**## 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 architecture is focused on separating an application into three components. They are the model the views and the controllers. The model corresponds to all the data logic, the view handles all the UI logic and the controller interfaces between the two.

Model View ViewModel architecture is similar to the MVC architecture in that it splits the model from the UI Logic, where it differs is the entry point of MVC is the viewController, where as the View of MVVM further separates the UI from the business logic.

Model View Presenter is a sort of middle ground between mvvm and mvc. The presenter is still tightly coupled to the view rather then in mvvm where the view and view model are loosely coupled so they may work more independently of each other.

Viper is the next step from mvvm, it splits the viewmodel into pieces to support cleaner testing environments. The view is in charge of communicating between the user and the interactor. The interactor handles most of the business logic. Data from the interactor is processed by the presenter and sent back to the view. Entity is just the model layer from the other architectures but they renamed it for a cooler acronym, and the router stores the navigation logic.

**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.

The Singleton design pattern ensures that only one instance of a class is alive. Forbidding any other objects spawning from that class. This object is accessible from anywhere in the project but is never changeable. This makes it useful for accessing vulnerable data such as config files and hardware.

The Factory design pattern allows for the making of related objects without specifying their classes. This means that an overarching class still handles the methods we want it to but we can choose to override any methods that we deem necessary. This makes it excellent at adapting to new developments such as new clientele or new business requirements.

The Facade is a design pattern meant to uncomplicate some app dependencies caused by a third party library or framework. In the process it loses a lot of functionality from the framework but obtains a much simpler project that focuses on the needs of the app. It is also useful for breaking down those large complicated frameworks into layers. To gain the full benefits of facades, make all the client code communicate with the subsystem via facades that way any changes to the subsystem will not affect client code.

A Decorator design pattern allows increased functionality by wrapping objects and adding new functionality in each layer. By chaining these decorators together you can really increase the usefulness of an object without breaking it at runtime. It also helps to break down business logic into layers.

A Prototype is blueprint of an existing object that is not reliant on its classes. This is useful for when you need many similar objects that do slightly different things but are unable, due to likely having private fields, to be copied by simply creating a new object. Its good to have on hand when you want to get rid of a bunch of subclasses that only differ in how they initialize.

The Flyweight design pattern is a very useful tool to keep ram usage down. It achieves greater performance by cutting down on redundant data usage and allowing objects to share common parts of their states rather then keeping each stating in each object. This allows for many many instancances of the same object to fit in rather finite amounts of ram.