Flutter App Architecture

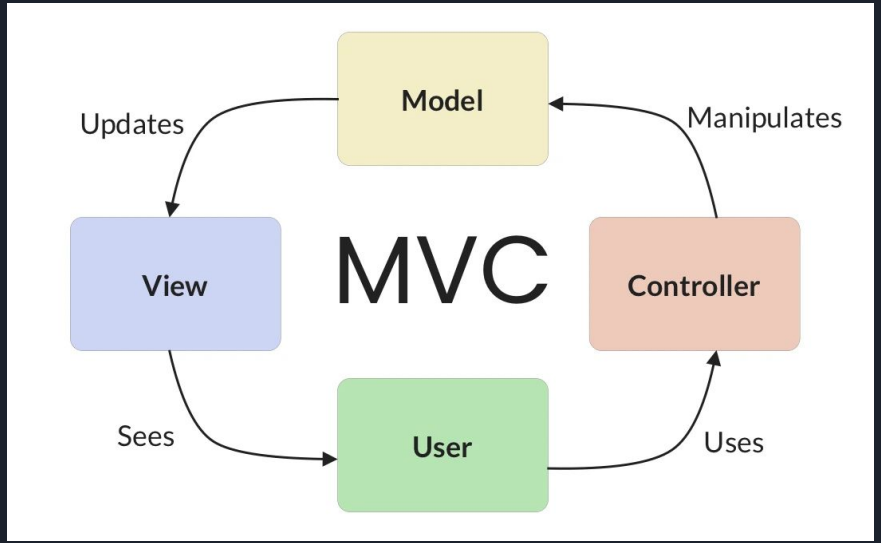
RICKSON DPENHA

MVC Pattern

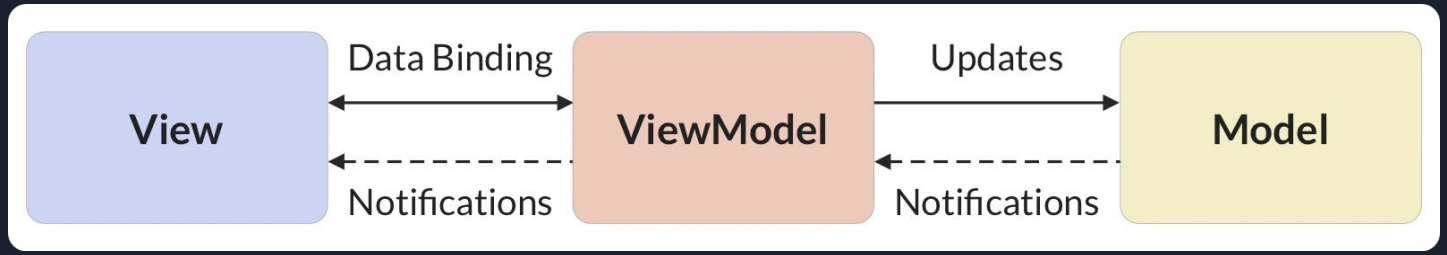
Model: Holds the data and business logic.

View: UI components, screens, Presentation logic (date formats, translations etc).

Controller: The Controller acts as an intermediary between the Model and the View. It handles user interactions, communicates with the Model to retrieve or modify data, and updates the View accordingly.



MVVM Pattern

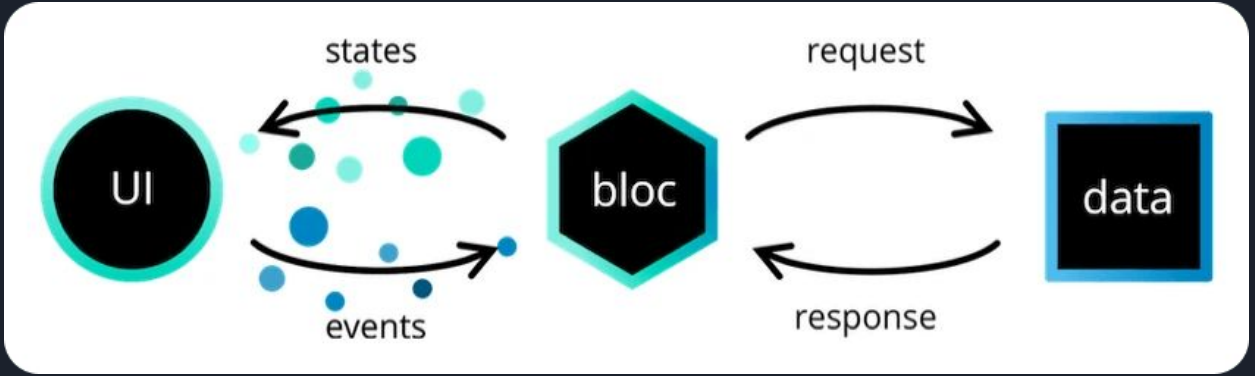


View: Screens & UI Components. Simple as possible

ViewModel: Interacts between View & Model. Holds the presentation logic, manages lifecycle, state, navigation etc

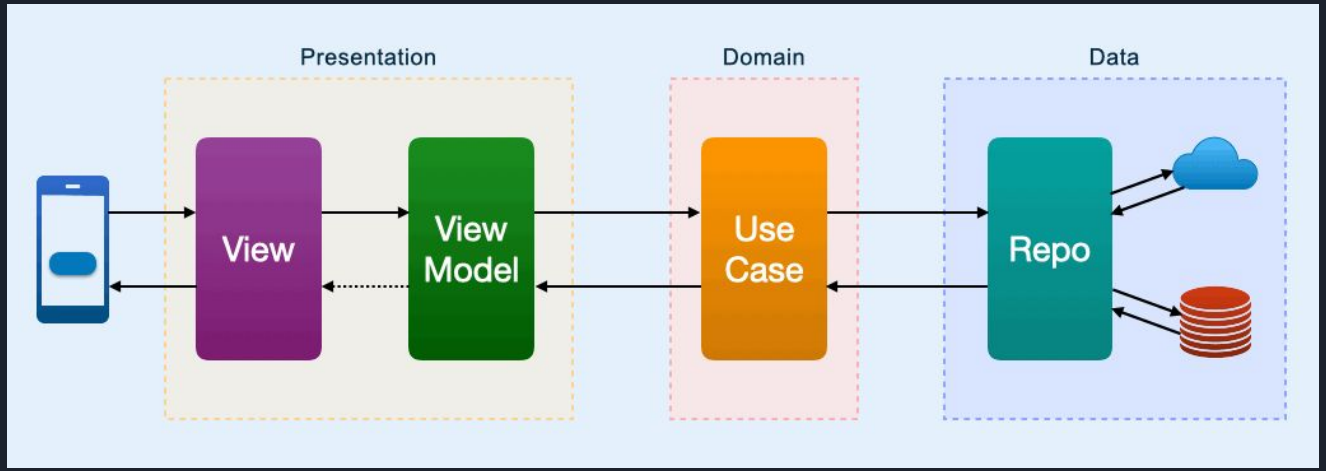
Model: Data layer and holds business logic.

Bloc Pattern



- Uses Streams to manage state
- Opinionated and well structured
- Comparible to MVC & MVVM patterns
- Simplifies State management using Cubits

Clean Architecture



Presentation: This layer includes the UI, State and Presentation logic. Testable with Widget testing

Domain: This layer consists of pure business logic. Easily testable with Unit Testing.

Data: The data layer where repositories and data sources are implemented. Contains network, local and platform code. Easily mockable for testing.

WHY MVVM?

- Modular Architecture, Easily picked up by native developers who are transitioning to flutter framework.
- Separates UI (presentation) code from business logic, which will help us write tests (unit & widget) easily.
- By using the right state management package we can easily maintain the code when making changes.
- Easy to maintain & scale.
- Best for large team.
- Can be implemented in existing apps

Overview of MVVM pattern in Flutter

- **Stacked package**
  - Ready to use architecture
  - Structured and Opinionated
  - Faster Development time
  - Tries to do lot of things like navigation, forms, Services etc
  - Entire app will be dependent on this package
  - Developers will be learning Stacked and not flutter
- **Provider package**
  - Simple & Lightweight
  - Reactive programming
  - Devtools support
  - Runtime errors
  - Boilerplate code
  - Accidental State updates
- **Bloc package**
  - Structured and Opinionated
  - Testability & Scalability
  - Community Support
  - Boilerplate code
  - Learning curve
  - Complexity
- **Riverpod**
  - Reactive and caching
  - Testable and Scalability
  - Compile Time safe
  - Learning curve
  - New package
  - Too many providers can be difficult to maintain

App Navigation

GoRouter :

- Maintained by flutter team
- Support for deep-links out of the box
- Declarative Routing

AutoRoute :

- Generates route using code generation
- Tons of features like IDE plugins
- Good Documentation

Project Structure

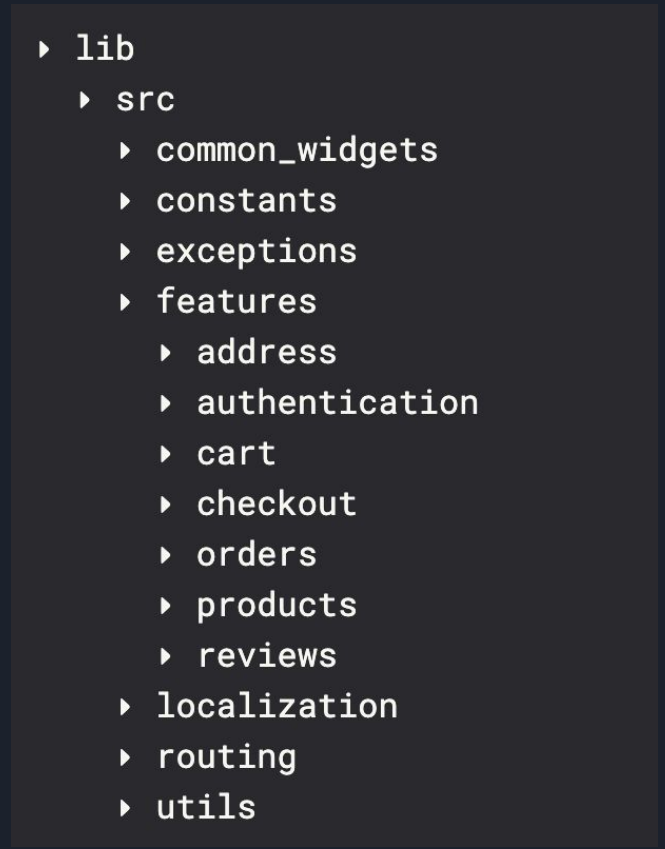
Layer First approach

```
▸ lib
  ▸ src
    ▸ presentation
      ▸ feature1
      ▸ feature2
    ▸ application
      ▸ feature1
      ▸ feature2
    ▸ domain
      ▸ feature1
      ▸ feature2
    ▸ data
      ▸ feature1
      ▸ feature2
```

Feature First approach

```
▸ lib
  ▸ src
    ▸ features
      ▸ feature1
        ▸ presentation
        ▸ application
        ▸ domain
        ▸ data
      ▸ feature2
        ▸ presentation
        ▸ application
        ▸ domain
        ▸ data
```

Project Structure



Packages & Plugins

Core packages:

- Provider / Riverpod
- AutoRoute / GoRouter (Navigation)

Utils:

- Collection
- Freezed / Json Serializable
- Http / Dio
- Logger
- Mokito
- Flutter_form_builder
- Intl

Good to have:

- flex_color_scheme
- Firebase crashlytics / Sentry
- Firebase Remote Config (Feature Flags)
- Firebase Analytics
- Upgrader
- Auto_size_text
- flutter_gen_runner
- Widgetbook
- App_review