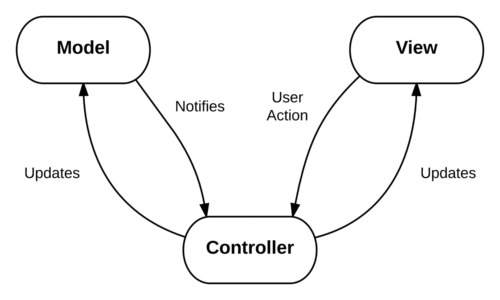
App Architectures

1. MVC

Model-view-controller (MVC) is the most popular design architecture in-app architecture design. It is the one Apple recommends and it’s easy to understand and use.



* The Model is where the App’s data resides. Things like persistence, model objects, parsers, managers, and networking code live there.
* The View layer is the face of the app. Its classes are often reusable as they don’t contain any domain-specific logic.
* The Controller mediates between the view and the model via the delegation pattern. it is the core component. It receives all the view actions, handles all user interaction logic, dispatches changes to the model, receives changes from the model and lastly applies those changes to the view.

The application starts by initializing the top-level view controllers. These view controllers then load and configure the views with the data from the model. The controller can either create its own models explicitly or access them through a singleton.

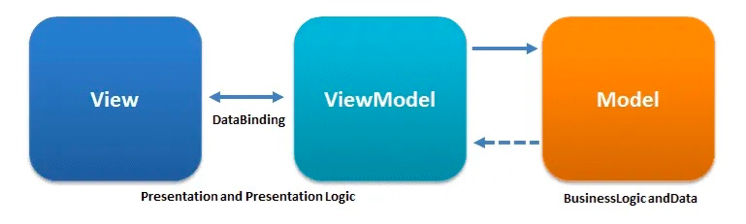
In this architecture, the controller will receive updates either through delegates or user interactions with the view. Views have no knowledge about their controllers. When an update or change occurs, the view controller will update the model or modify its own internal state.

View changes occur after the controller has updated the model. After the controller has updated the model, another function in the view controller is listening to changes to the model. When changes are detected to the view model, the controller will make the appropriate changes to the view itself. This is a one-directional flow of data which makes it easier to debug and manage this code later on.

The drawback of the MVC architecture is the view controller may become too complicated and it is difficult to perform test.

1. MVVM

Model-View-ViewModel separates a lot of the logic the view controller had to handle into a separate component called the view model. The view model is responsible for all the presentation logic. It is essentially an augmented version of the MVC architecture. This lets view controllers be smaller and more manageable. Due to its similarity with Model-View-Controller, MVVM can be easily incorporated into an existing MVC app.



MVVM initialization is very similar to MVC. A difference is that the view controller now also initializes a view model. The view controller then binds each view to relevant properties in the view model.

The models are also no longer directly owned by the view controllers and instead owned by the view models. The view controller receives updates or changes from user interactions or delegate calls, just like the MVC pattern. However, instead of the view controller changing its state or the model, it calls a function of the view model. The view model then either changes its internal presentation state or makes changes to the model. The view-model is responsible for listening to changes to the model. The view model handles model changes and updates its own internal state. The view controller is subscribed to the changes in the view model and will update the view accordingly when it detects those changes. Typically, these are done using a reactive programming framework or other observation mechanism. View states are either properties of the view themselves or stored in the view model. The view controller does not store any view state, unlike MVC.

1. MVP

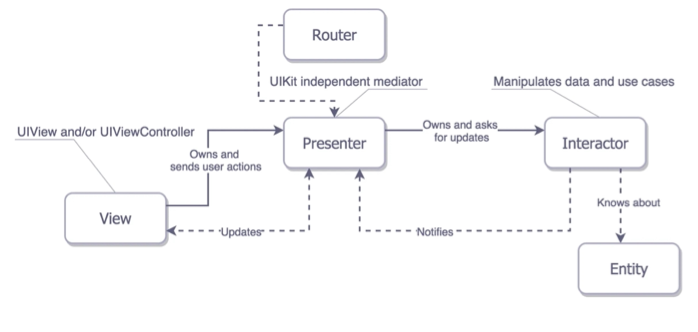
MVP is an architectural pattern, a derivation of the Model–View–Controller. It's represented by three distinct components: Model, View and the Presenter. It was engineered to facilitate automated unit testing and improve the separation of concerns in presentation logic.



1. Model is an interface responsible for the domain data (to be displayed or otherwise acted upon in the GUI), it is responsible for communicating with DB layer and raising appropriate events
2. View is responsible for the rendering of data and presenting to user.
3. Presenter is the "middle-man" between Model and View. It reacts to the user’s actions performed on the View, retrieves data from the Model, and formats it for display in the View
4. VIPER

VIPER is an adaption of Clean Architecture for iOS applications. It stands for View, Interactor, Presenter, Entity, and Routing. VIPER is designed to build components around different user cases.

VIPER is a hard pattern to start with as it requires many different components. Teams that use VIPER almost always use code generators as the amount of boiler plate code required to use this pattern is quite large.



1. View

The view displays what the presenter tells it to. It sends messages back to the presenter based on user interaction.

1. Interactor

The interactor contains business logic on a per use-case basis.

1. Presenter

This contains the view logic and view state. It changes the view based on messages sent from the interactor.

1. Entity

Entities are basic models that the interactor uses.

1. Routing

The router controls which view will be displayed and what the next view will be.