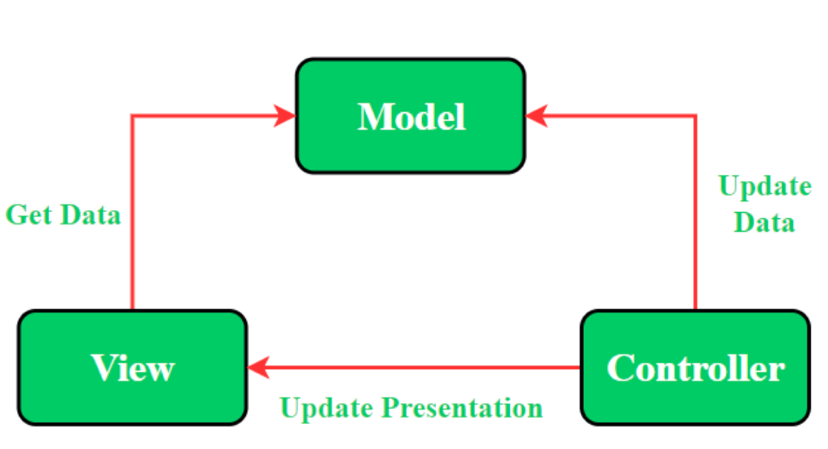
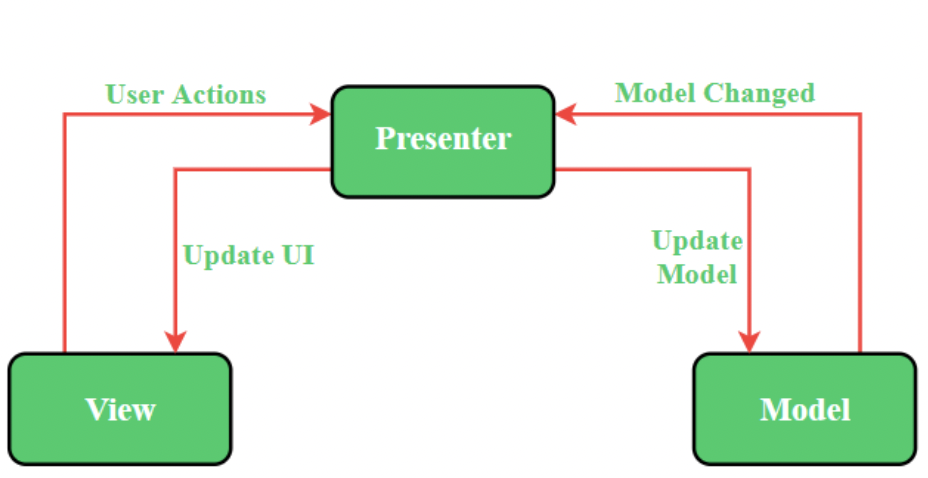
**Design Patterns**

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



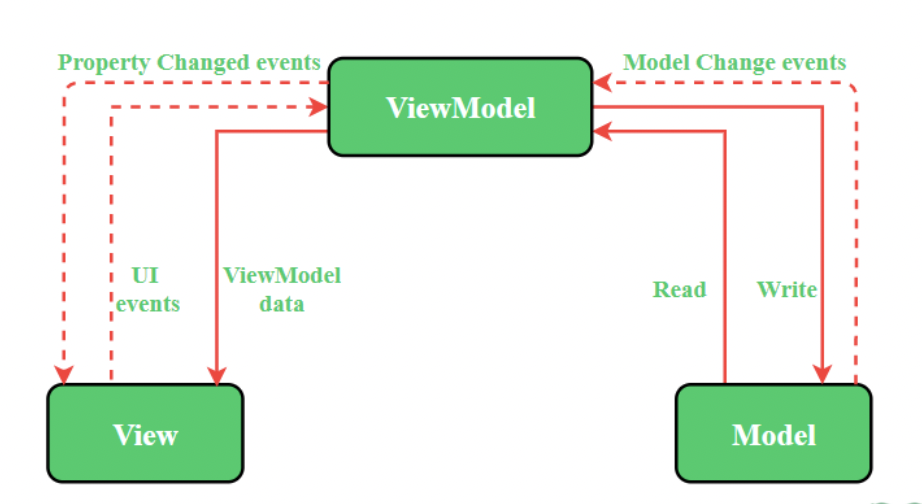
* UI(**View**) and data-access mechanism(**Model**) are tightly coupled.
* **Controller** and **View** exist with the one-to-many relationship. One Controller can select a different View based upon required operation.
* The **View** has no knowledge about the**Controller**.
* Difficult to make changes and modify the app features as the code layers are tightly coupled.
* User Inputs are handled by the **Controller**.

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



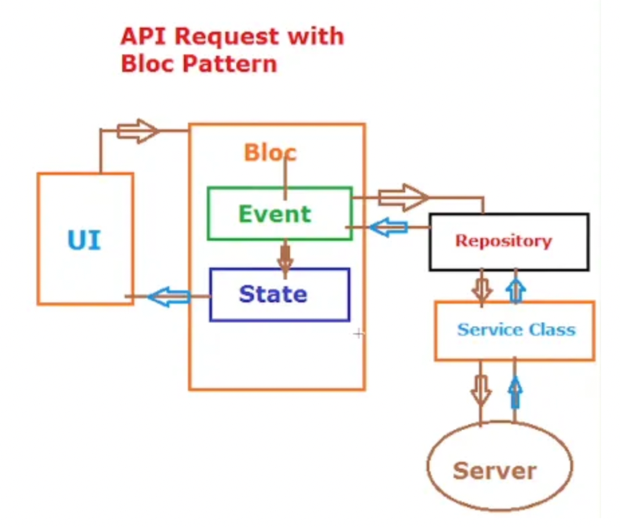
* It resolves the problem of having a dependent **View** by using **Presenter** as a communication channel between **Model** and **View**.
* The one-to-one relationship exists between **Presenter** and **View** as one Presenter class manages one View at a time.
* The **View** has references to the **Presenter**.
* Code layers are loosely coupled and thus it is easy to carry out modifications/changes in the application code.
* The **View** is the entry point to the application.

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

****

* This architecture pattern is more event-driven as it uses data binding and thus makes easy separation of **core business logic** from the **View**.
* Multiple **View** can be mapped with a single **ViewModel** and thus, the one-to-many relationship exists between View and ViewModel.
* The **View** has references to the **ViewModel**
* Easy to make changes in the application. However, if **data binding logic** is too complex, it will be a little harder to debug the application.
* The **View** takes the input from the user and acts as the entry point of the application.

**Bloc (Business Logic Component):**



* This pattern is a state management pattern that is commonly used in Flutter applications.
* First we have the UI, and from the UI we make request to the bloc.
* Bloc will have two things, event and the state. First when the UI connects to bloc it creates and triggers event.
* Event eventually call the repositories to the server through an endpoint.
* From the server now we get the data and it is pass back to the bloc. As we have the data we trigger the state.
* As we have the change in the state UI knows from the Bloc pattern and updates the UI.

<https://www.geeksforgeeks.org/difference-between-mvc-mvp-and-mvvm-architecture-pattern-in-android/>

<https://levelup.gitconnected.com/fetch-api-with-bloc-in-flutter-730b9e305c54>