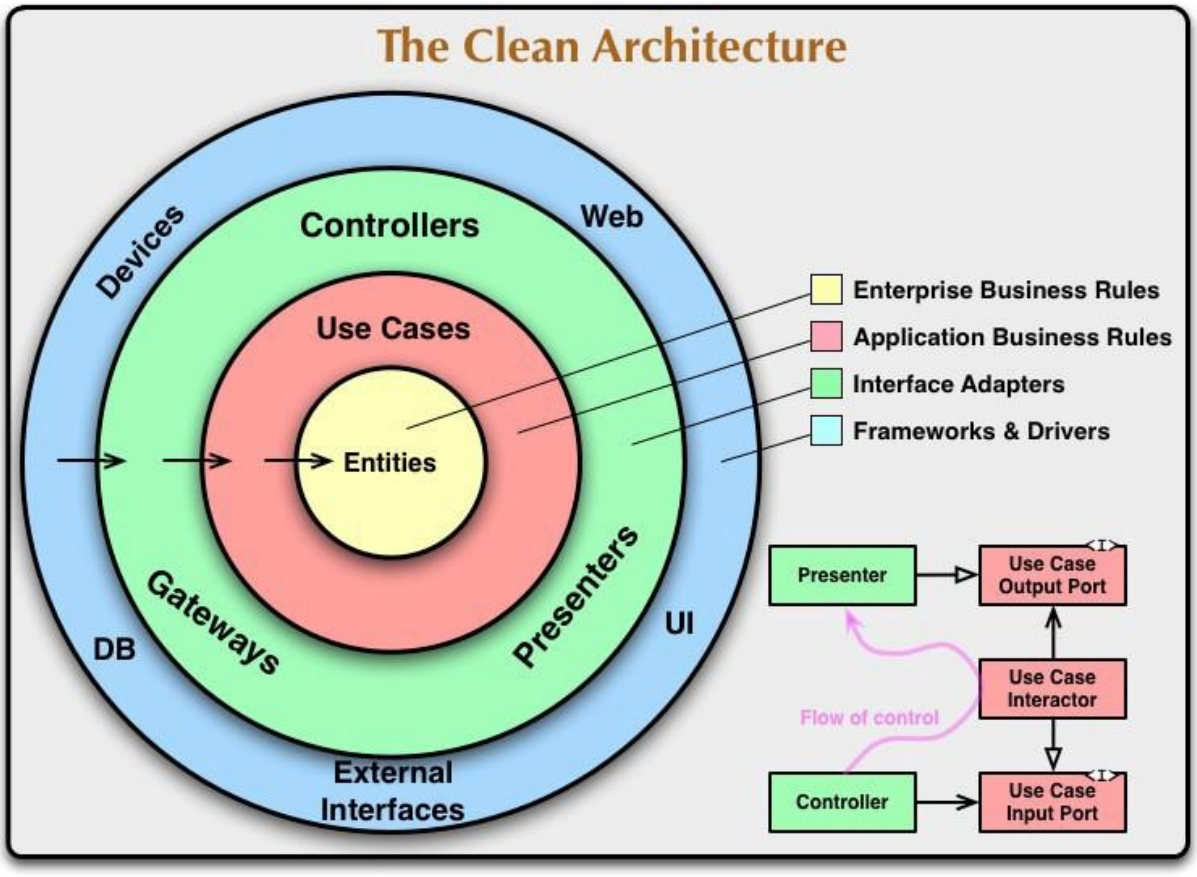
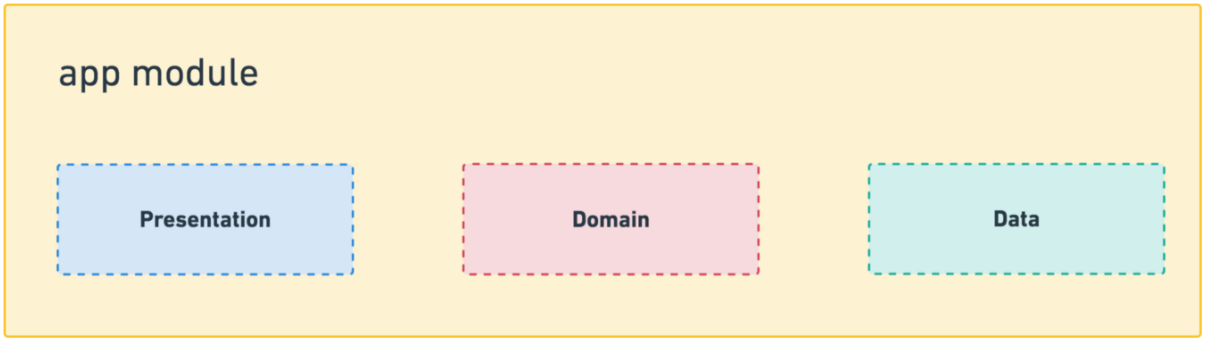
Before Starting, Let's know what **Clean Architecture** is? Its principles & it's advantages.

Introduction to Clean Architecture



For android we usually use 3 layer approach for Clean Architecture however Number of layers are not fix it can be more according to requirements



Clean Architecture is a software architecture intended to **keep the code under control** without all tidiness that spooks anyone from touching a code after the release. The main concept of Clean Architecture is the application code/logic which is very unlikely to change, has to be written without any direct dependencies. So it means that if I change my framework, database, or UI, the core of the system(Business Rules/ Domain) should not be changed. It means **external dependencies** are completely **replaceable**.

Rule of Clean Architecture

Each component should do only one task. If more tasks are there, separate it.

Benefits of Clean Architecture

- Independent of Database and Frameworks.
- Independent of the presentation layer. Anytime we can change the UI without changing the rest of the system and business logic.
- Highly testable, especially the core domain model and its business rules are extremely testable.

SOLID Principles for Clean Architecture

Single responsibility

Open-closed

Liskov substitution

Interface segregation

Dependency inversion

Single responsibility

A class should only have one job One reason to change If there are two reasons to change it should be split into two different classes

Open-closed

Open for extension, closed for modification If new functionality needs to be added,it should be added to an extension of the class Abstract away stable functionalityPut volatile functionality in extension classes

Liskov substitution

Low level classes can be substituted without affecting higher levels Achieved using abstract classes and interfaces

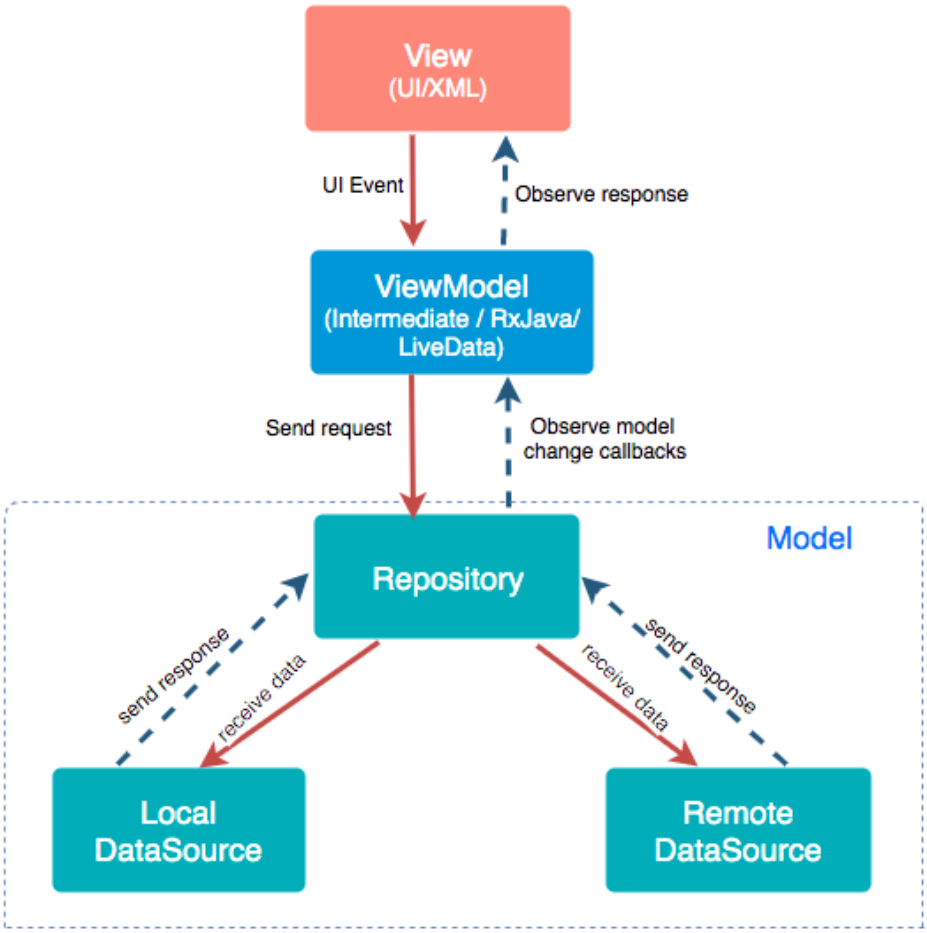
Interface segregation

Use interface to advertise functionality Many specific interfaces are better than one generic interface An interface only exposes the methods that the dependent class needs not more.

Dependency inversion

Concrete classes depend on abstract classes not the other way around Volatile functionality depends stable functionality Framework specific functionality depends on business logic

Introduction to MVVM



MVVM architecture is a Model-View-ViewModel architecture that removes the tight coupling between each component. Most importantly, in this architecture, the children don't have the direct reference to the parent, they only have the reference by observables.

Model

It represents the data and the business logic of the Android Application. It consists of the business logic - local and remote data source, model classes, repository.

View

It consists of the UI Code(Activity, Fragment), XML. It sends the user action to the ViewModel but does not get the response back directly. To get the response, it has to subscribe to the observables which ViewModel exposes to it.

ViewModel

It is a bridge between the View and Model(business logic). It does not have any clue which View has to use it as it does not have a direct reference to the View. So basically, the ViewModel should not be aware of the view who is interacting with. It interacts with the Model and exposes the observable that can be observed by the View.

Clean Architecture With MVVM