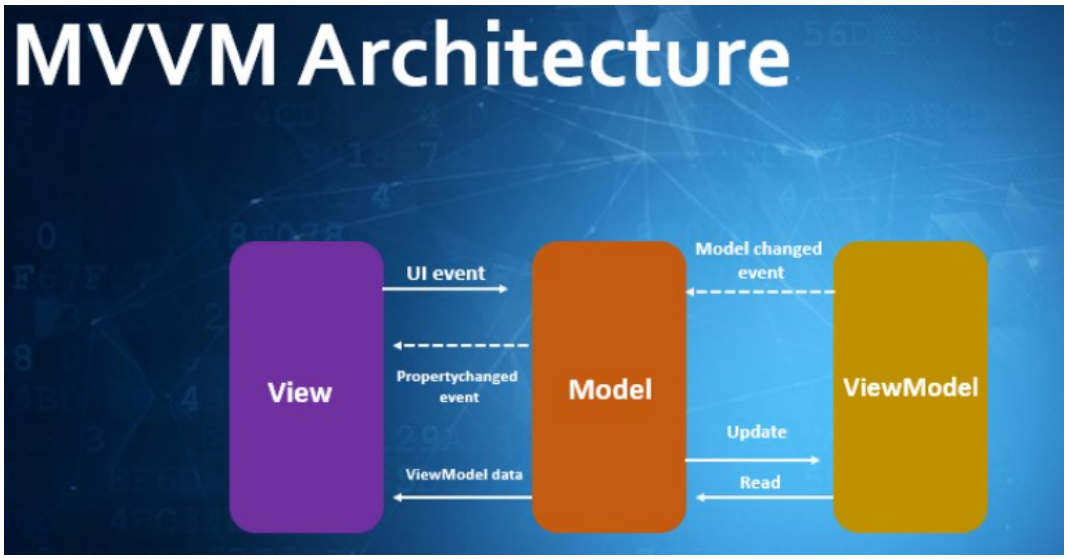
ULPGC · UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA

Práctica 3

Diseño arquitectónico



Programación de Aplicaciones Móviles Nativas
08 de octubre de 2023

Autores:

Ana del Carmen Santana Ojeda (ana.santana152@alu.ulpgc.es)

Alejandro David Arzola Saavedra (alejandro.arzola101@alu.ulpgc.es)

Índice

1. Introducción

Este ejercicio práctico tiene como objetivo principal explorar el diseño básico de una aplicación móvil mediante la **aplicación de un patrón de diseño estructural**, en nuestro caso hemos elegido la arquitectura MVVM.

2. Desarrollo

2.1. Dominio de TUNEWAVE

Tunewave es una aplicación de música diseñada para ofrecer a los usuarios una **experiencia musical única**. Con un enfoque en la música más reciente, permite a los usuarios **descubrir y disfrutar de las últimas canciones de sus artistas favoritos**.

Las **principales funcionalidades** de nuestra aplicación son las siguientes:

- Los usuarios pueden **escuchar todas las canciones** que se publiquen en la aplicación.
- Los usuarios tienen la posibilidad de **dar 'me gusta'** a canciones específicas para organizarlas en una **lista personalizada**, permitiéndoles así crear una playlist de sus 'me gusta' **para escuchar fácilmente en cualquier momento**.
- La función de **añadir a favoritos** proporciona una manera rápida y sencilla de **guardar canciones para futuras escuchas**.
- La aplicación también ofrece a los usuarios la opción de utilizar una **función de reconocimiento de canciones**. Si encuentran una canción desconocida, simplemente **pueden activar el reconocedor de canciones, que escuchará la melodía y tratará de identificarla**, brindando así a los usuarios la oportunidad de **descubrir nueva música de manera intuitiva**.

2.2. Identificación de los principales elementos de la arquitectura MVVM

Modelo

- **Canción (Song)**: Contiene información sobre una canción, como el título, el artista, el álbum, la duración.
- **Usuario (User)**: Puede contener información sobre el usuario, como su ID, nombre de usuario y una lista de canciones marcadas como favoritas o me gustas.
- **Reconocimiento de Canción (Song Recognition)**: Un componente que almacena información relacionada con el reconocimiento de canciones, como el nombre de la canción, el artista y otros detalles.

Vista

- **Music Player**: La interfaz de usuario que permite al usuario reproducir, pausar y navegar entre canciones.
- **Song List**: Muestra la lista de canciones disponibles para que el usuario las reproduzca. Incluye opciones para dar 'me gusta' y marcar como favorita.
- **Favorites List**: Muestra las canciones marcadas como favoritas por el usuario.
- **Song Recognition UI**: La interfaz de usuario que muestra información sobre la canción que está siendo reconocida en tiempo real.

ViewModel

- **MusicPlayerViewModel**: Contiene la lógica para controlar la reproducción de canciones y mantener el estado del reproductor.

- **SongListViewModel**: Gestiona la lógica para mostrar la lista de canciones y manejar las interacciones del usuario, como dar 'me gusta' y marcar como favorita.
- **FavoritesViewModel**: Contiene la lógica para mostrar la lista de canciones favoritas y manejar las interacciones relacionadas.
- **SongRecognitionViewModel**: Contiene la lógica para gestionar el reconocimiento de canciones y actualiza la interfaz de usuario correspondiente. También puede interactuar con otros ViewModels para agregar la canción reconocida a la lista de canciones disponibles.

Servicios

- **Playback Service**: Controla la reproducción de las canciones y gestiona eventos como reproducción, pausa y cambio de canción.
- **User Service**: Gestiona la información del usuario, incluidas las listas de favoritos y las canciones marcadas como 'me gusta'.
- **Song Recognition Service**: Utilizado para procesar la entrada de audio del usuario y devolver información sobre la canción reconocida.

Comandos

- **Like**: Un comando asociado a la acción de dar 'me gusta' a una canción.
- **Mark as Favorite**: Un comando para marcar una canción como favorita.
- **Start Recognition**: Un comando asociado a la acción de comenzar el reconocimiento de una canción.

Data Binding

- Se utiliza para vincular automáticamente la información de las canciones en el modelo a los elementos de la interfaz de usuario en las listas y en el reproductor de música.
- **Observador de Canción Reconocida (Recognized Song Observer)**: Actualiza la interfaz de usuario con la información de la canción reconocida.

Observadores

- Se utilizan para observar cambios en la lista de favoritos y en las canciones marcadas como 'me gusta', actualizando automáticamente la interfaz de usuario.

Navegación

- Gestiona la transición entre la lista de canciones, la lista de favoritos y el reproductor de música.
- Puedes agregar transiciones entre las diferentes pantallas relacionadas con el reconocimiento de canciones.

Experiencia del Usuario

- Puedes considerar cómo integrar la función de reconocimiento de canciones de manera intuitiva en la interfaz de usuario, por ejemplo, mediante un botón de 'Escuchar y Reconocer' en el reproductor de música.

Manejo de Resultados

- Es importante tener en cuenta cómo gestionar los resultados del reconocimiento, como qué acciones tomar en caso de que la canción no pueda ser reconocida o cómo administrar múltiples resultados.

2.3. Diseño de la Arquitectura

En el siguiente apartado, presentaremos cómo planeamos orientar nuestro proyecto utilizando la arquitectura MVVM (Model-View-ViewModel).

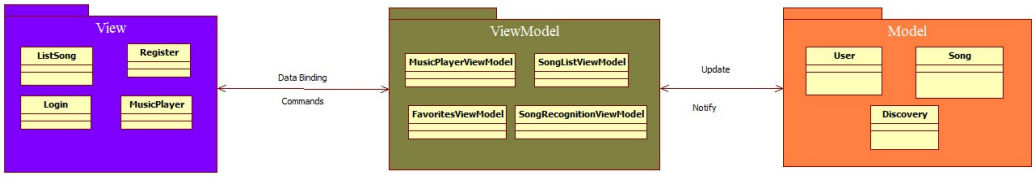


Imagen 1: Diagrama

2.4. Caso de Uso

2.4.1. Descripcion de los casos de uso principales

Para tratar el siguiente apartado, hemos seleccionado **dos historias de usuario** de bastante relevancia en nuestro proyecto. Ambas están dirigidas al desarrollador, y su impacto en el proyecto es considerable, debido a su **complejidad y alcance significativos**.

La primera historia de usuario que abordaremos se centra en la funcionalidad de **crear, modificar y eliminar listas por parte del usuario**. Como se observa en la **Imagen 2**, consideramos que es de suma importancia permitir al usuario agregar nuevas listas, además de las predeterminadas, como 'Favoritas' o 'Me gusta'. Hemos tomado la decisión de **priorizar el desarrollo de esta historia de usuario**, asignándole una prioridad de nivel 2 en la sección de 'Iteración asignada', con el objetivo de implementarla lo más pronto posible.

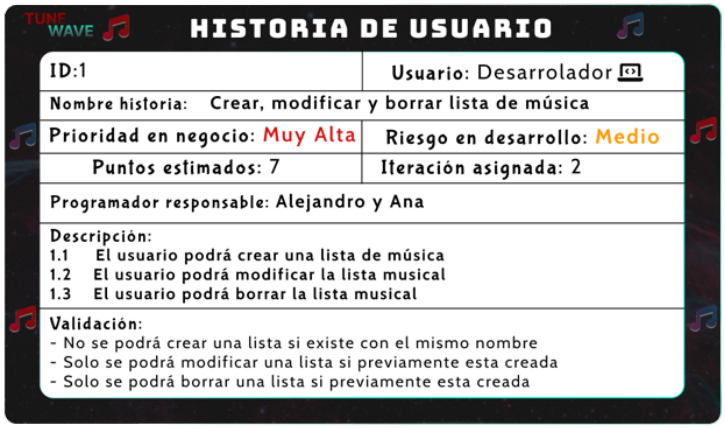


Imagen 2: Caso de uso

La última historia de usuario que abordaremos se enfoca en la funcionalidad de **reconocimiento de canciones**. Como se puede apreciar en la **Imagen 3**, consideramos de vital importancia brindar a los usuarios la capacidad de **descubrir canciones mediante la tecnología de reconocimiento de sonido**.

Hemos tomado la determinación de dar menos prioridad al desarrollo de esta historia de usuario, respecto a la anterior, asignándole una prioridad de nivel 4 en la sección de 'Iteración asignada', con el firme propósito de **implementarla tan pronto como sea posible**.

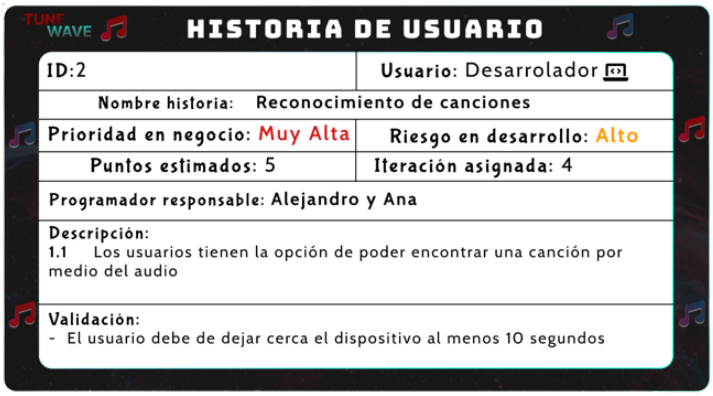


Imagen 3: Caso de uso

2.4.2. ¿Cómo se procesarían estos casos de uso a través del MVVM?

Los casos de uso los procesariamos de la siguiente manera:

CRUD de Lista de Música

- **Modelo (Model)**: Define la estructura de los datos, como la clase Song para representar las canciones. Puede haber modelos adicionales para representar listas de reproducción, usuarios, etc.
- **ViewModel (ViewModel)**: SongViewModel podría contener métodos para recuperar listas de canciones, agregar nuevas canciones, eliminar canciones, etc. Se comunica con el repositorio o un proveedor de datos para realizar operaciones CRUD.
- **Vista (View)**: La interfaz de usuario (Activity o Fragment) observa los datos en el SongViewModel a través de LiveData u otras técnicas de observación. Al realizar operaciones CRUD, la Vista interactúa con el ViewModel, que a su vez se comunica con el modelo y actualiza la interfaz de usuario mediante LiveData.

Reconocimiento de Canciones

- **Model**: Define la clase SongRecognition para representar los resultados del reconocimiento de canciones.
- **ViewModel**: RecognitionViewModel podría contener métodos para iniciar el reconocimiento de canciones y manejar los resultados obtenidos. Se comunica con el componente de reconocimiento de canciones.
- **View**: La interfaz de usuario tiene un componente (botón, por ejemplo) que inicia el reconocimiento de canciones. Cuando se completa el reconocimiento, el ViewModel actualiza la interfaz de usuario con los resultados mediante LiveData.
- **Interfaz de Conexión a Reconocedor de Voz**: Implementa VoiceRecognitionListener para manejar los resultados del reconocimiento de voz y cualquier error.

3. Conclusión

Hemos elegido la arquitectura MVVM para el diseño de nuestra aplicación móvil de música, ya que la consideramos óptima. Esta elección se basa en la eficiencia que ofrece la arquitectura MVVM en términos de organización y separación de responsabilidades.

Optamos por evitar complicaciones innecesarias asociadas con otras arquitecturas más complejas, ya que nuestro objetivo principal es agilizar el lanzamiento del producto. La simplicidad y claridad de MVVM nos permiten concentrarnos en el desarrollo efectivo de la aplicación sin comprometer la calidad del código ni la experiencia del usuario.

MVVM
Ventajas:
- Desacoplamiento entre la lógica de presentación (ViewModel) y la View.
- Fácil manejo del ciclo de vida con ViewModels.
- Enfoque en la reactividad y observabilidad.

Desventajas:
- Mayor cantidad de clases y abstracciones puede parecer complejo.

MVP
Ventajas:
- Enfoque claro en la separación de responsabilidades.
- Presenters pueden facilitar las pruebas unitarias.

Desventajas:
- A veces, la comunicación entre el Presenter y la View puede ser compleja.
- La actualización de la interfaz de usuario puede ser más manual.

MVVM:
- Se enfoca en la separación de responsabilidades y la observabilidad.
- Ideal para proyectos medianos a grandes con una interfaz de usuario dinámica.
- Integración nativa con Android Jetpack.

MVI:
- Proporciona un flujo unidireccional de datos y un manejo claro del estado. - Puede ser más adecuado para proyectos donde la gestión del estado es crucial.

Clean Architecture:
- Enfoca la separación clara de capas y reglas de negocio en el núcleo.
- Escalable y adaptable a cambios, pero puede ser excesivo para proyectos más pequeños.