# Introducing the presentation pattern

Chosing a presentation pattern for your application dependes on the APIs for the development platform and the developer’s personal preference.



fig 01. Most common presentation patterns for mobile applications

The benefits to selecting a well-known architectural style are several. Separating the presentation code from the domain/logic can help developers maintain, and scale and test their code.

* They give the developer a selection of tried and tested solutions to work with
* They are language neutral and so can be applied to any language that supports object-orientation
* They aid communication by the very fact that they are well documented and can be researched if that is not the case.
* They have a proven track record as they are already widely used and thus reduce the technical risk to the project
* They are highly flexible and can be used in practically any type of application.



*fig 02. Benefits of architectural style.*

# Model-View-Presenter (MVP)

The Model-View-Presenter is a separate presentation pattern where we include a *presenter* that works with views that manage their own behavior (e.g. Controls)



*fig 03. Diagram of “how” work the MVP pattern.*

* ***The model*** is where your business logic and application data is stored. Not only Model is responsible for storing your application data, it also consists of components that holds responsibilities for generating, exposing and fetching the data.
* ***The view*** is a passive interface that displays data (the model) and routes user commands (events) to the presenter to act upon that data.
* ***The presenter*** is the middleman or mediator between View and Model which hold responsibilities of everything which has to deal with presentation logic in your application.

Many popular framework are designed to use the Model-View-Presenter pattern to separate business logic from UI.



*fig 04. Common framework*

# MVP - Passive View

The Model-View-Presenter variants, Passive View and Supervising Controller, specify different approaches to implementing view updates.

In Passive View, the interaction between the view and model is always done through the presenter, this makes the UI completely testable through the presenter.



*fig 05. Diagram of “how” works MVP Passive View.*

# MVP – Supervising Controller

In supervising controller, the view interacts directly with the model without the presenter being involved, the presenter updates the model and the model then pushes those changes directly to the view,



*fig 06. Diagram of “how” works MVP supervising controller*

## Which one use?

The decision to use Passive View or Supervising Controller primarily depends on how testable you want your application to be. If testability is a primary concern in your application, Passive View might be more suitable because you can test all the UI logic by testing the presenter. On the other hand, if you prefer code simplicity over full testability, Supervising Controller might be a better option because, for simple UI changes, you do not have to include code in the presenter that updates the view. When choosing between Passive View and Supervising Controller, consider the following:

* Both variants allow you to increase the testability of your presentation logic.
* Passive View usually provides a larger testing surface than Supervising Controller because all the view update logic is placed in the presenter.
* Supervising Controller typically requires less code than Passive View because the presenter does not perform simple view updates.