Clean Architecture in Flutter – All you need no know!

***Student**: Elias D. Franco.*

***Date:** 20/07/2025.*

Link Video:

https://youtu.be/zon3WgmcqQw?si=nnnv5AlLQUZW2Mr3

Have you ever heard about clean Architecture? You must have heard it, but do you know what it is exactly? Clean Architecture is the blueprint for a modular system which strictly follows the design principle called separation of concerns. More specifically, this style of architecture focuses on dividing software into layers to simplify the development and maintenance of the system itself. When layers are well separated, individual pieces can be reused as well as developed and updated independently. Clean Architecture is one of the most powerful solutions to build clean apps with independent data layers that multiple teams can work on. The resulting app would also be scalable, readable, testable, and can be easily maintained at any time. As we can see in the diagram, we have three main layers of the architecture: data, domain, and the feature layer, which is the same as the presentation layer. We also have two additional supporting layers: the resources and shared library.

Presentation layer: this layer presents the app content and triggers events that modify the application state. This layer has three parts: Pages part, which includes application pages; State Management part, which contains files related to state management that we use such as Bloc or Riverpod or other state managements; and the Widgets part, which includes all the specific widgets that we use on the pages.

Domain layer: domain layer is the innermost part of the layers and has no dependency on other layers. It contains entities, repository interfaces, and use cases. The domain layer would be written purely in Dart without any Flutter elements. The reason is that the domain should only be concerned with the business logic of the application, not with the implementation details. The entities must be our data types or classes that are used in different parts of our software, or in other words, we define in our version of clean that our entities are the objects that can be returned to us or we can send to an API. Repositories in domain layer are abstract classes or contracts and define the properties and methods that our project will need in a specific feature. Use cases include application-specific business rules. Each event is an interaction of the user with the system and we can call this a use case like sign up, log in, and other interactions. Use cases are nothing more than a bridge between layers — it's a single call to business logic.

Data layer: the data module, which is a part of the outermost layer, is responsible for data retrieval. This can be in the form of API calls to a server or a local database. It also contains repository implementations. This layer has three parts: one of the parts is the repository part — it includes actual implementations of the repositories in the domain layer. Repositories are responsible to coordinate data from the different data sources. The other part is the data source — it consists of remote and local data sources. Remote data source will perform HTTP requests on the API, while local data source will cache or persist data. And the last part is models, which represent the JSON structure and allow us to interact with our data sources.

Now let's go to create the folder structure in the project. The first folder that we need to create is the features folder. Every feature that the app has is placed inside this folder, for example, the auth feature. Every feature of the app will be divided into three main layers: presentation, domain, and data layers. So for each feature, we have to create these three folders. As we said before, the presentation layer has three parts called Pages, State Management (for example Bloc), and the Widgets part. And also, as I explained, the domain layer has three parts called Use Cases, Entities, and Repository — we will create these as well. And finally, we have to create folders for the data layer. In the data layer we have three parts named Repository, Data Sources, and Models.

Apart from the features folder, we can have other folders in the project. Usually we have two folders: config and core. In the configs folder we put project-related configurations such as theme or routes. And in the core folder we usually pull anything which has to be shared between multiple features into the core folder such as network, error, util, or use cases. So in general, the structure of the project with clean architecture will be like this, but we may have other folders. The base structure is like this. In the next videos we will do many projects with clean architecture, so stay with us and make sure to hit the subscribe button to get the next video.