App Architecture Assignment

1. MVC: Model View Controller

- The user's interactions call methods on the Controller
- The Controller updates the Model, where the model is what actually stores state
- The Model exposes change listeners to notify observers that it has changed
- The View subscribes for changes emitted by the model to update itself

"When-to-use-it"(WTUI):

- When you have a very simple project like some iOS feature demo or testing an app feature in a Sample template
- When you have trivial business rules

2. MVP: Model View Presenter

- The user's interactions call methods on the Presenter
- The Presenter stores state, manages state, and calls methods on the view to notifyy it to update itself
- The Model is just data
- The View receives method calls from the presenter to be notified of changes to update itself

WTUI:

- There is a well-defined contract between Presenter and View, and it can be verified that under the right circumstances, the Presenter calls the right methods on the View.
- Great for unit testing
- Separating responsibilities among different components

3. VIPER: View Interactor Presenter Entity Routing

- The Interactor, where all the business rules are placed. This layer is UIKit independent.
- The Presenter, where the presentation logic resides. Always listening to user interactions in the ViewController handling events, and sometimes implementing some data source protocols or other UI components delegates. It's responsible for receiving the Interactor's use case outputs to translate it to a UI structure to be shown in the View. Presenter also decides when it's time for a change of context.
- The Router, also the Wireframe, is also like the Coordinator design pattern, but is a mandatory layer. Must be able to manage the screen context in multiple ways. Has access to the context ViewController or Nav Controller, so it can change it to another screen by pushing a new UIViewController instance to the stack or presenting it modally. Usually, there is a superclass that implements various operations for changing a simple context.
- The View is the ViewController with the UIView. There is no logic in here, so it's "dumb." Receives orders from the Presenter to render some content based on the VIewModel data structure. Always avoid if/else or loop statements in this layer
- The Entity resides only in a data model related to the screen's use cases. Only the Interactor has access to the entities. Usually simply data models, always structs, as value types.

WTUI:

- When you have a big project which relies on a lot of complex business rules and has a lot of screens or modules
- When not all the interactions in the UI trigger a use case, and so in that case, just the Presenter may attend, not discarding that business logic still exists
- When your team aims for a good testing framework making all the layers and responsibilities completely independent
- When you have different formats for data models, such as: an object for the API fetched data(Decodable), or one for manipulating business rules in the Interactor and another for presenting in the UI(ViewModel)

4. MVVM: Model View ViewModel

- The user's interactions call methods on the ViewModel
- The ViewModel stores state, magages state, and exposes change events when the state changes
- The Model is the data mapped into the observable fields, or any other form of event emissions
- The View subscribes for change events exposed by the viewmodel to update itself

WTUI:

- When your project has some simple business logic that demands some testing
- When there's not a huge number of screens
- When there are not too complex use cases
- When the app is still in its concept phase, where you don't know exactly what it will be like in a distant future.

5. Coordinator

- This is not actually an architecture, but a design pattern.
- Solves most of the problems in coupled architectures
- This maintains a reference to the scene's view controller
- It can push a new one to its navigation controller or maybe present a new screen modally
- Must access the ViewController to be able to change context

WTUI:

- Any time you want to separate navigation logic from the rest of the scene

6. Flux

- The object encapsulates an action, essentially the View emitting an event instead of calling a method on a presenter or viewmodel
- The dispatcher is an event queue, where actions are placed. It is similar to the contract that allowed the View to call methods on the presenter or viewmodel
- The store stores state and emits change events, simply the viewmodel, and also subscribes for the actions in the dispatcher
- The View observes state changes emitted by the store

WTUI:

- When views can create actions that will update many stores and stores can trigger changes that will update many views.
- In an app, where views don't map directly to domain stores.
- Actions need to be persisted and then replayed.