**Architecture**

Architecture trong Android là một tập hợp các thư viện giúp bạn thiết kế các ứng dụng một cách mạnh mẽ, có thể kiểm tra và bảo trì được. Bắt đầu với các lớp để quản lý vòng đời thành phần UI của bạn và xử lý độ bền của dữ liệu.

**Standard android ( Model – View – Controller)**

Là một mô hình kiến trúc phân tách một ứng dụng thành ba thành phần logic chính: the Model , the View và the Controller. Mỗi thành phần này được xây dựng để xử lý các khía cạnh phát triển cụ thể của một ứng dụng. MVC là một trong web tiêu chuẩn ngành được sử dụng thường xuyên nhất để tạo ra các dự án có thể mở rộng.những khung phát triển

* + - Model : nó bao gồm tất cả dữ liệu và logic liên quan của nó.
    - View: Giao diện người dùng đồ họa hoặc giao diện người dùng (GUI).
    - Controller: Là lớp logic. Nó nhận được thông báo về hành vi của người dùng và cập nhật cho Model theo yêu cầu.

MVC giúp bạn chia nhỏ mã giao diện người dùng và mã phụ trợ thành các thành phần riêng biệt. Bằng cách này, sẽ dễ dàng hơn nhiều để quản lý và thực hiện các thay đổi cho cả hai bên mà không ảnh hưởng lẫn nhau.

MVC chỉ là một mẫu áp dụng tốt cho web. Đặc biệt, nó áp dụng tốt cho ý tưởng rằng một ứng dụng được truy cập theo cách sau:

* + - Người dùng yêu cầu một số tài nguyên
    - Một số dữ liệu cơ bản được lấy từ một nơi nào đó.
    - Sau đó, một mẫu sẽ được áp dụng cho dữ liệu đó để hiển thị cho người dùng.

**Clean architecture (Model – View – Presenter)**

MVP là một dẫn xuất của mẫu kiến trúc MVC (Model View Controller example). Nó được sử dụng để xây dựng giao diện người dùng. MVP cho phép tách lớp trình bày khỏi logic. Nó có nghĩa là tất cả mọi thứ từ cách hoạt động của giao diện đến cách nó thể hiện trên màn hình. MVP là một mẫu kiến trúc giao diện người dùng, giúp giảm bớt quá trình kiểm tra đơn vị tự động và nó có trách nhiệm cung cấp mã sạch

* Model: nắm giữ logic nghiệp vụ cũng như kiểm soát cách dữ liệu được tạo, lưu trữ và sửa đổi. Trong Android, nó là một lớp truy cập dữ liệu chẳng hạn. Model bao gồm các thành phần chịu trách nhiệm về các chức năng như tạo, lưu trữ, hiển thị và tìm nạp dữ liệu. Tất cả các chức năng này thường thực hiện trong luồng nền.
* View: là một giao diện thụ động, hiển thị dữ liệu và định tuyến các hành động của người dùng đến Presenter. Trong Android, Chế độ xem được biểu thị bằng Activity, Fragment or View. The View chứa giao diện người dùng và nó không chứa bất kỳ logic hoặc kiến thức nào về dữ liệu được hiển thị.
* Presenter: nằm giữ Model và View. Và, nó kích hoạt logic nghiệp vụ và cho phép biết "View" khi nào cần cập nhật. Nó khôi phục dữ liệu nhận được từ Model và hiển thị nó trong View. Nó tương tác với Model, sau đó tìm nạp và chuyển đổi dữ liệu từ Model để cập nhật View.

Trong Android, ứng dụng phải dễ dàng mở rộng và bảo trì. Vì thế, để duy trì mức độ, điều quan trọng là phải xác định tốt các lớp được tách biệt. Và, MVP giúp mọi thứ dễ dàng hơn cho các developer và nó làm cho các lượt xem độc lập với nguồn dữ liệu.

MVP (Model – View – Presenter) rất được khuyến khích cho các nhà phát triển Android để xây dựng một ứng dụng Android nhanh nhẹn và có thể mở rộng với chi phí tối thiểu so với MVC.

**Data-binding MVVM (Model-View-ViewModel)**

Là một kiến trúc Model-View-ViewModel loại bỏ sự kết hợp chặt chẽ giữa từng thành phần. Quan trọng nhất, trong kiến trúc này, con cái không có tham chiếu trực tiếp đến cha mẹ, chúng chỉ có tham chiếu bởi những người quan sát.

* + - Model: Nó đại diện cho dữ liệu và logic kinh doanh của ứng dụng Android . Nó bao gồm logic nghiệp vụ - nguồn dữ liệu cục bộ và từ xa, các lớp mô hình, kho lưu trữ.
    - View: Nó bao gồm Mã giao diện người dùng (Fragment,Activity), XML…Nó gửi hành động của người dùng đến ViewModel nhưng không nhận lại phản hồi trực tiếp. Để nhận được phản hồi, nó phải đăng ký đểcó thể quan sát những gì ViewModel hiển thị với nó.
    - ViewModel: Nó là cầu nối giữa View và Model (business logic). Nó không có bất kỳ manh mối nào mà View phải sử dụng nó vì nó không có tham chiếu trực tiếp đến View.

MVVM hướng dẫn chúng ta cách phân phối trách nhiệm giữa các lớp trong ứng dụng GUI (hoặc giữa các lớp ), với mục tiêu có một số lượng nhỏ các lớp, trong khi vẫn giữ cho số lượng trách nhiệm trên mỗi lớp nhỏ và được xác định rõ ràng. MVVM thích hợp giả định ít nhất là một ứng dụng phức tạp vừa phải, xử lý dữ liệu mà nó nhận được từ "một nơi nào đó". Nó có thể lấy dữ liệu từ cơ sở dữ liệu, tệp, dịch vụ web hoặc từ vô số nguồn khác.

**Solid**

**S**ingle responsibility principle( Nguyên tắc nhiệm vụ duy nhất)

Một lớp nên có một và chỉ một lý do để thay đổi, nghĩa là một lớp chỉ nên có một công việc.

**O**pen-Closed Principle( Nguyên tắc đóng mở)

Các đối tượng hoặc thực thể phải được mở để mở rộng nhưng bị đóng để sửa đổi.

**L**iskov Substitution Principle( Nguyên tắc thay thế Liskov)

Nếu lớp A là một kiểu con của lớp B, chúng ta có thể thay thế B bằng A mà không làm gián đoạn hoạt động của chương trình của chúng ta.

**I**nterface Segregation Principle(Nguyên tắc phân tách giao diện)

Nó chỉ đơn giản có nghĩa là các giao diện lớn hơn nên được chia thành các giao diện nhỏ hơn. Bằng cách đó, chúng ta có thể đảm bảo rằng việc triển khai các lớp chỉ cần quan tâm đến các phương thức mà chúng quan tâm.

**D**ependency Inversion Principle(Nguyên tắc đảo ngược)

Các thực thể phải phụ thuộc vào những điều trừu tượng, không phụ thuộc vào những cụ thể. Nó nói rằng mô-đun cấp cao không được phụ thuộc vào mô-đun cấp thấp, nhưng chúng phải phụ thuộc vào các phần trừu tượng.

Nguyên tắc SOLID là một tiêu chuẩn mã hóa mà tất cả các nhà phát triển nên có một khái niệm rõ ràng để phát triển phần mềm đúng cách để tránh một thiết kế xấu. Khi được áp dụng đúng cách, nó làm cho mã của bạn dễ mở rộng, logic và dễ đọc hơn.

Khi nhà phát triển xây dựng phần mềm theo một thiết kế xấu, mã có thể trở nên không linh hoạt và dễ gãy hơn. Những thay đổi nhỏ trong phần mềm có thể dẫn đến lỗi. Vì những lý do này, chúng ta nên tuân theo nguyên tắc SOLID.

**Design pattern**

Design Pattern là một giải pháp chung để giải quyết các vấn đề thường gặp khi thiết kế phần mềm trong lập trình hướng đối tượng

Tại sao nên sử dụng Design pattern?

* Tăng tốc độ phát triển phần mềm
* Hạn chế các lỗi tiềm ẩn
* Hỗ trợ sử dụng lại code
* Làm cho code dễ đọc hơn

Creational design patterns

Có thể chia thành các class – creation và object – creation pattens. Trong khi class-creation patterns sử dụng hiệu quả kế thừa trong quá trình khởi tạo, các mẫu tạo đối tượng sử dụng ủy quyền một cách hiệu quả để hoàn thành công việc.

1. Abstract Factory

Tạo một phiên bản của một số họ lớp.

1. Builder

Tách việc xây dựng đối tượng khỏi biểu diễn của nó.

1. Factory Method

Tạo một phiên bản của một số lớp dẫn xuất.

1. Object Pool

Tránh mua lại và giải phóng tài nguyên tốn kém bằng cách tái chế các Object không còn sử dụng.

1. Prototype

Một phiên bản được khởi tạo đầy đủ để được sao chép hoặc nhân bản.

1. Singleton

Một lớp mà chỉ một cá thể duy nhất có thể tồn tại..

Structural design patterns

Các mẫu thiết kế này đều là về thành phần lớp và đối tượng. Các class – creation cấu trúc sử dụng kế thừa để soạn giao diện. Các object – patterns có cấu trúc xác định các cách soạn đối tượng để có được chức năng mới.

1. Adapter

Khớp các giao diện của các lớp khác nhau

1. Bridge

Tách giao diện của một đối tượng khỏi việc triển khai nó

1. Composite

Cấu trúc cây của các đối tượng đơn giản và tổng hợp

1. Decorator

Thêm trách nhiệm cho các đối tượng một cách động

1. Facade

Một lớp duy nhất đại diện cho toàn bộ hệ thống con

1. Flyweight

Một phiên bản chi tiết được sử dụng để chia sẻ một cách hiệu quả

1. Private Class Data

Hạn chế quyền truy cập của trình truy cập / trình đột biến

1. Proxy

Một đối tượng đại diện cho một đối tượng khác

Behavioral design patterns

Các mẫu thiết kế này đều là về giao tiếp các đối tượng của Lớp. Các patterns hành vi là những patterns quan tâm cụ thể nhất đến giao tiếp giữa các đối tượng..

1. Chain of responsibility

Một cách truyền yêu cầu giữa một chuỗi các đối tượng

1. Command

Đóng gói một yêu cầu lệnh dưới dạng một đối tượng

1. Interpreter

Bao gồm các yếu tố ngôn ngữ trong một chương trình

1. Iterator

Truy cập tuần tự các phần tử của một tập hợp

1. Mediator

Xác định giao tiếp đơn giản giữa các lớp

1. Memento

Chụp và khôi phục trạng thái bên trong của đối tượng

1. Null Object

Được hiết kế để hoạt động như một giá trị mặc định của một đối tượng

1. Observer

Thông báo thay đổi đối với một số lớp

1. State

Thay đổi hành vi của một đối tượng khi trạng thái của nó thay đổi

1. Strategy

Đóng gói một thuật toán bên trong một lớp

1. Template method

Trì hoãn các bước chính xác của thuật toán cho một lớp con

1. Visitor

Xác định một hoạt động mới cho một lớp mà không thay đổi