https://api.themoviedb.org/3/movie/popular?language=en-US&page=1&api\_key=6622998c4ceac172a976a1136b204df4

**## App Architecture Assignment**

Write a small excerpt on each of the above architectural patterns and discuss how they are different.  
Give a list of reasons why to use one over the other.

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

Model-view-controller is the most popular design architecture in-app architecture design. It is the one Apple recommends and it’s easy to understand and use. In most code examples on the official Apple or Swift websites, applications are built using the MVC architecture.

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.

View actions and delegates calls are saved as model changes

View controller listens to model changes

View controller updates view

**Model-View-ViewModel (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 our 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.

In MVVM, the view controller does not observe the model. Instead, 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

**Model-View-Controller-ViewState:**

The model-view-controller-viewstate architecture aims to build on MVC. It is similar to the MVVM pattern as well. However, the view controller still determines all model changes, unlike the view model in MVVM. It aims to separate out view state and make it more manageable.

The view controller observes the view state or the document view model and changes to the view occur through these observations.

**## Design Patterns Assignment**

Write a small paragraph about different design patterns. (1 paragraph per) Particularly Singleton, Factory, Facade, Decorator, and two others of your choice. Explain what they are and why/when they should be used.

Singleton:

**What they are and why:** A singleton class only allows one instance of itself to be created. This is a some-what disputed design pattern, as some describe it as an anti-pattern as it introduces global state into your application

**When they should be used:** should only use it when you are sure you only want one instance of a class created. Apple uses the Singleton pattern often in their code. For instance, to create singleton we use a static type of property. Static properties are guaranteed to be lazily initialized only once, even across multiple threads.

Example:

class SomeSingleton {

static let sharedInstace = SomeSingleton()

}

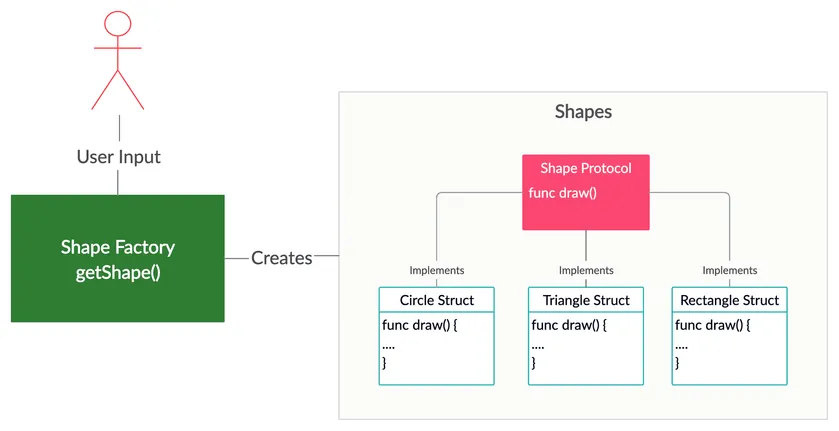
Façade:

**What they are and why:** A facade is a structural design pattern. The word facade means the “face of a building”. As the name suggests, the clients walking outside don’t know what’s inside the building. The complexities of the electrical, plumbing, elevators and stairways etc. The facade design pattern does the same, it hides the complexities of a system and displays a friendly face.

**When they should be used:** The facade design pattern is great to use when you want to simply the interface of a complex system. You should not use a facade if the system is already simple, as this adds an unnecessary layer of code.

Factory

**What they are and why:** The factory design pattern is a very popular pattern not only in Swift, but in many programming languages. The factory pattern allows you to create an object without exposing the creation details to the client. The object adheres to a common interface or in Swift, a common protocol.



**When they should be used:** When you don’t know what kind of object until runtime. Only use factories when you need to abstract away object creation. In a sense the factory method just keeps all your code organized in one struct.

Decorator:

**What they are and why:**

**When they should be used:**

Memento:

**What they are and why:**

**When they should be used:**

Iterator:

**What they are and why:**

**When they should be used:**