Tema 2.2 - Arquitectura de una aplicación

Descargar estos apuntes pdf o html

Índice

Introducción

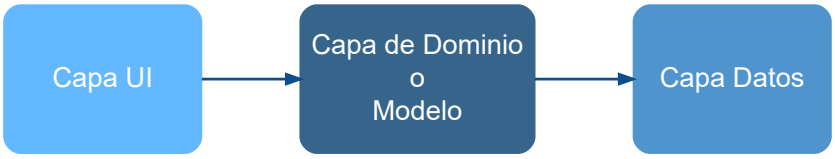
Básicamente una aquitectura software es la estructura de un sistema de software, los componentes del software, las propiedades externamente visibles de esos componentes y las relaciones entre ellos.

La idea tras las arquitecturas es tener capas que separen las responsabilidades de cada una de ellas. De esta forma se consigue que el código sea más mantenible, escalable y testeable. Además, podremos cambiar una capa sin que afecte a las demás.

Actualmente existen arquitecturas complejas de sistemas cómo por ejemplo Arquitectura Hexagonal, Microservicios, Event-driven Architecture, etc. Sin embargo, en el mundo del diseño diseño de aplicaciones para dispositivos móviles tenemos patrones arquitectónicos más sencillas cómo MVC, MVP, MVVM, etc.

> 📌 **Nota**: Puedes ver más información sobre la arquitectura propuesta por Google para Android en Guía de arquitectura de apps

Esquema básico de una aplicación Android propuesto por Google



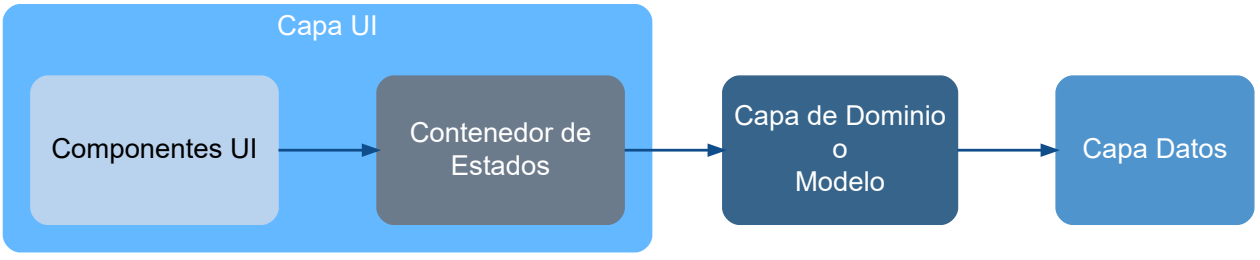
Capa UI

La función de la capa de la IU (o capa de presentación) consiste en mostrar los datos de la aplicación en la pantalla. Cuando los datos cambian, ya sea debido a la interacción del usuario (como cuando presiona un botón) o una entrada externa (como una respuesta de red), la IU debe actualizarse para reflejar los cambios.

La capa de la IU consta de los siguientes dos elementos:

- Elementos de la IU que renderizan los datos en la pantalla (puedes compilar estos elementos mediante las vistas XML o las funciones de **Jetpack Compose**)
- Contenedores de estados (como las clases **ViewModel**) que contienen datos, los exponen a la IU y controlan la lógica

> ✋ **Importante**: Documentación oficial de Android sobre la **Capa de UI** en la que más adelante profundizaremos

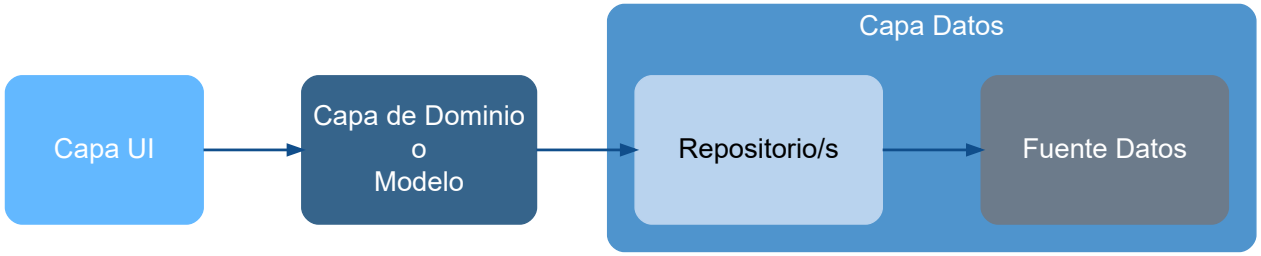


Capa de Datos

La capa de datos está formada por repositorios que pueden contener de **cero a muchas fuentes de datos**. A este tipo de diseño se le denomina **Repository Pattern** y es un patrón de diseño que se utiliza para separar la lógica empresarial de la lógica de acceso a los datos. El repositorio se encarga de obtener los datos de una fuente de datos y convertirlos en un formato con el que el resto de la aplicación pueda trabajar.

> 📌 **Nota**: Este patrón fué introducido **Martin Fowler** en su libro Patterns of Enterprise Application Architecture en el año **2002** y posteriormente desarrollado por **Eric Evans** en su libro Domain-Driven Design en el año **2003**. Hoy en día es un patrón muy utilizado en el desarrollo de software y ámpliamente utilizado en el desarrollo de aplicaciones móviles.

> ✋ **Importante**: Documentación oficial de Android sobre la **Capa de datos** en la que más adelante profundizaremos



Capa de Dominio

La capa de dominio es una **capa opcional** que se ubica entre la capa de la IU y la de datos.

La capa de dominio es responsable de encapsular la lógica empresarial compleja o la lógica empresarial simple que varios ViewModels reutilizan. Esta capa es opcional porque no todas las apps tendrán estos requisitos. Solo debes usarla cuando sea necesario; por ejemplo, para administrar la complejidad o favorecer la reutilización.

Esta terminología viene del **DDD (Domain Driven Design)** que es una metodología de diseño de software que se centra en la lógica empresarial y en la comunicación entre los expertos en el dominio y los desarrolladores.

Sin embargo, nosotros vamos a obviar esta capa y vamos a implementar una arquitectura más sencilla basada en el patrón **MVVM** en el cual esta capa se denomina **Modelo** ya que representará el modelo de datos.

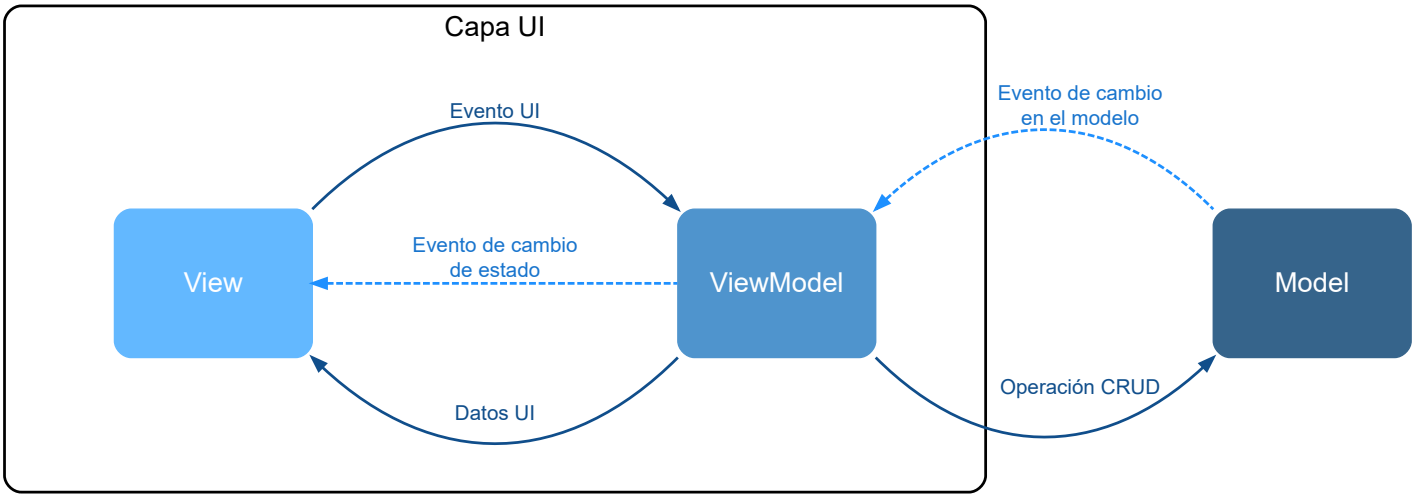
Modelo vista vista modelo MVVM

MVVM es un patrón de diseño de software que facilita la separación de la lógica de presentación de la lógica de negocio.

MVVM es una variante del patrón MVC (Modelo-Vista-Controlador) que se utiliza para la construcción de interfaces de usuario. En MVVM, el ViewModel asume la funcionalidad del "intermediario". En MVVM, toda la lógica de presentación se coloca en el ViewModel.

El ViewModel es responsable de recuperar los datos de los modelos y exponerlos a la vista. El ViewModel también se utiliza para manejar todas las interacciones del usuario. El ViewModel recupera los datos de los modelos, los formatea y los expone a la vista. El ViewModel también acepta las entradas del usuario y las valida antes de actualizar los modelos.

Implementaciones o aproximaciones a esta arquitectura hay muy diversas, pero básicamente la que nosotros vamos a implementar un esquema similar al siguiente para aproximarnos a MVVM siguiendo las recomendaciones de Google.



Donde ...

- **View** son los **Componentes UI** en nuestro caso los componentes de **Jetpack Compose** que renderizan los datos en la pantalla.
- **ViewModel** es el **Contenedor de estados** que contiene datos, los exponen a la IU y controlan la lógica.
- **Model** es una abstracción de los datos de la aplicación. En nuestro caso serán los **Repositorios** que se encargan de obtener los datos de una fuente de datos y convertirlos en un formato con el que el resto de la aplicación pueda trabajar.

Visión general de la arquitectura en un proyecto de Android

A la hora de concretar la arquitectura de una aplicación de Android, aunque hay una serie de recomendaciones por parte de Google, no hay una forma única de hacerlo. Por tanto, el equipo de desarrollo de una aplicación debe decidir algún tipo de **convenio organizativo** de paquetes, nombres de clases, etc. que se adapte a sus necesidades. De tal manera que cualquier miembro del equipo sepa encontrar fácilmente donde se encuentra la implementación de una determinada funcionalidad. Además de modificar fácilmente con el menor número de fallos y conflictos de trabajo en paralelo.

Esto último, se consigue creando diferentes módulos de la aplicación que se puedan desarrollar de forma independiente y que se puedan integrar fácilmente en el proyecto final. Sin embargo, la parte de modularización en diferentes paquetes no la vamos a abordar en este curso por falta de tiempo.

Descarga del siguiente enlace tienes un **'Cheat Sheet' de la Arquitectura** propuesta en el curso que debes descargar y tener accesible durante el presente curso.

> 📌 **Nota**: A medida que vayamos entrando en las diferentes capas de desarrollo iremos concretando más sobre su implementación, las diferentes sub-capas de la arquitectura, los convenios de nombres que hemos usado en el curso, etc.