🐍VIPER Design Architecture

Viper:

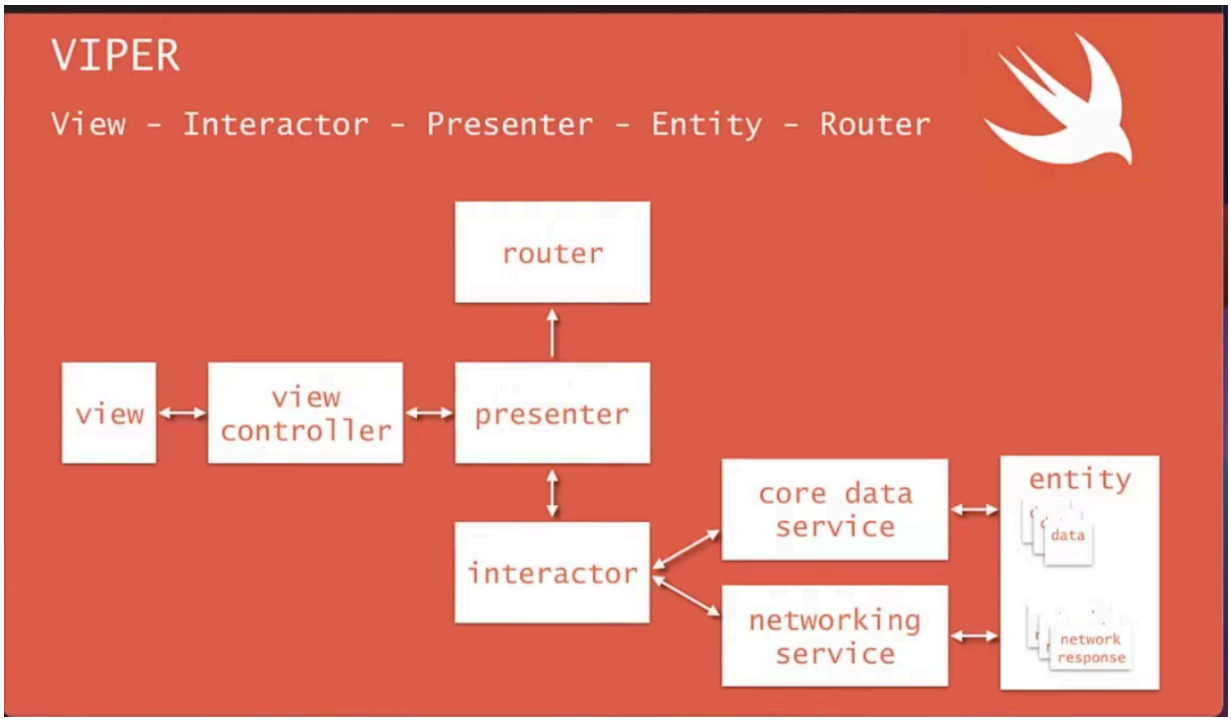
https://medium.com/swlh/viper-architecture-and-solid-principles-in-ios-96480cfe88b9
https://medium.com/@thanhlam_19048/viper-architecture-pattern-in-swift-3636545be812

Create IOS App with RxSwift + Viper :
https://medium.com/@vish_18086/creating-ios-app-using-viper-and-rxswift-8f6fc3475551

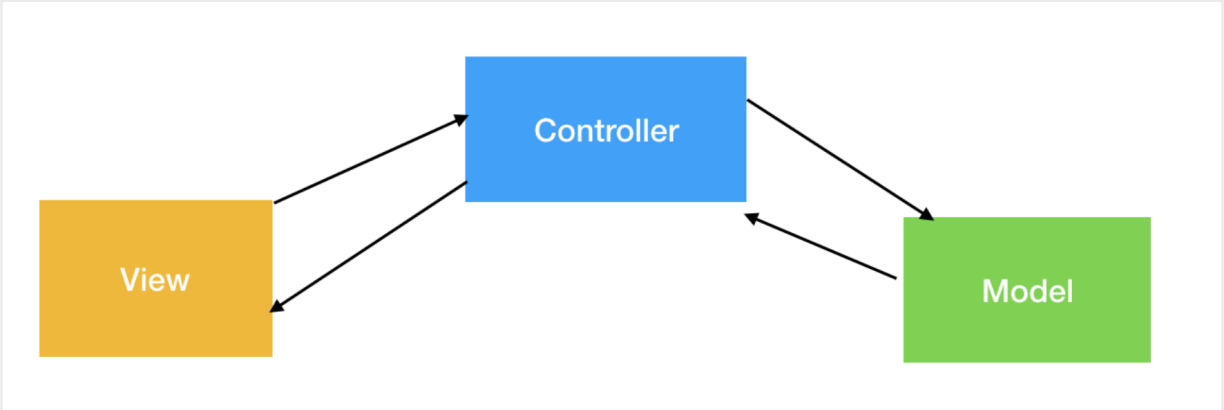
Create IOS App with RxSwift + MVVM:
https://medium.com/swlh/creating-your-first-rxswift-app-fe1e696c0528

MVVM Architecture:
https://medium.com/technology-nineleaps/mvvm-architecture-for-ios-6d78794e13d8

VIPER Architecture with SwiftUI:
https://www.raywenderlich.com/8440907-getting-started-with-the-viper-architecture-pattern



MVC - Model-View-Controller

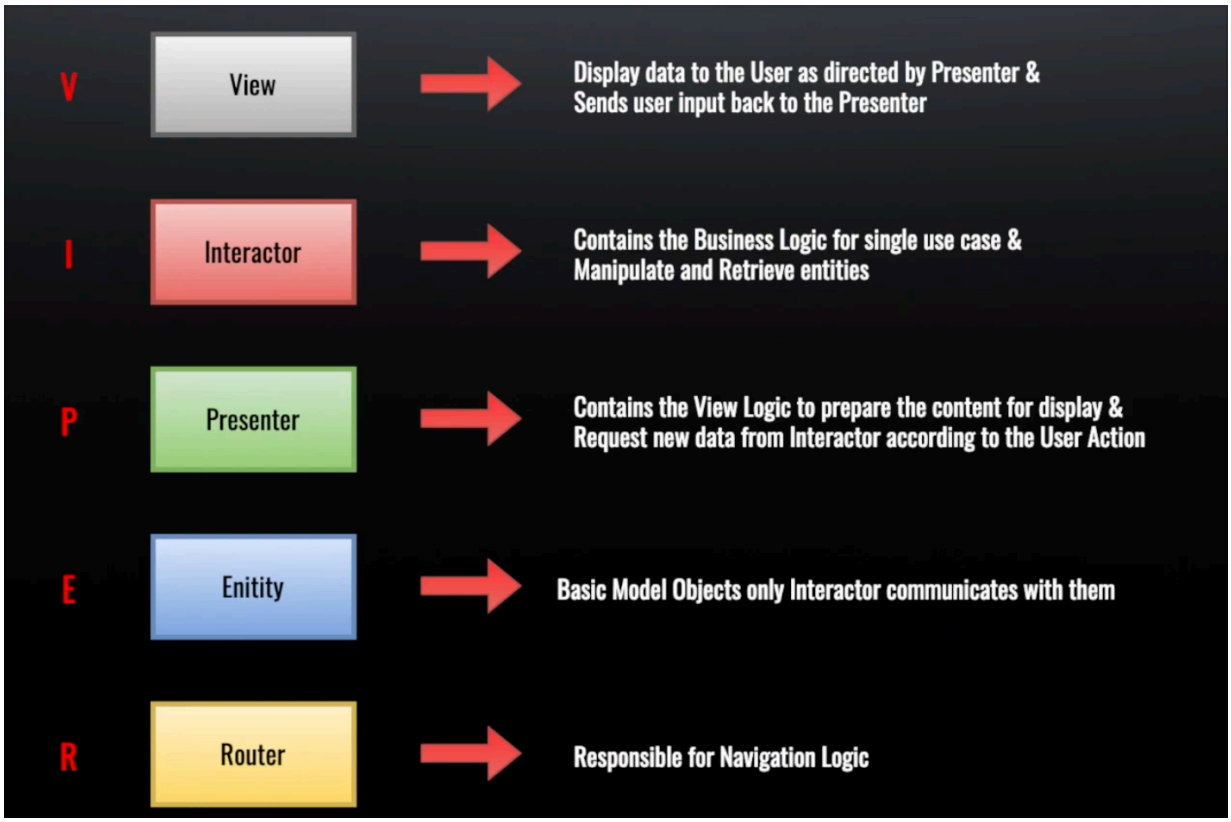


- View : All the user interface : views, containers, buttons, text fields and any other class that represents some component the user shall see.
- Model: entities, data model
- Controller:
  - manipulate data within the models and shows it through view
  - listen to user event with UI and reflects it into model layer
  - Controller also handle everything that is not View or Model such as API call, database query,...etc

=> A lot of methods, features and properties to be hold by a single class
=> A lot of testing, a lot of code to be understood by other developers and refactored

+ hard work to find out what is causing a specific bug.

- ✓ Viper architectures (View - Interactor - Presenter - Entity - Routing)
- It separates the content in five layers:



+ View :

- All the interface classes like views, buttons, text fields and switches, and also the controllers(UIViewController, UINavigationController, etc..)
- Show interface based on Presenter
- Accept input and hand it to Presenter

+ Presenter

- Takes care of the user events triggered by the View
- Sends messages to the business layer (Interactor) and

prepares the data from the use cases to be presented by the View.
- delegates a change of screen event to the Router layer.

+ Interactor

- The business rules layer.
- Operates with the application entities to make the use cases happen.
- All the logic features like login, logout, fetch data, delete date, create object and others may occur in the Interactor layer.

+ Model

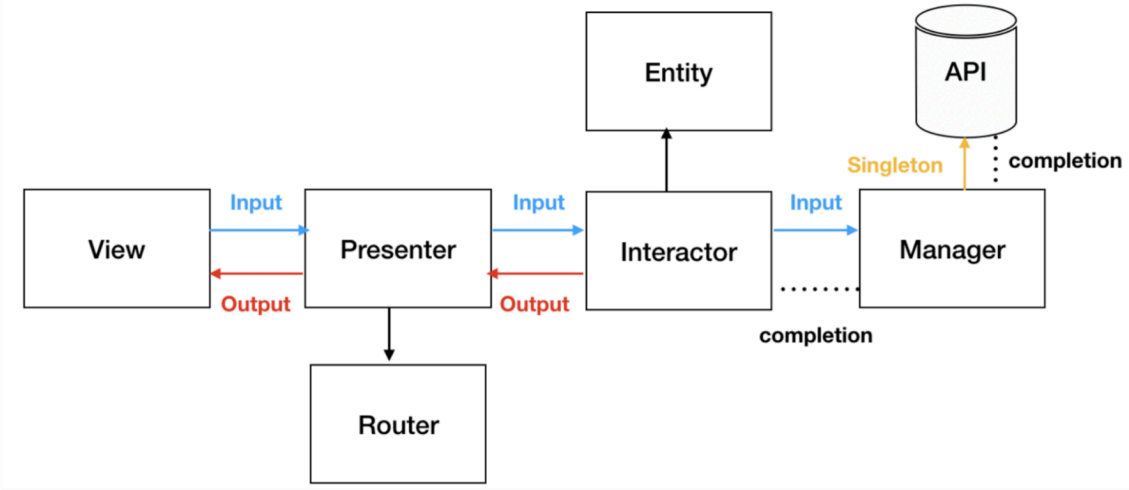
- all the entity classes and structs, being only an aggregation of data

+ Router

- responsible for the navigation of the app.
- it has access to the navigation controllers and windows, using them to instantiate other controllers to push into them.
- Communicates **only with the Presenter,**
  - and in some cases, with the view(when it wants to show a UI Alert)

+ Manager

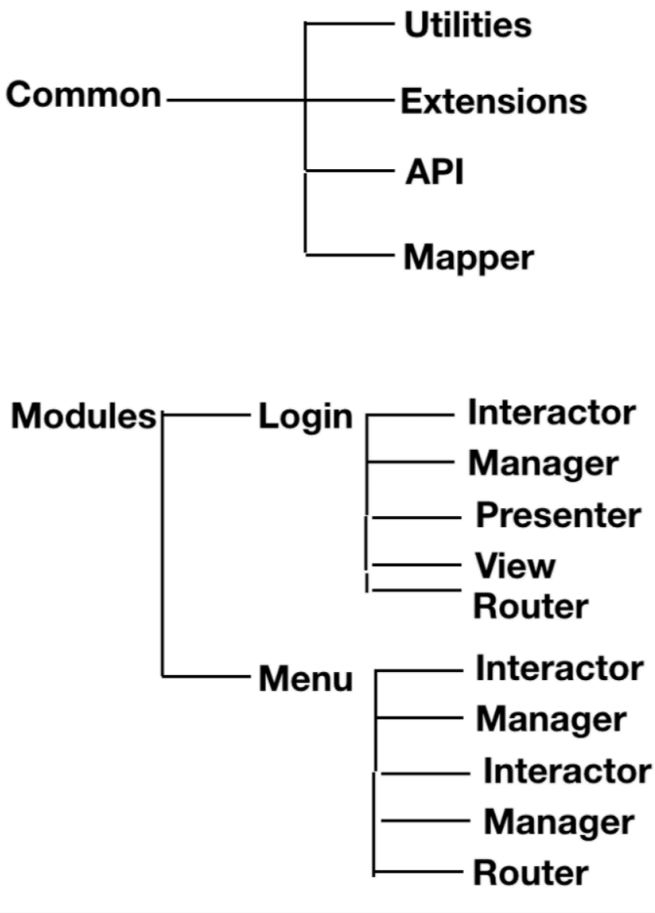
- Layer does not exist in the original Viper and Clean architectures
- Responsible for customizing request services to other data sources.
  - Example: Preparing headers and attributes for an API request, building a query to an SQL database or any other kind of data .
- It is triggered by the Interactor



VIPER diagram

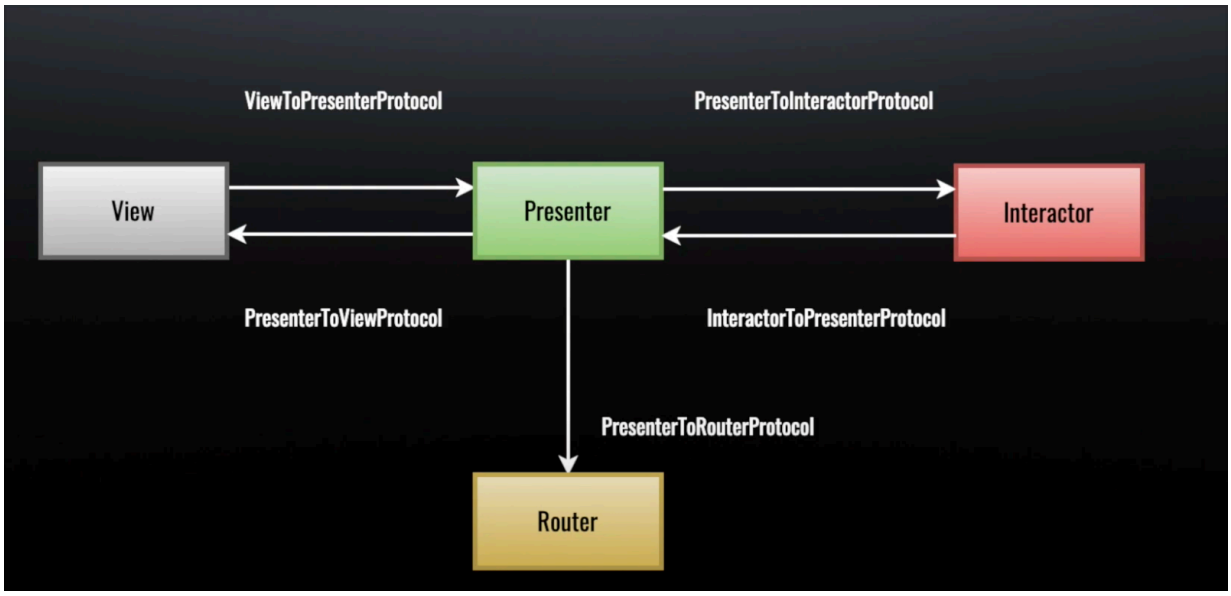
Each View Controller(**View** layer) has its own:

+ **Presenter**, which communicates with one or more **interactors**(each one corresponding to a use case)
+ **Router** to communicate with other modules
+ **Manager** if there is an API request or anything like that.



Template for a VIPER project in XCode

Các tầng liên kết với nhau (communications) thông qua protocols và delegates:



- **View → Presenter :** View tương tác với Presenter về tất cả những gì người dùng tương tác với View, bất cứ user input nào từ view sẽ được chuyển cho presenter.

Ví dụ : Người dùng đăng nhập, nhập vào username và password

- **Presenter → Interactor:** Sau khi tiếp nhận user input (là username và password) từ view thì Presenter sẽ chuyển tiếp cho Interactor để Interactor xử lý thông tin

- **Interactor → Entity**: Trong lúc xử lý thông tin nếu cần đến dữ liệu hoặc database thì Interactor sẽ gọi đến Entity để lấy dữ liệu (để check xem usename và password có đúng không chẳn hạn).

- **Entity → Interactor:** Entity sẽ gởi thông tin trả về cho Interactor về các moduls object hay các database, Entity có thể chứa các hàm để validate dữ liệu chẳn hạn.

- **Interactor — Presenter:** Sau khi tương tác đến Entity và xử lý xong dữ liệu thì Interactor sẽ output trả lại cho Presenter

Ví Dụ: Username và password hợp lệ, thực hiện thông báo đăng nhập thành công

- **Presenter → View:** Thực hiện các thay đổi về UI cho phù hợp (ở đây cụ thể là hiện đăng nhập thành công), gọi đến Router để chuyển màn hình nếu cần.

- **Presenter → Router**: Presenter gọi đến Router để điều khiển màn hình (tắt màn hình đăng nhập đi và hiển thị màn hình main chẳng hạn).