Tema 3.4 - ViewModel

Descargar estos apuntes pdf o html

Índice

▼ Introducción
- Alcance de un ViewModel (ViewModel Scope)
- ✋ Prácticas no recomendadas

Introducción

En anteriores temas cuando hablamos de la **arquitectura de aplicaciones de Android** ya mencionamos este componente como parte de la Capa de UI. En este tema vamos a centrarnos en él para concretar la implementación de MVVM en nuestra capa UI de la arquitectura.

`ViewModel` es una de esas clases que **Google** definió, allá por 2018, en la primera versión de **Jetpack** para ayudar a los desarrolladores a crear aplicaciones de Android más robustas y fáciles de mantener. `ViewModel` es una clase que está **diseñada para almacenar y administrar datos relacionados con la interfaz de usuario de una manera que sobrevive a los cambios de configuración**, como la rotación de la pantalla.

Los beneficios clave de la clase ViewModel son básicamente dos:

- Te permite conservar el estado de la IU.
- Proporciona acceso a la lógica empresarial, idealmente a través de casos de uso definidos en el dominio.

Alcance de un ViewModel (ViewModel Scope)

Cuando se crea una instancia de `ViewModel`, se pasa un objeto que implementa la interfaz `ViewModelStoreOwner`.

Los objetos que implementan `ViewModelStoreOwner` puede ser por ejemplo:

- Una `Activity` o `ComponentActivity`.
- Un `Fragment`.
- Un destino de Navigation `NavBackStackEntry`.

> ✋ **Importante**: El alcance de tu ViewModel se define en el Ciclo de vida del `ViewModelStoreOwner`. Esto es, Continúa en la memoria hasta que su `ViewModelStoreOwner` desaparece de forma permanente.

Cuando se destruye el fragmento o la actividad para los que se definió el alcance del ViewModel, el trabajo asíncrono continúa en el ViewModel específico. Esta es la clave de la persistencia.

> 📌 **Nota**: Cuando se definió la clase `ViewModel` en la primera versión de **Jetpack**, aún no existia **Compose** y se usaba asociado a una vista como un `Activity` o a un `Fragment`. Nuestras aplicaciones de Android se componían de una o más `Activity` o `Fragment` y cada uno de ellos tenía su propio `ViewModel` que se creaba como una **instancia única**. `ViewModel`

> se usaba para almacenar datos que se necesitaban en la interfaz de usuario de la `Activity/Fragment` o el `Fragment` y queríamos que sobrevivieran a los cambios de configuración o queríamos compartir datos entre ellos.
>
> Con la llegada de **Jetpack Compose** este enfoque ha cambiado y Google ahora **recomienda aplicaciones de actividad única** donde las diferentes pantallas se cargan como contenido dentro de la misma actividad. Por tanto, un ViewModel utilizado por una actividad permanece en memoria hasta que la actividad finalice esto es hasta que la aplicación finalice.

✋ Prácticas no recomendadas

Las siguientes son varias prácticas recomendadas clave que debes seguir cuando implementes `ViewModel`:

- ❌ **NO** definas `ViewModel` para *composables* reutilizables en tu UI o para componentes de IU que no sean de nivel superior. Deberíamos definirlos a nivel de Screen (pantalla).
- ❌ Los ViewModels **NO deberían conocer los detalles de implementación de la IU**. Mantén los nombres de los métodos que expone la API de ViewModel y los de los campos del UIState lo más genéricos posible.
- ❌ Como pueden tener una vida más larga que el `ViewModelStoreOwner`, los ViewModels **NO deberían contener ninguna referencia de APIs relacionadas con el ciclo de vida**, como `Context` o `Resources` para evitar fugas de memoria.
- ❌ **NO pases ViewModels a funciones ni otros componentes de la IU**. Esto evita que los componentes de nivel inferior accedan a más datos y lógica de los que necesitan.
- ❌ Derivado del anterior **NO instancies directamente un objeto ViewModels**. En su lugar, usa algún tipo de *'proveedor de ViewModels'* para obtener una instancia del mismo.