Architecture Documentation

➢ What is Architecture in Flutter?

What comes to mind when you hear or see the word "architecture"? The skeleton and inner workings of a framework or tool? You are absolutely right.

In simple terms, Flutter's architecture is the foundation behind the high-quality, cross-platform, device-agnostic apps built using the Flutter framework on the Dart programming language ecosystem

Ref Site: https://dev.to/bigdexter/understanding-flutter-architecture-part-1-introduction-2h09#:~:text=In%20simple%20terms%2C%20Flutter's%20architecture,the%20Dart%20programming%20language%20ecosystem

➢ Previous used architecture

MVVM (Model-View-ViewModel)

MVVM stands for Model View-ViewModel and is a design pattern used in software development. It divides the application's data model, the user interface, and the application logic into different layers. MVVM is used in many software development environments, such as mobile development, to enhance code organization, reuse, and maintainability.

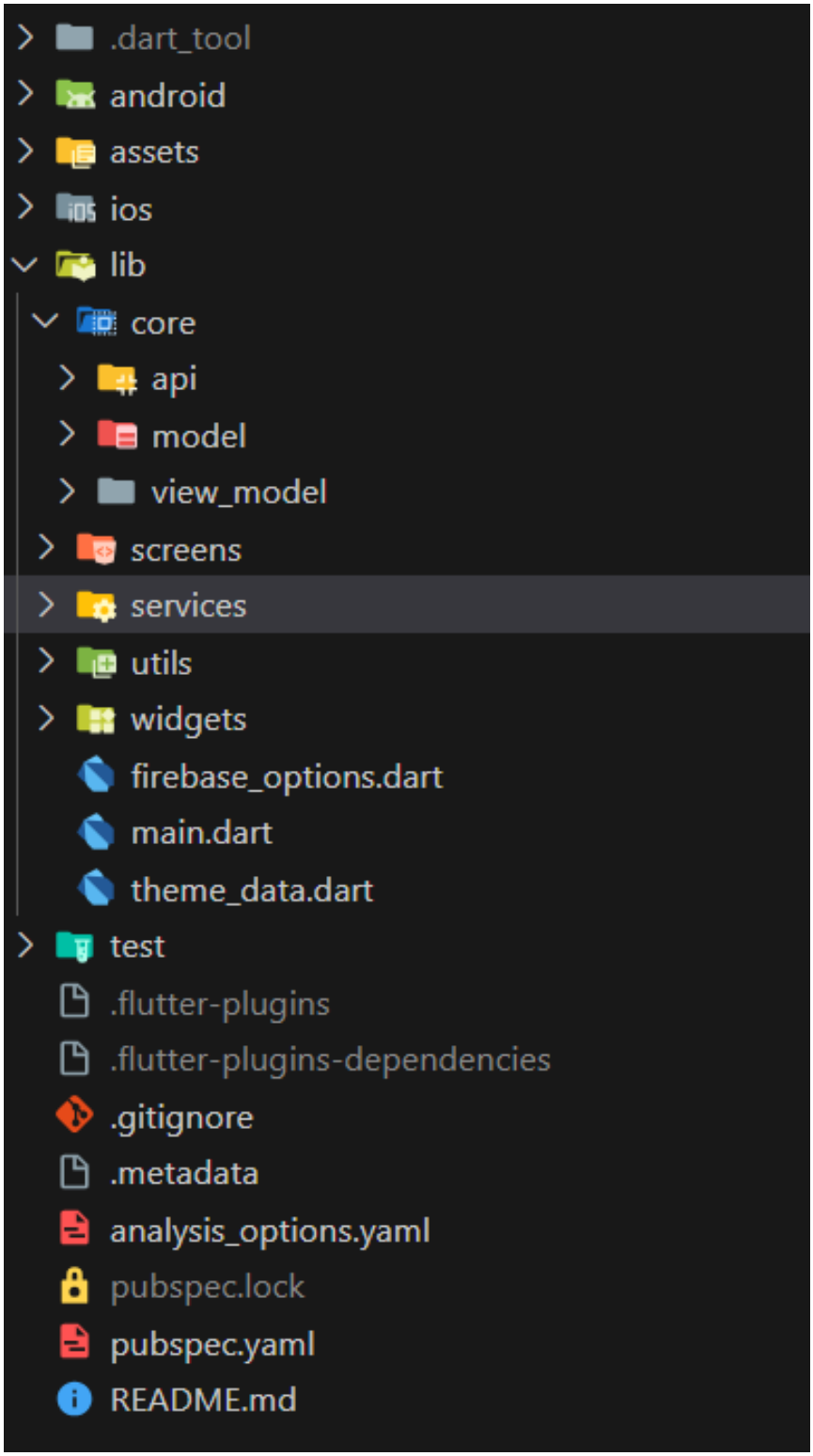
Model: The data model describes the application's data structure and business logic. The Model is a collection of model classes, repository classes, and utility classes. Model classes represent the application's data objects. Repository classes manage all network requests. Utility classes provide the helper functions used throughout the application.

View: View is responsible for showing the UI that captures all the user events that display the data and interaction with the user.

ViewModel: The ViewModel connects the Model to the View, and the View is responsible for managing all the user events and business logic within the application. The View provides data to the View and conveys user interactions to the Model. It contains all the classes related to the ViewModel that manage user event requests and update the View.

Ref Site: https://crm-masters.com/what-is-mvvm-architecture-in-flutter/

➢ **MVVM folder structure**

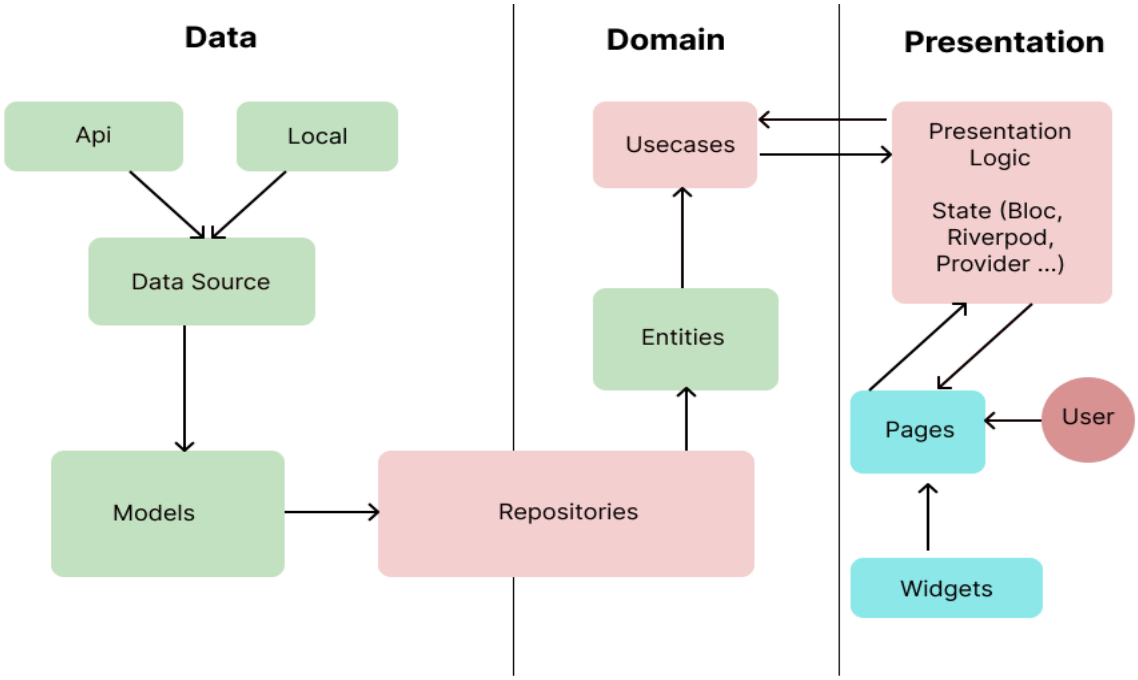


➢ What is Clean Architecture?

Clean Architecture is a software design paradigm introduced by Robert C. Martin, and it aims to create maintainable and scalable software by organizing the codebase into distinct layers with clear dependencies and responsibilities.

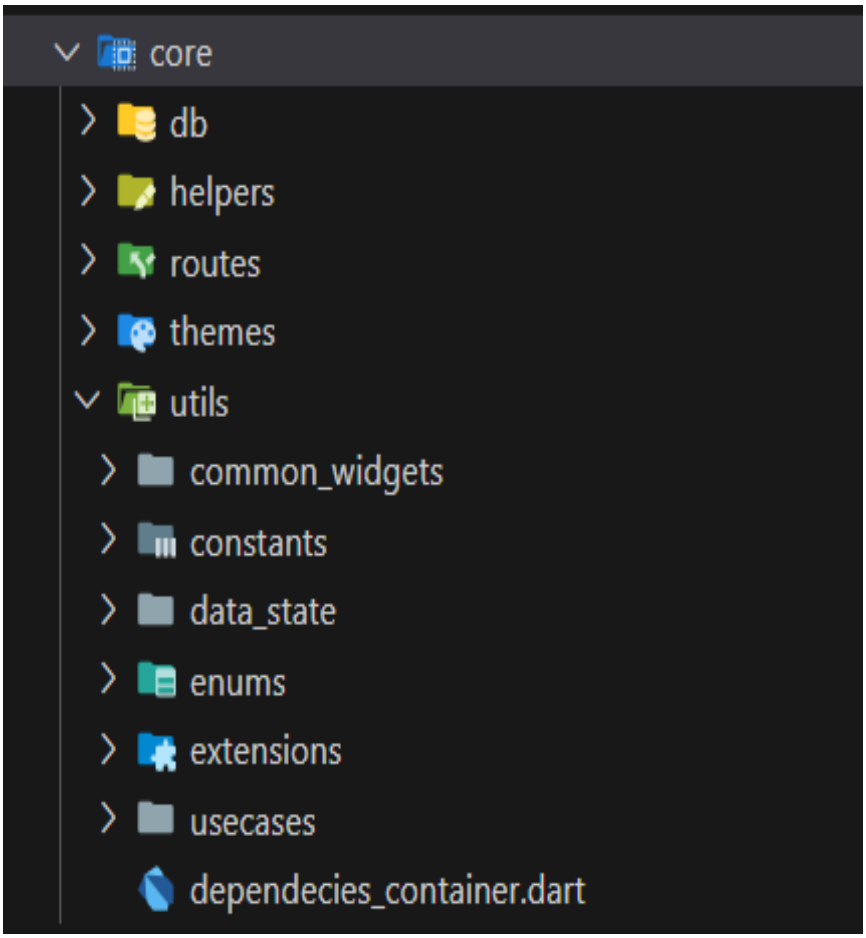
➢ Depth explanation of Clean Architecture with Flutter

Clean Architecture provides a way to structure applications that separate the different components of an application into modules, each with a well-defined purpose. The main idea behind Clean Architecture is to separate the application into three main layers: the **presentation** layer, the **domain** layer, and the **data** layer.



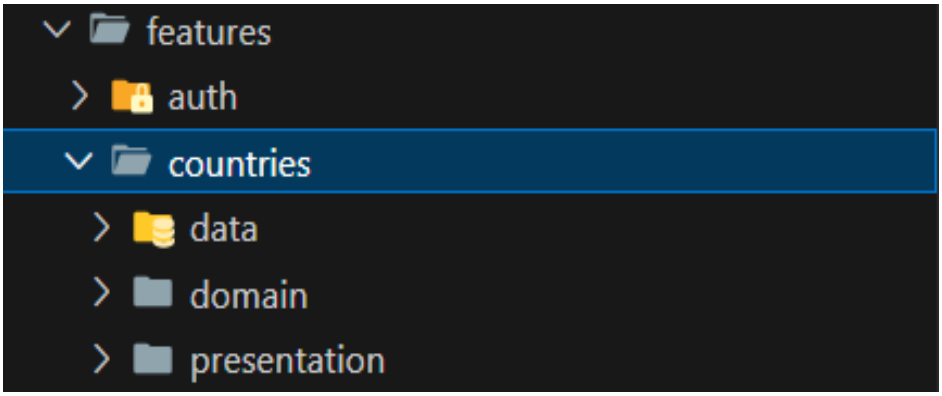
➢ **Main folder structure**

➢ **Core folder structure**



➢ Feature folder structure

We will separate the project folder base on the feature. Also separate the feature into three layers - Data, Domain, Presentation layer.



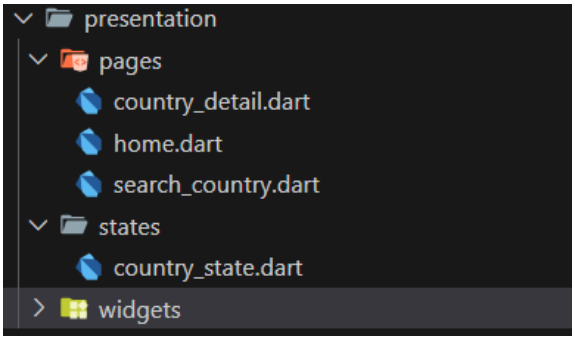
1- Presentation Layer

Responsibility

The Presentation Layer is the outermost layer, responsible for presenting information to the user and capturing user interactions. It includes all the components related to the user interface (UI), such as widgets, screens, and presenters/controllers (State Management).

Components

- **Screens**: Represent the feature screens.
- **Widgets and UI Components:** Represent the visual elements of the application.
- **Manager/Controllers**: Contain the presentation logic that interacts with the UI components. They receive user input, communicate with the Use Cases in the Domain Layer, and update the UI accordingly.



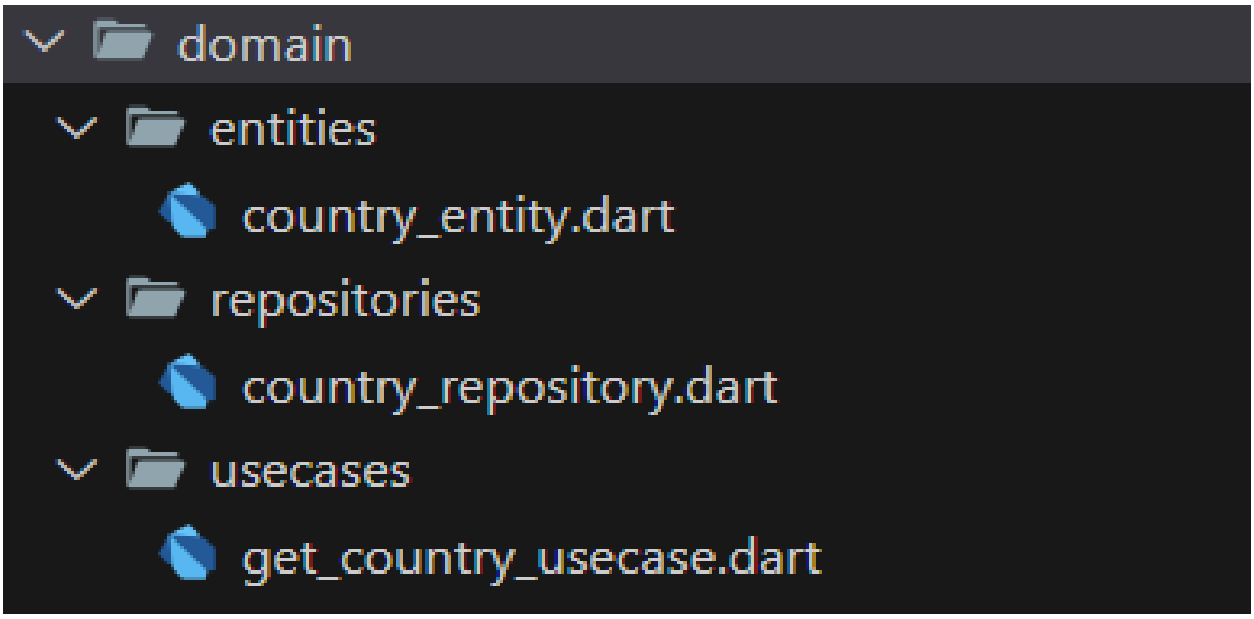
2- Domain Layer

Responsibility

The Domain Layer, also known as the Business Logic or Use Case Layer, contains the core business rules and logic of the application. It represents the heart of the software system, encapsulating the essential functionality that is independent of any particular framework.

Components

- **Entities:** Represent the fundamental business objects or concepts.
- **Use Cases**: Contain application-specific business rules and orchestrate the flow of data between entities. They are responsible for executing specific actions or operations.
- **Business Rules and Logic (Repository):** Core functionality that is crucial to the application's domain.



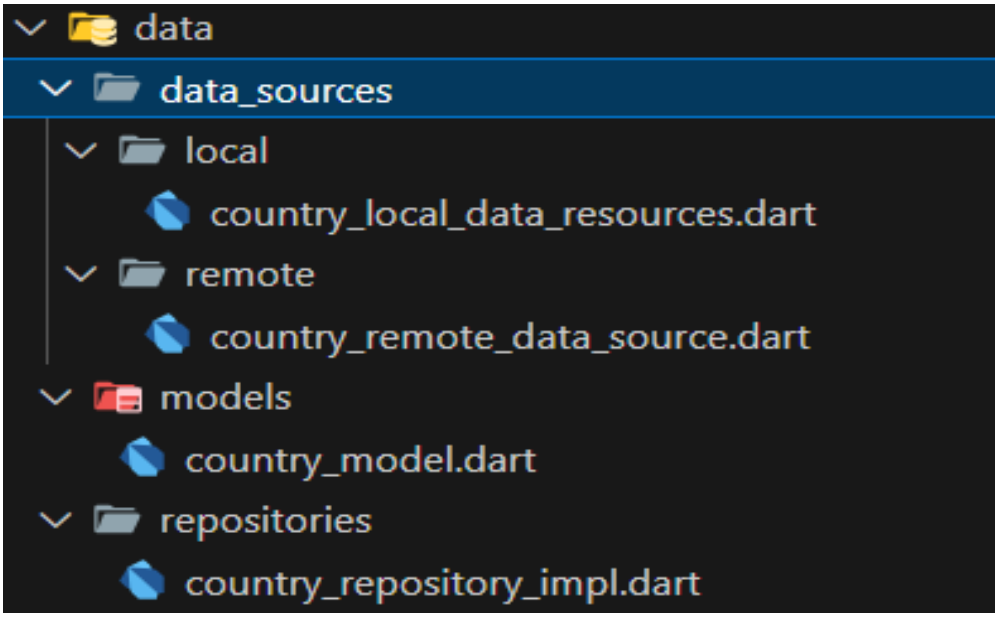
3- Data Layer

Responsibility

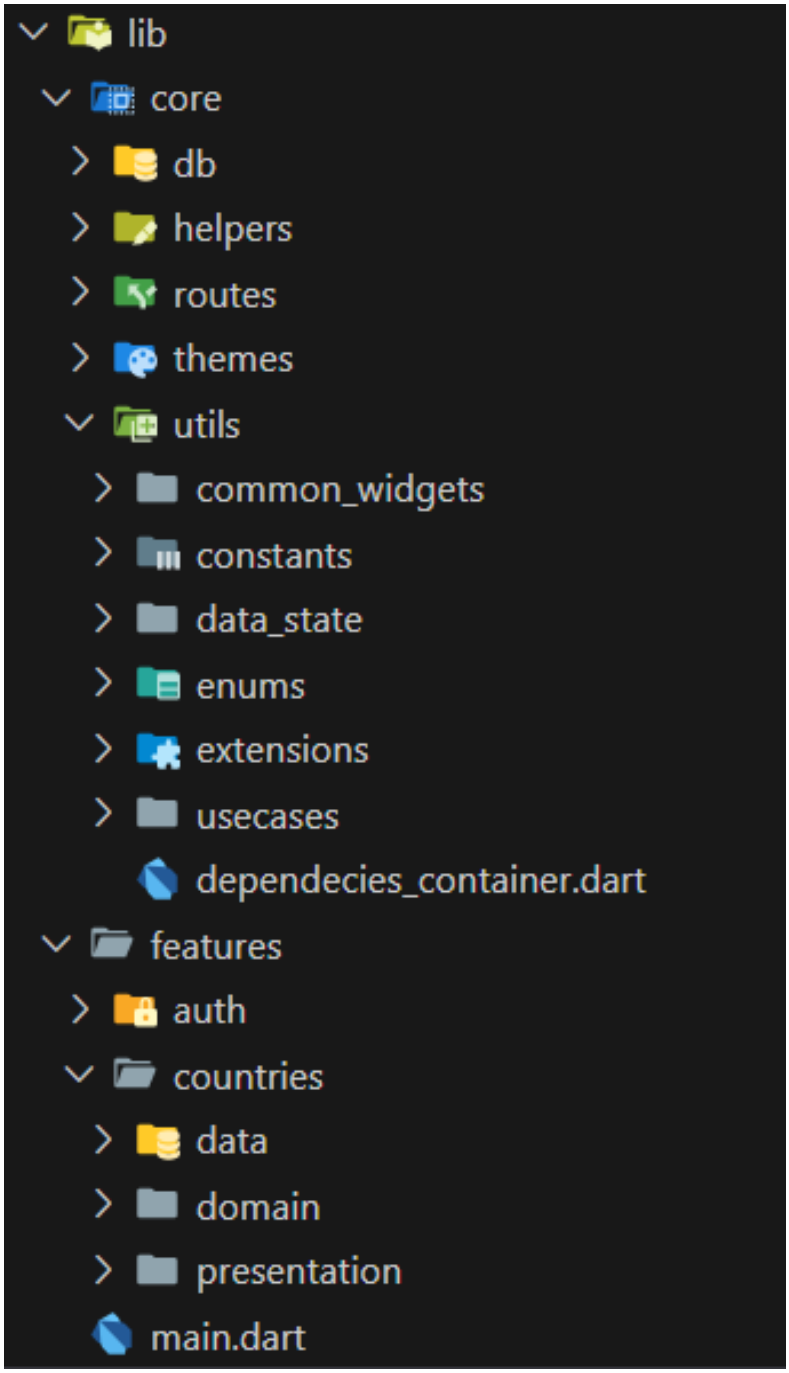
The Data Layer is responsible for interacting with external data sources, such as databases, network services, or repositories. It handles the storage and retrieval of data.

Components

- **Repositories or Gateways**: Abstract interfaces that define how data is accessed and stored.
- **Data Models:** Represent the structure of the data as it is stored in the external data sources.
- **Data Sources:** Implementations of repositories that interact with databases, APIs, or other external services.



Folder Structure Summary



See demo project here:
https://github.com/AungKhantSoe21/clean_architecture_flutter