esto se generará una carpeta llamada *android* en la raíz del proyecto.

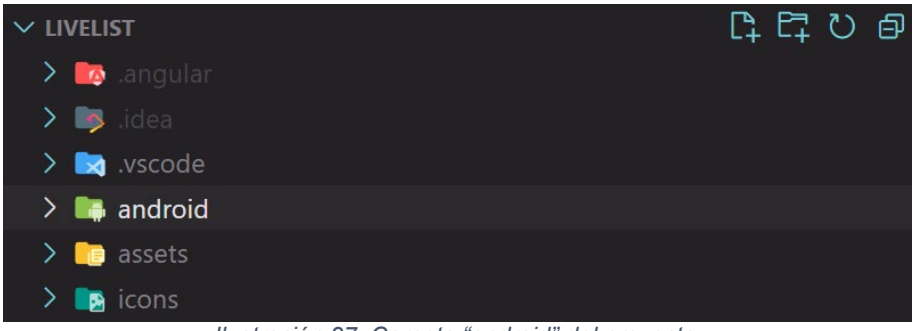


Ilustración 27: Carpeta "android" del proyecto.

En esa carpeta se encontrará el proyecto compilado para ejecutarlo en *Android Studio*.

Para compilar y ejecutar la aplicación en un dispositivo virtual o físico Android se seguirán los siguientes pasos:

- Instalación del SDK (*Software Development Kit*) de Android.

Se abrirá el ***SDK manager*** desde el menú ***Tools***:

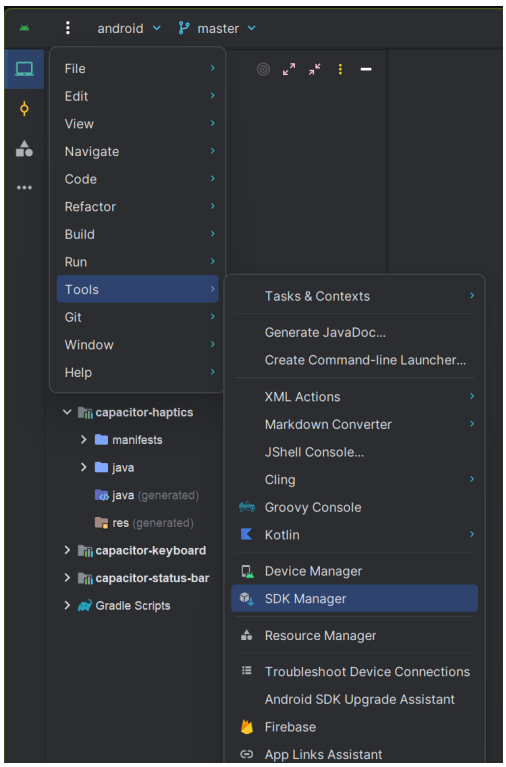


Ilustración 28: SDK Manager menú.

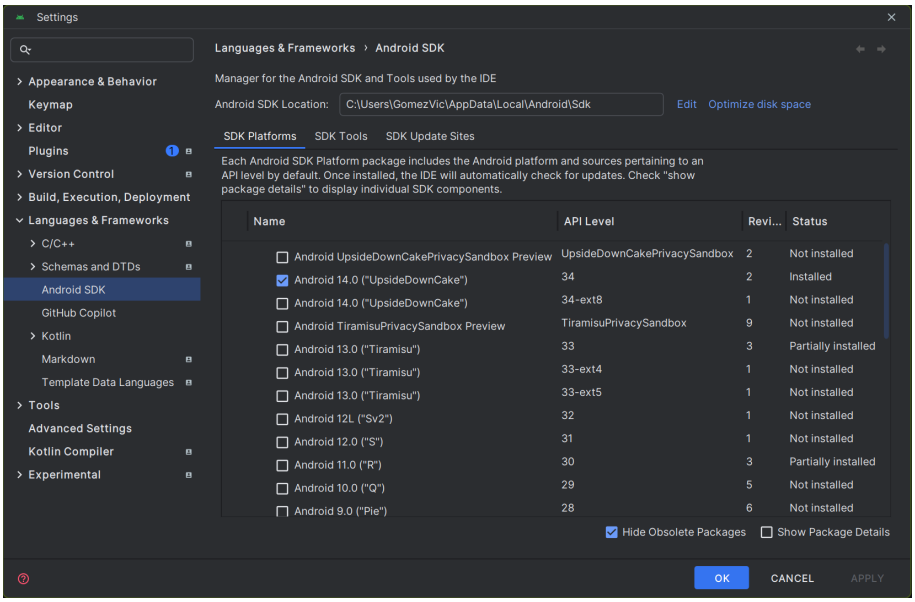


Ilustración 29: Android SDK Manager.

Se recomienda descargar la versión más actual.

- Creación de un dispositivo virtual Android.

Para crear un dispositivo virtual se abrirá el gestor de dispositivos (***Device Manager***) desde el menú ***Tools***:

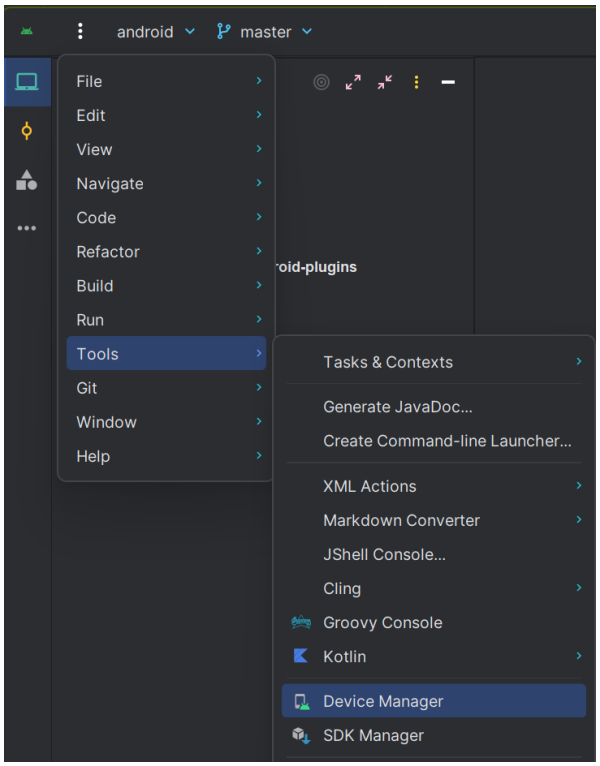


Ilustración 30: Device Manager menú.

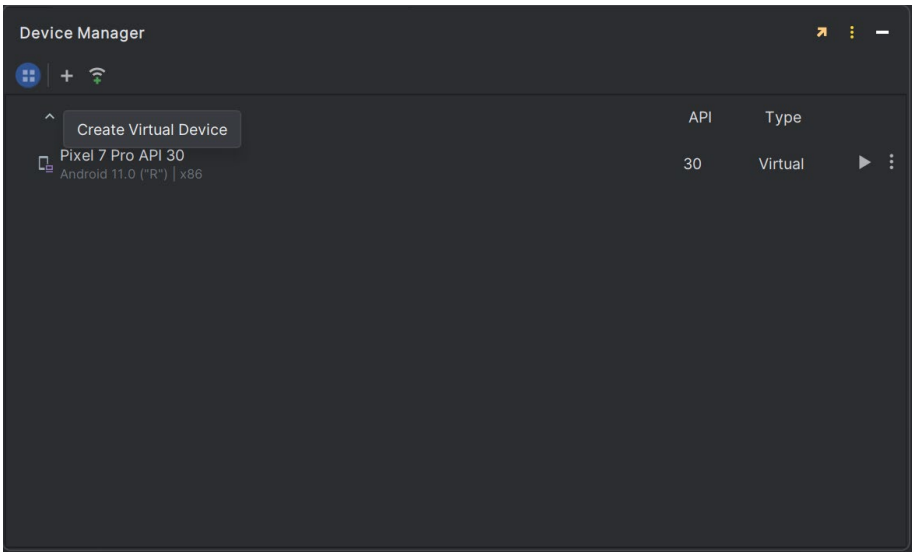


Ilustración 31: Android Studio Device Manager

Desde la ventana del gestor de dispositivos virtuales se creará un nuevo dispositivo.

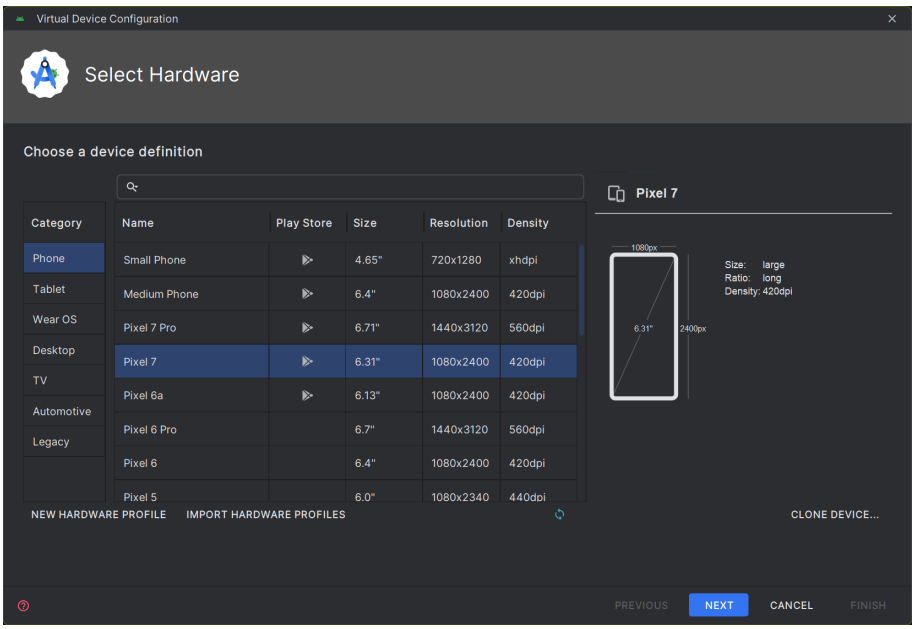


Ilustración 32: Virtual Device Configurator.

- Ejecutar la aplicación en el dispositivo virtual.

Una vez creado el dispositivo virtual se podrá ejecutar la aplicación es el mismo mediante el botón  .

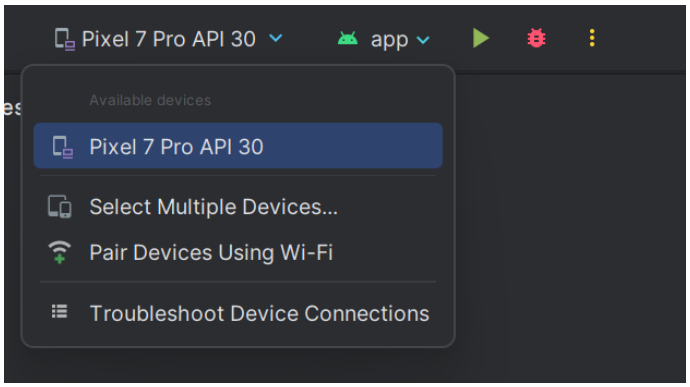


Ilustración 33: Run app menú.

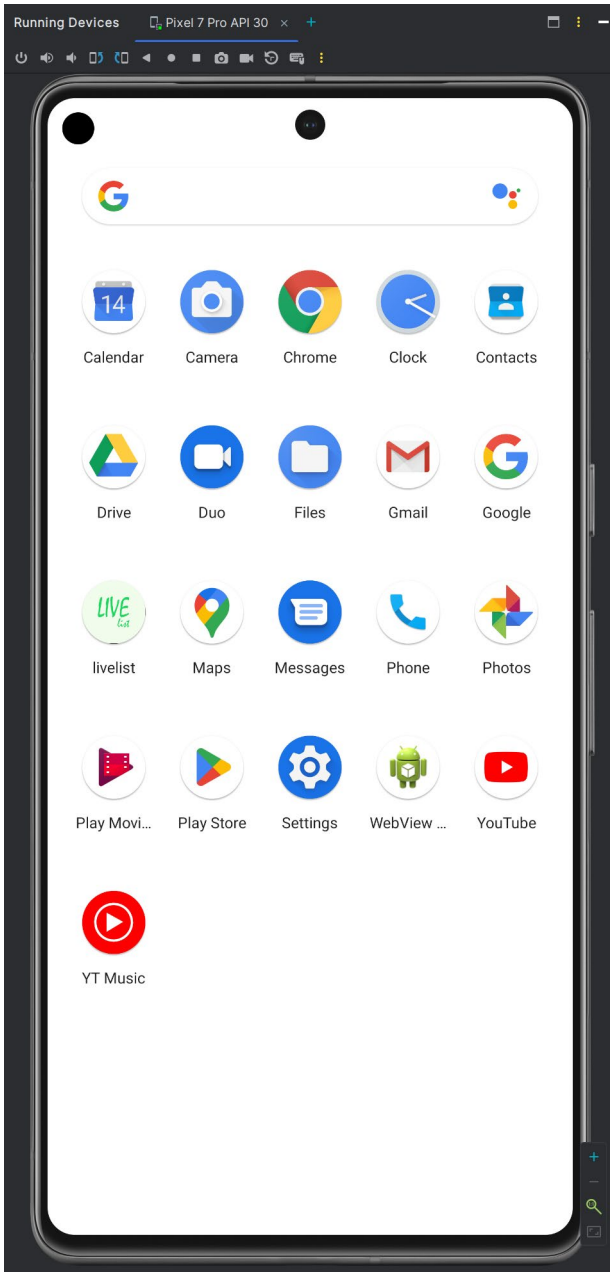


Ilustración 34: Dispositivo virtual.

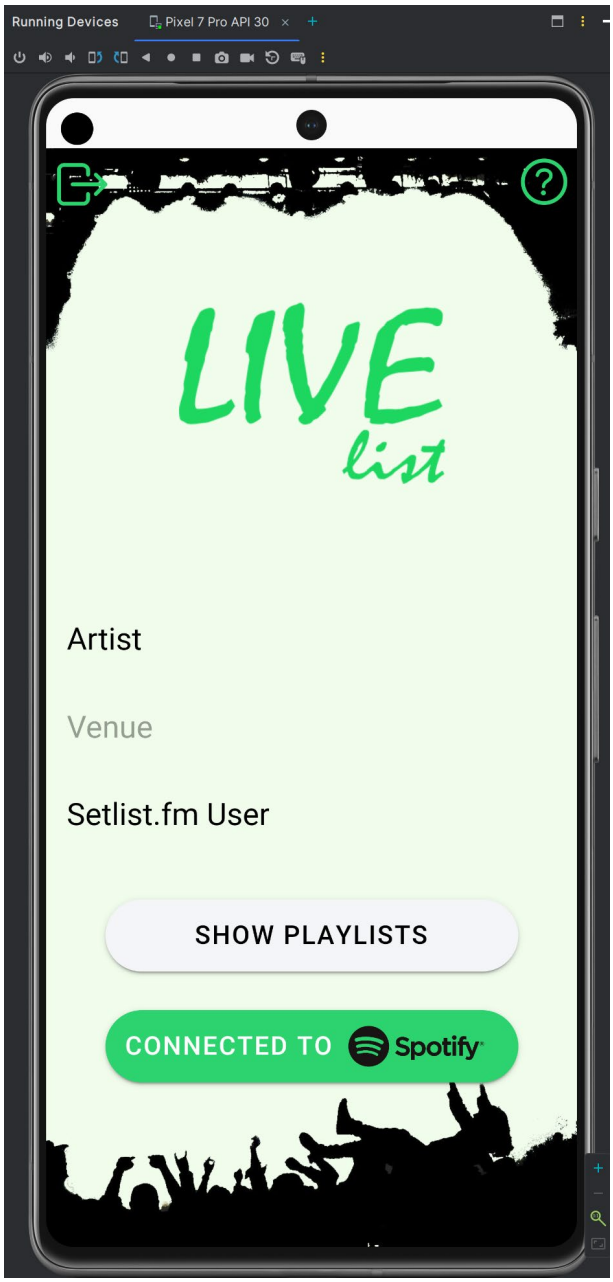


Ilustración 35: Aplicación en ejecución

- Ejecutar la aplicación en un dispositivo físico.

Para ejecutar en el dispositivo físico real se seguirán los siguientes pasos:

1. Se conectará el dispositivo a la máquina de desarrollo mediante un cable USB.
2. En el dispositivo físico se completarán los siguientes pasos a fin de habilitar la depuración por USB en el menú *Opciones para desarrolladores*:
    a. Se abrirá la app de *Configuración*.
    b. Se seleccionará el menú *Sistema*.
    c. Se seleccionará *Acerca del teléfono*.
    d. Se pulsará sobre *Número de compilación* siete veces.
    e. Se regresará a la pantalla anterior, y se presionará *Opciones para desarrolladores*.
    f. En la ventana *Opciones para desarrolladores*, se habilitará la *Depuración por USB*.
3. Para ejecutar la aplicación en Android Studio, igual que el apartado anterior, se seleccionará el dispositivo físico correspondiente y se hará clic en el botón Ejecutar .

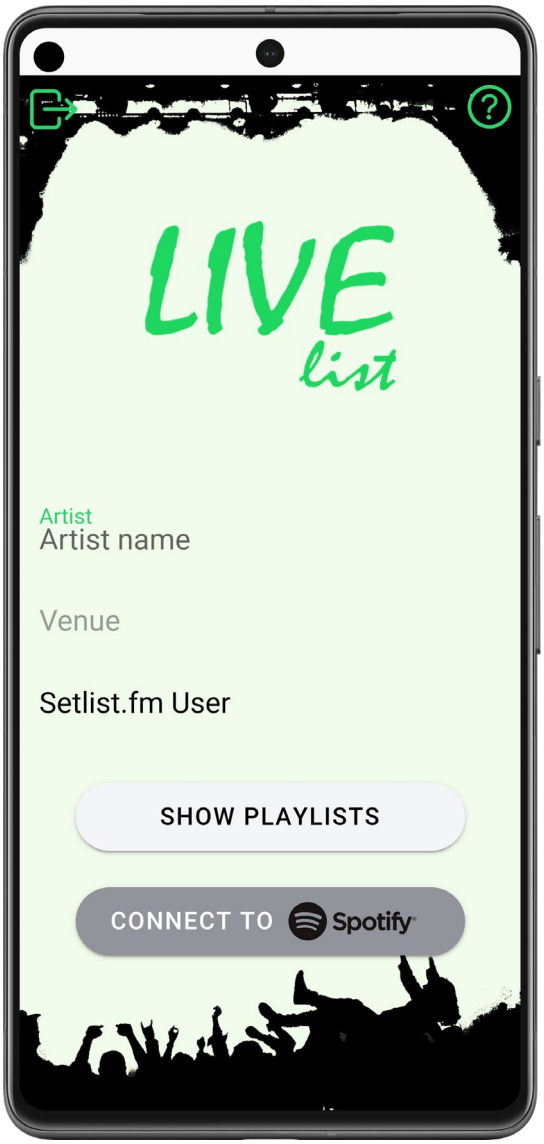
7.4. Anexo 4: Manual de uso de la aplicación

Livelist:

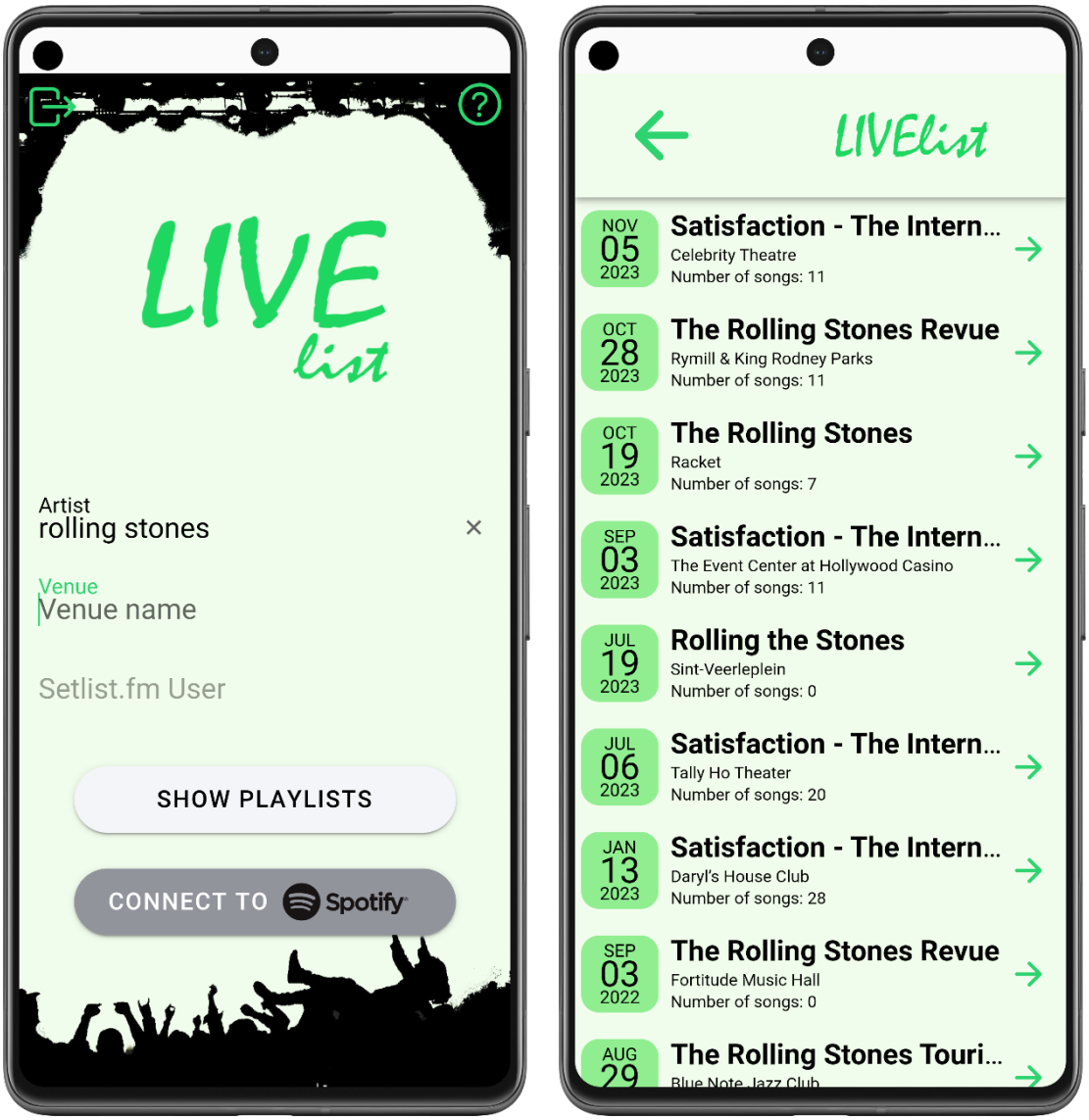
Livelist es una aplicación que permite a los usuarios ver listas de conciertos desde la *API* de *Setlist.fm* y crear listas de reproducción directamente a través de los servicios de *streaming* de música de *Spotify*.
A continuación, se detallan los pasos para utilizar la aplicación:

1- Inicio de la Aplicación:

Al iniciar la aplicación, se mostrará directamente la pantalla principal de la aplicación.



2- Pantalla Principal y de búsqueda:

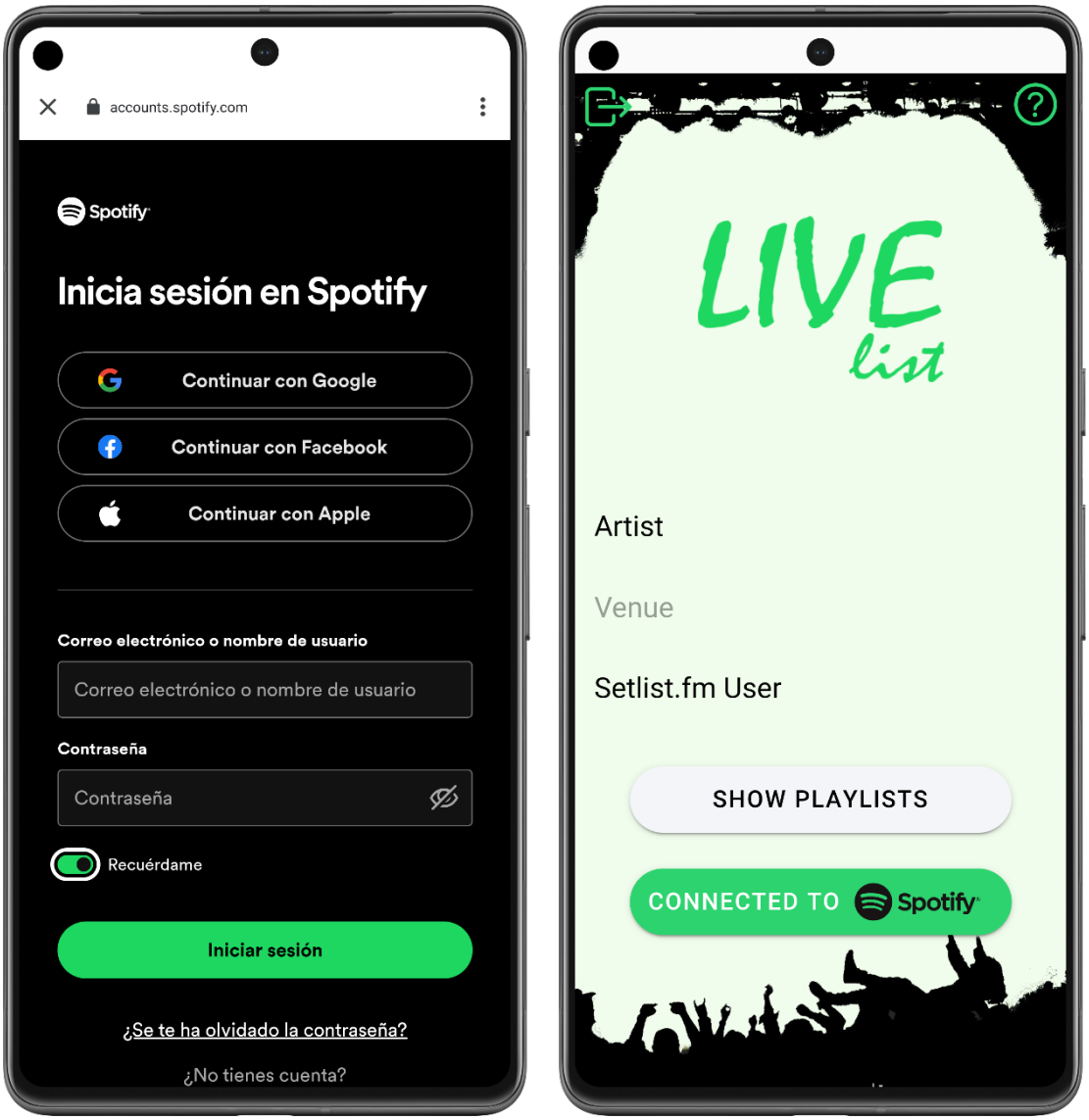


En la pantalla principal, se puede buscar un *setlist* por artista y/o recinto, o por nombre de usuario de Setlist.fm.

Para buscar un *setlist*, se debe introducir el nombre del artista y/o recinto, o el nombre de usuario de *Setlist.fm* en el campo de búsqueda correspondiente y hacer clic en el botón "Show Playlists".

Si existe algún *setlist*, se mostrará una lista como resultado, y en caso de no haber ninguno, se mostrará una alerta y se volverá a la pantalla principal.

3- Conexión a Spotify:



Para poder agregar canciones a una lista de reproducción de *Spotify*, se debe conectar la aplicación a una cuenta de *Spotify*.

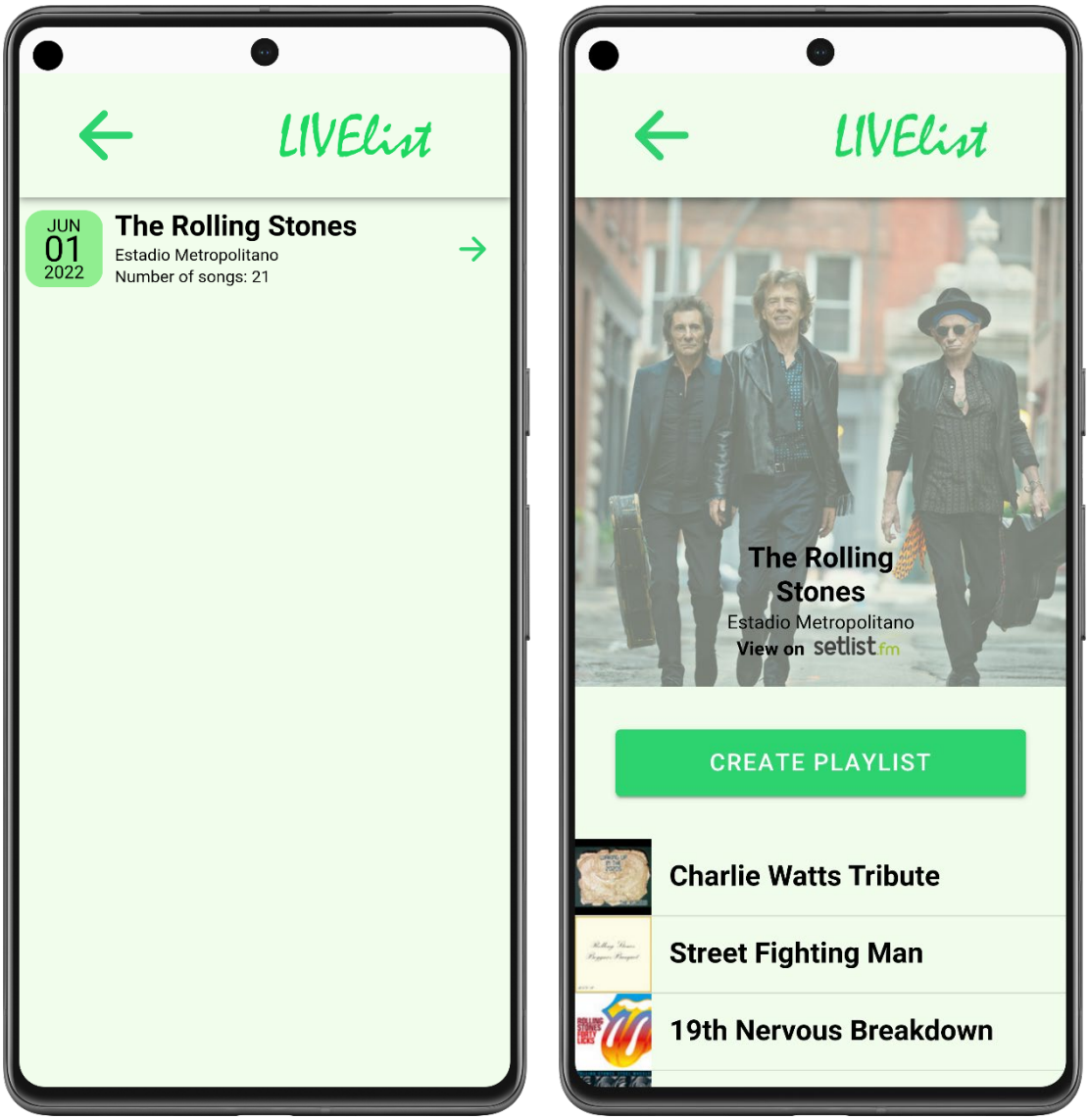
Para hacer esto, se debe hacer clic en el botón "Connect to Spotify" en la pantalla principal de la aplicación.

Se abrirá una ventana emergente para iniciar sesión en la cuenta de Spotify y permitir que la aplicación acceda a la cuenta.

Una vez conectado, se mostrará el botón de color verde con la leyenda “Connected to *Spotify*” en la pantalla principal.

Si se quiere desconectar la cuenta de Spotify habrá que pinchar en el icono de salir situado en la esquina superior izquierda.

4- Visualización de *Setlists*:



Al hacer clic en un *setlist* en la lista de resultados, se mostrará una pantalla con los detalles del *setlist*, incluyendo el nombre del artista, la fecha y el lugar del concierto, y la lista de canciones interpretadas.

Si se pincha en “View on Setlist.fm” se abrirá la página del *setlist* en el navegador predeterminado.

En esta pantalla, se puede hacer clic en el botón "Create Playlist" para agregar las canciones del *setlist* a una lista de reproducción de Spotify. Aparecerá en pantalla un mensaje de confirmación.