***MOBILE APP ARCHITECTURES AND DESIGN PATTERNS***

Mobile Application architectures and Design Patterns define how apps are structured, how components interact, and how maintainability, scalability, and performance are ensured.

Common architectures include; Layered, Monolithic, and Microservices, While design patterns like; MVC, MVP, and MVVM help structure code and UI.

**Architectures:**

1. ***Layered Architecture:***

It Organizes the app into layers (presentation, business logic, data access) for modularity and easier maintenance, especially for large, complex apps.

However, layered architecture also has its downsides. It can lead to high coupling between layers, making changes difficult to implement. Also, changes in one layer can affect all layers above it, leading to potential instability.

1. ***Monolithic Architecture:***

In this Architecture, all Components of the software are interconnected and interdependent, and the components cannot function independently of one another. It packages all components into a single unit, suitable for smaller, simpler apps, which makes the application easy to develop, test, and deploy. However they can be harder to scale and maintain, especially as the size and complexity of the application increase.

1. ***Microservices Architecture:***

Here, the application is built as a collection of small, independent, deployable services, enhancing scalability and flexibility, and also each service runs in its own process and communicates with others via well-defined APIs.

As others it also presents challenges. It can lead to complex systems and require substantial operational overhead. Furthermore, the need for inter-service communication can result in network latency and data consistency issues.

**Design Patterns:**

* ***Model-View-Controller (MVC):***

This is a Design Model that separates an application into three interacting parts which are: Model, View, and Controller. This separation allows for better code design and Modularization.

- Model: Represents application data and business logic.

- View: Displays data to the User

- Controller: Processes user input and controls data flow between Model and View

* ***Model-View-Presenter (MVP):***

This is a design model that separates an application into three interacting parts which are: Model, View and Presenter. This is similar to MVC but puts more responsibility on the Teacher to manage the interaction between Model and View.

- Model: Represents application data and business logic.

- View: Displays data to the User

- Presenter: Acts as an intermediary processing user input and updating the View and Model.

* ***Model-View-ViewModel (MVVM):***

MVVM is a design model widely used in mobile development, especially in frameworks like Android’s Jetpack. Its purpose is to separate the application into three parts: Model, View, and ViewModel.

- **Model**: Represents data and business logic.

-**View**: Represents the user interface.

-**ViewModel**: Acts as an interface between the Model and the View, which contains the reference logic.

* ***VIPER (View-Interactor-Presenter-Entity-Router):***

VIPER stands for View, Interactor, Presenter, Entity, and Router. VIPER is primarily based at the clean architecture ideas, which purpose to separate the concerns of different layers of the utility. Each layer has a single duty and communicates with different layers through properly-defined interfaces.

- *View:*

The view is chargeable for showing the information provided by way of the presenter and forwarding the person moves to the presenter.

- *Presenter:*

The presenter is liable for fetching the records from the interactor, reworking it right into a suitable layout for the view, and updating the view hence. The presenter additionally handles the consumer movements acquired from the view and calls the router to navigate to other screens

- *Interactor:*

The interactor is accountable for gaining access to the facts from the service layer, acting any vital operations on it, and returning it to the presenter. The interactor additionally communicates with the entity layer to store and retrieve the information fashions.

- *Entity*:

The entity is responsible for representing the data in a constant and coherent manner throughout the software. The entity layer also can encompass records get entry to gadgets (DAOs) or repositories that summary the information of records patience and retrieval.

- *Router*:

The router is chargeable for developing and providing the view controllers, passing any vital facts to them, and coping with any dependencies or configurations. The router also communicates with the presenter to get hold of the navigation requests and execute them.

* ***Singleton:***

The singleton policy ensures that there is only one instance of a class and provides global access. This is especially useful when you want to manage a single instance of an object or control access to a delayed object.

For example: Singleton can be used to manage player’s score in mobile game. There can only be one instance that is responsible for tracking scores and is updated throughout the game.

* ***Factory Method:***

The Factory Method model defines an interface for creating an object but allows subclasses to modify the type of the created object. Especially useful when you need to create objects with a common interface but different functionality.

For Example: In a mobile app that supports multiple payment gateways, payments can be made using the Factory Method. Each payment gateway (e.g., PayPal, Stripe) is a small business and provides its services.

* ***Observer:***

The observer structure defines one to many dependencies between objects, so when one object changes its state, all its dependents are automatically notified and updated. This is useful for scheduling distributed events.

For Example: In the reports app, many features (Observers) such as the title widget, the report feed view, and the notification provider (Themes) can subscribe to updates when new information arrives. The observer model ensures that they are created all registered parts report, and accordingly You can update it.

* ***Dependency Injection (DI):***

Dependency Injection is a method of providing class dependencies from the outside, rather than creating them in the class. It improves code modularity and testability by making classes independent of their dependencies.

For Example: In an Android app, instead of creating a single database connection object in the class, you can place the database object externally, allowing you to easily test and modify database operations.

***Conclusion:***

Design processes play an important role in mobile app development by providing proven solutions to common software design challenges. Using this framework allows developers to create maintainable, extensible, and efficient applications. Understanding when and how to apply these options can significantly improve the quality of your mobile app codebase.