Documentación

En este documento se mostrarán y explicarán las librerías usadas en la prueba técnica.

RetroFit

Esta es una librería para realizar llamadas a una API.

Los motivos por los cuales he decidido utilizar esta librería son los siguientes:

- **Ya la conocía:** Este es uno de los principales motivos ya que en mi día a día trabajo con ella.
- **Simplicidad y legibilidad del código:** Retrofit proporciona una capa de abstracción sobre el protocolo HTTP que hace que sea mucho más fácil realizar solicitudes de red. En lugar de tener que manejar conexiones de red de bajo nivel, puedes concentrarte en definir la lógica de tu aplicación.
- **Alto rendimiento y eficiencia:** Retrofit utiliza OkHttp en el fondo, que es una biblioteca de red eficiente que maneja automáticamente la conexión, la persistencia y la recuperación de conexiones, así como el almacenamiento en caché de respuestas.
- **Ahorro de tiempo y recursos:** Al encargarse de todas las tareas repetitivas y técnicas de manejo de API, Retrofit te permite centrarte en la lógica de la aplicación, lo que ahorra tiempo de desarrollo y, a largo plazo, recursos.

Dagger – Hilt

Esta es una librería para la inyección de dependencias.

- **Configuración simplificada:** Hilt se encarga de la configuración del gráfico de inyección de dependencias que normalmente tendrías que hacer manualmente con Dagger. Esto reduce la posibilidad de errores y hace que el código sea más limpio y fácil de leer.
- **Estandarización:** Hilt promueve un enfoque estandarizado para la inyección de dependencias en Android, lo que puede facilitar el trabajo en equipo y hacer que tu código sea más fácil de entender para otros desarrolladores familiarizados con Hilt.
- **Rendimiento de Dagger:** Al ser una extensión de Dagger, Hilt se beneficia del rendimiento y la eficiencia de Dagger. Dagger realiza mucha de su trabajo en tiempo de compilación, lo que significa que no ralentizará tu aplicación en tiempo de ejecución.
- **Soporte oficial de Google**: Hilt es la biblioteca recomendada por Google para la inyección de dependencias en Android, lo que significa que tendrás un soporte actualizado y documentación oficial para respaldar tu implementación.

Lifecycle

Esta librería se usa para manejar los ciclos de vida de la aplicación.

Los motivos por los cuales he decidido utilizar esta librería son los siguientes:

- **Manejo simplificado del ciclo de vida**: Las bibliotecas de Lifecycle simplifican el manejo del ciclo de vida de los componentes, proporcionando clases y métodos que te permiten reaccionar a los cambios de estado sin tener que recordar y manejar todos los detalles por tu cuenta. Esto puede ayudar a evitar errores y hacer que tu código sea más fácil de leer y mantener.
- **Prevención de fugas de memoria**: Si no se manejan correctamente, los cambios en el ciclo de vida pueden causar fugas de memoria, lo que puede ralentizar o incluso bloquear tu aplicación. Las bibliotecas de Lifecycle proporcionan patrones y utilidades para ayudarte a evitar estas fugas.
- **Mejor integración con otras bibliotecas de Android Jetpack:** Muchas otras bibliotecas de Android Jetpack, como LiveData y ViewModel, están diseñadas para funcionar bien con las bibliotecas de Lifecycle. Esto significa que si estás utilizando estas otras bibliotecas, puede ser beneficioso utilizar también las bibliotecas de Lifecycle. En este caso en particular al usar el patrón de diseño Model View ViewModel (MVVM), nos es muy útil esta librería.
- **Soporte para Kotlin Coroutines:** La biblioteca de Lifecycle proporciona soporte para Kotlin Coroutines, permitiéndote lanzar coroutines que estén vinculadas al ciclo de vida de un componente, lo que puede simplificar el manejo de operaciones asíncronas.
- **Mantenimiento y soporte por parte de Google:** Al ser parte del conjunto de bibliotecas de Android Jetpack, las bibliotecas de Lifecycle son mantenidas y soportadas por Google, lo que significa que puedes esperar actualizaciones y mejoras regulares.

Mockk

Esta librería se usa para la realización de las pruebas.

El hecho de usar esta librería y no otra más popular en el mundo Java como puede ser ***Mokito*** es por los siguientes motivos:

- **Soporte de primer nivel para Kotlin:** Mockito fue diseñado originalmente para Java y, aunque funciona con Kotlin, no se aprovecha completamente de las características del lenguaje Kotlin. Por otro lado, ***MockK*** fue diseñado para Kotlin.
- **Sintaxis más limpia y concisa:** La sintaxis de MockK a menudo puede ser más limpia y más concisa que la de Mockito.
- **Soporte para corutinas de Kotlin:** MockK tiene un buen soporte para las corutinas de Kotlin, que son una característica importante del lenguaje para realizar tareas asíncronas. En nuestra aplicación encontramos varias tareas asíncronas a la hora de realizar las llamadas a la API

Kotlinx-coroutines

Es probablemente la biblioteca kotlinx más famosa y utilizada. Proporciona soporte para coroutines en Kotlin, que son una forma de realizar tareas

Esta librerías la usaremos para las llamadas asíncronas realizadas en la app