
UT-02

— Arquitectura de una App —

Objetivos didácticos

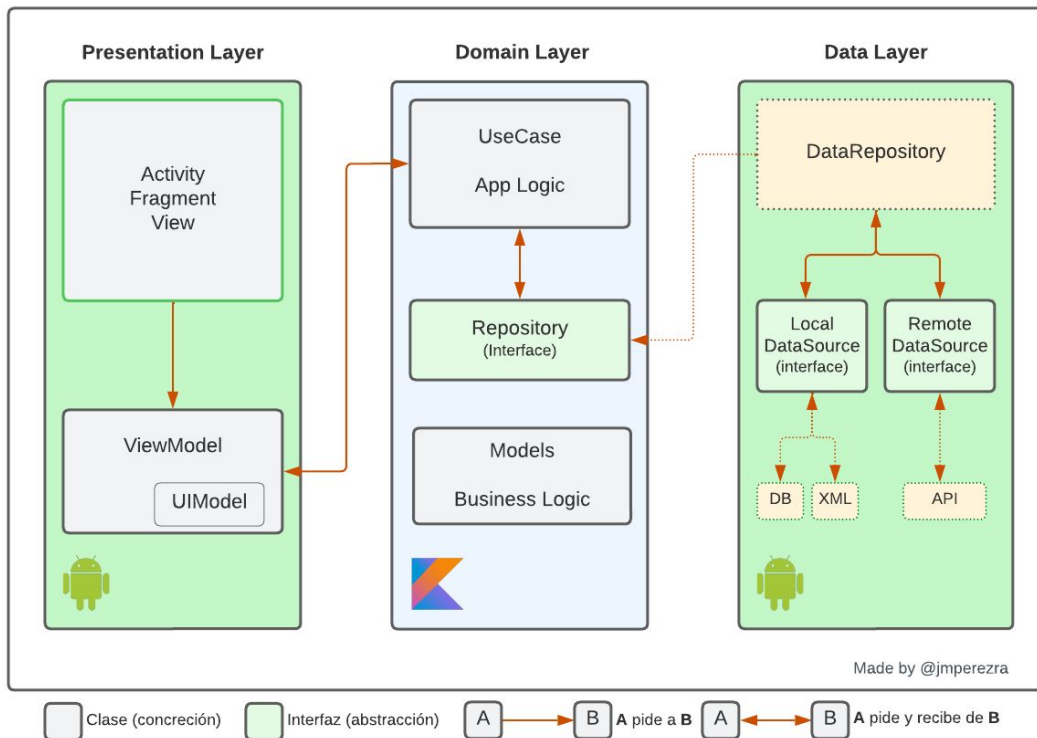
- Crear y trabajar con Activity.
- Crear una vista tradicional (xml) con componentes básicos.
- Definir la arquitectura de una aplicación Android: presentation, domain y data.
- Crear un ViewModel como controlador del patrón MVVM.
- Crear un caso de uso.
- Crear un repositorio de datos.
- Crear una fuente de datos local.
- Crear el flujo completo de una pantalla

Arquitectura

- Caos en el desarrollo de las primera generación de aplicaciones móviles.
- La comunidad Android se inclina por el modelo de arquitectura definido por Robert C. Martin "Clean Architecture".
- Hace unos 4 años, el equipo Android de Google establece una guía de estilos de desarrollo de aplicaciones Android basada en la arquitectura que ya seguía la comunidad.
- En la actualidad, es muy complicado que un candidato progrese si no conoce y trabaja con esta arquitectura.
- Una arquitectura **NO** es una jerarquía de carpetas/paquetes, es una organización lógica del código que puede llevar asociada una organización de paquetes.
- Se relaciona con los principios **SOLID**.

Arquitectura

Android Architecture Flow



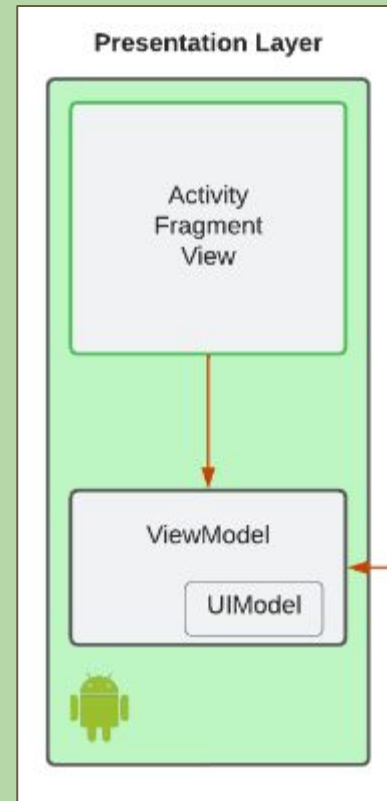
Arquitectura

Ejercicio 1

- Crear la rama **ut2-ej1** desde main.
- Crear la jerarquía de carpetas en nuestro proyecto.
- Actualiza el README.md con este ejercicio.
- Fusiona con el main.

Presentation

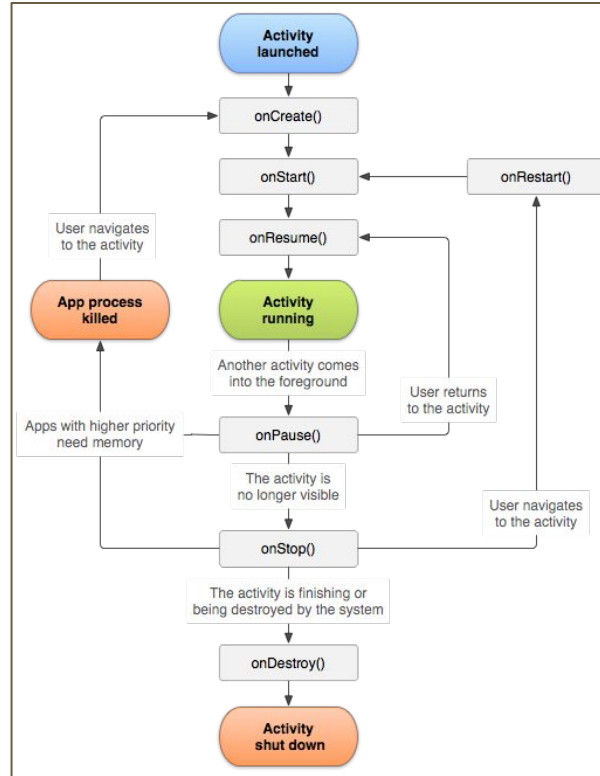
Capa de la UI



Activity

- Es una de las clases más importante de una aplicación.
- Es un punto de entrada a la aplicación.
- Lleva asociado ciclos de vida que tenemos que tener muy en cuenta a la hora de desarrollar.
- Se considera una vista y por tanto va en la capa de presentación.
- Se define en el fichero AndroidManifest para que pueda ser usada.
- Relacionado con el principio de **Inversión de Control**.
- Naming a seguir: Sustantivo + Activity, ejemplo, LoginActivity.

Activity



Vista XML

- Es la forma tradicional de desarrollar vistas (Interfaces de Usuarios) en aplicaciones Android.
- Google ya no actualiza esta librería ya que va a ser reemplaza por Jetpack Compose (Vistas declarativas).
- Las vistas XML son las más usadas en los proyectos y no están deprecadas, hay que seguir aprendiendo a desarrollar vistas en XML (proyectos antiguos).
- Estas vistas deben ir asociadas a una Actividad o Fragment.
- Naming a seguir: `activity_sustantivoAsociadaALaActividad`. Ejemplo: `activity_login`.

Vista XML

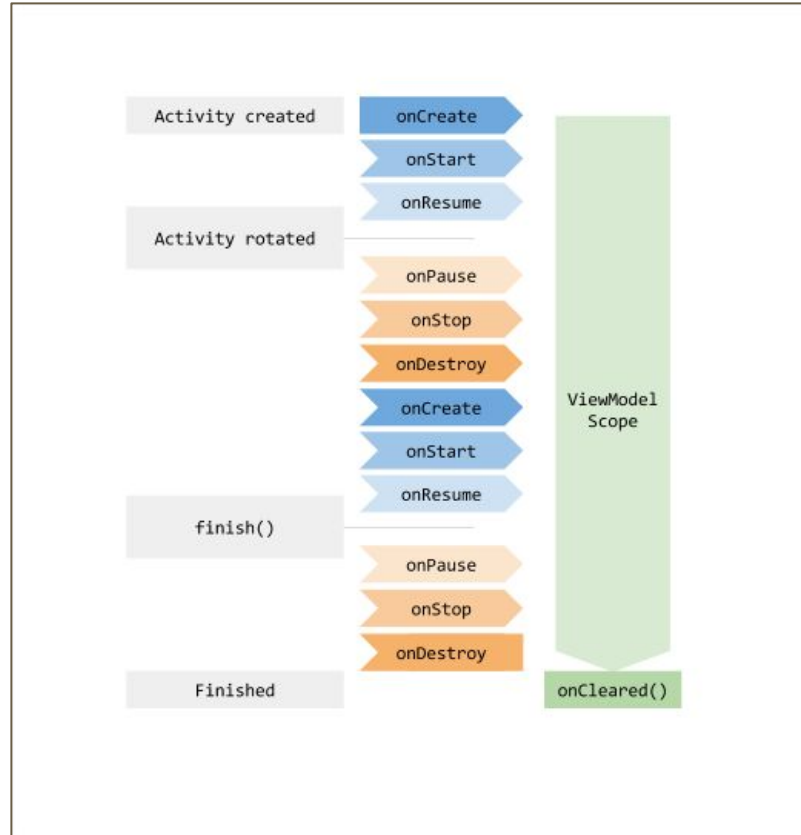
Ejercicio 2

- Crea una rama desde main con el nombre **ut2-ej2**.
- Crea una actividad llamada LoginActivity
- Crear una vista XML con el nombre activity_login.
- Añade un componente TextView con el texto: Identificación de Usuarios
- Establece esta actividad como la actividad principal de la aplicación, la que se muestra nada más abrir la app.
- Actualiza el README.md con este ejercicio.
- Fusiona con el main.

ViewModel

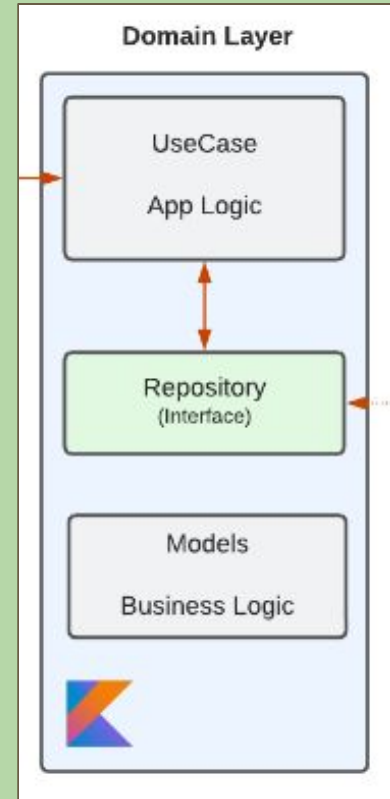
- Es una clase que pertenece al **SDK de Android**.
- Es un patrón de diseño relacionado con la capa de presentación (UI). Antes se usaba el patrón MVP (Model View Presenter).
- Surge como solución para sobrevivir al ciclo de vida de las vistas. El ViewModel es capaz de sobrevivir a la destrucción de la vista (rotación móvil).
- Se encarga de separar la lógica de presentación de la lógica de negocio. Recibe los eventos generados en la UI y ejecuta Casos de Usos.
- Se usa un modelo para representar los estados de la vista.
- Cuando el ViewModel se destruye se ejecuta el método ***onCleared***
- Naming: NombreActividadAsociada + ViewModel. Ejemplo: LoginViewModel

ViewModel



Domain

Capa de la Lógica de Negocio



UseCase

- Es una clase Kotlin, no tiene dependencia con Android.
- Sólo realiza una acción por eso sólo tendremos un método público.
Usaremos el operador invoke como método de ejecución del caso de uso:
 - `operator fun invoke()`
- Un caso de uso puede depender de otros casos de uso:
- Naming: Verbo + UseCase. Ejemplo: CreateUserUseCase

Modelos

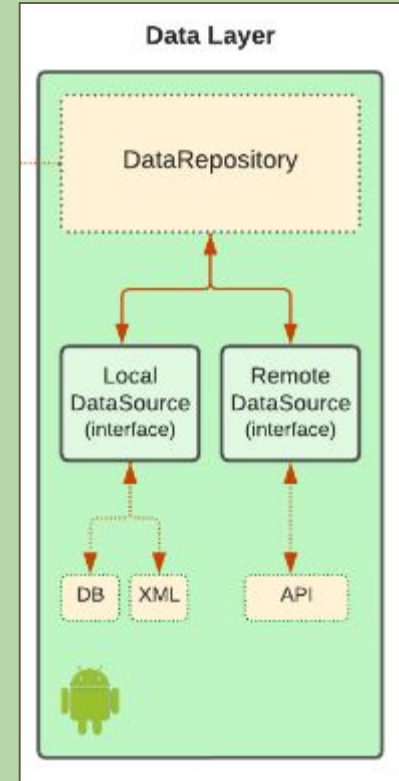
- Son clases que modelan las entidades de la lógica de negocio.
- En Android, suelen ser clases 'anémicas'. Sólo contienen estados (atributos).
- Un modelo puede depender de otros modelos.
- Un modelo no depende de otras clases, ejemplo: UseCase.
- Naming: Sustantivo. Ejemplo: BankAccount, User, Pet, etc.

Repository

- El repositorio es una abstracción (interfaz) que se usa para obtener datos (modelos con estados).
- Se suele tener un repositorio por modelo.
- El objetivo de usar un Repository es abstraernos de la tecnología que usamos para gestionar los datos: ficheros, bases de datos, API, etc.
- Naming: NombreModelo + Repository. Ejemplo: UserRepository.

Data

Capa con la Lógica de los datos



Concreción de Repository

- Clase que puede tener dependencias a Android.
- En esta capa se crea una concreción del Repository definido en el dominio.
- Con esto evitamos tener dependencias hacia Android en nuestro dominio.
- El objetivo del repositorio es definir las fuentes con las que cuenta la app (local, remota, etc) y establecer la lógica de cómo se obtienen esos datos.
- Naming: NombreRepoInterfaz + DataRepository.

Fuente de datos local

- Clase que puede tener dependencias a Android.
- Es la encargada de obtener los datos almacenados en local: fichero, base de datos, etc.
- Si se tienen varias fuentes de datos, habría que crear una interfaz donde se definan los métodos para obtener la información. Las concreciones serían cada una de las tecnologías usada: Ficheros, bases de datos, etc.
- Naming:
 - LocalDataSource (interfaz). Ejemplo: UserLocalDataSource.
 - Tecnología + LocalDataSource (concreción). Ejemplo: XmlLocalDataSource, DbLocalDataSource.

Fuente de datos remota

- Clase que puede tener dependencias a Android.
- Es la encargada de obtener los datos almacenados en remoto, ajeno a la aplicación.
- Si se tienen varias fuentes de datos, habría que crear una interfaz donde se definan los métodos para obtener la información. Las concreciones serían cada una de las tecnologías usada: Ficheros, bases de datos, etc.
- Naming:
 - Modelo + RemoteDataSource (interfaz). Ejemplo: UserRemoteDataSource.
 - Tecnología + Modelo + RemoteDataSource (concreción). Ejemplo: FirestoreUserRemoteDataSource, ApiUserRemoteDataSource.

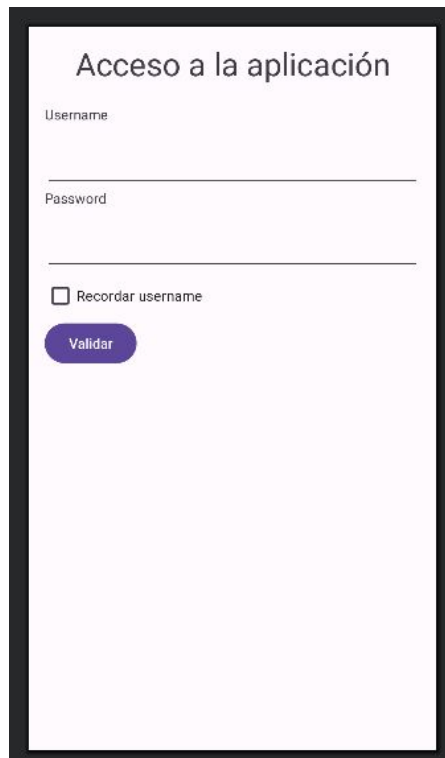
Ejercicios

Ejercicio 3:

- Crea una rama llamada ut2-ej3 desde main.
- Crea el flujo necesario para simular un login respetando la arquitectura vista: presentation, domain y data.
- La parte de validación de credenciales se hará en la clase remota y será un mock.
- Haz que la actividad del login sea la principal (la primera que se ejecuta al abrir la app)
- Actualiza el README con el link a este ejercicio.
- Crea una PullRequest.
- Fusiona la rama con la principal.
- No elimines la rama.

Ejercicios

Diseño básico del Login



A basic login form design with a light pink background and a dark border. The form contains the following elements:

- Header:** "Acceso a la aplicación" (Access to the application)
- Username field:** A text input field with the label "Username" above it.
- Password field:** A text input field with the label "Password" above it.
- Remember me checkbox:** A checkbox with the label "Recordar username" (Remember username).
- Submit button:** A purple button with the text "Validar" (Validate).

Ejercicios

Ejercicio 2:

- Se pide desarrollar la funcionalidad que me permite almacenar el username en local y autocompletarlo en el login.
- Crea una rama llamada ut2-ej4 desde main.
- Crea las clases necesarias en presentation, domain y data (local/mock).
- Actualiza el README con el link a este ejercicio.
- Crea una PullRequest.
- Fusiona la rama con la principal.
- No elimines la rama.