# Basic Exercises Part 5.1. Design Patterns. MVC.

## MVC Introduction

* MVC – short for Model-View-Controller – is Apple’s preferred way of architecting apps for its platforms, and so it’s the default approach used by most developers on Apple platforms.
* What is MVC?

The Model View Controller (MVC) Pattern is a design pattern most commonly used for creating user interfaces. The major advantage of MVC is that it separates:

* + The internal representation of the application state (the Model). Models store your data, such as the names of products in a store.
  + How the information is presented to the user (the View), Views render data for users, for example a table showing the list of products available.
  + The logic which controls how the user interacts with the application state (the Controller). Controllers combine the two, by querying the model and converting its data to something views can show.
* Use cases of MVC
  + The primary use case for MVC is in Graphical User Interface (GUI) programming.
  + The View component listens to the Model component for changes.
  + The Model acts as a broadcaster; when there is a change mode to the Model, it broadcasts its changes to the View and the Controller.
  + The Controller is used by the View to modify the Model Component.
* Graphic definition:

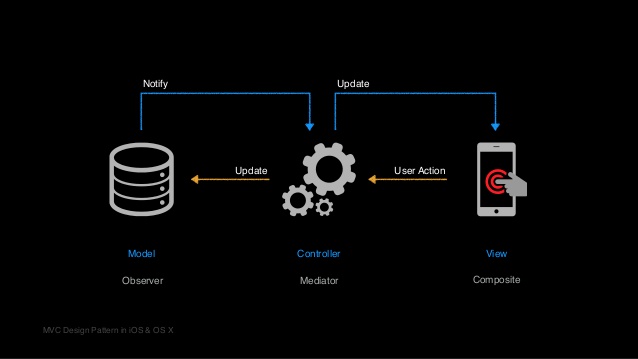


Fig 1. Source (<https://www.slideshare.net/nmdiaspt/ios-apps-in-swift>)

* How the messages flow? This is happens in 4 ways:
  + The *View* tells the *Controller* when a user interaction takes place (step #1).
  + The *Controller* updates the *View* when the data changes (step #4, in the future).
  + The *Controller* updates the *Model* when the data changes (step #2).
  + The *Model* notifies the *Controller* when data changes (step #3).
* Some examples:
  + Objects like UIButton, UIView and UILabel are all examples of Views. More complex views like MKMapView contain many visual properties like mapType and isZoomEnabled.
  + Objects like UIViewController, CLLocationManager and UINavigationController are examples of Controllers. In iOS you also use delegates, a special kind of controller that you can hand-off messages to.
  + When using the Core Data SDK, your .xcdatamodeld file contains your app’s Models. In Realm, these models are created as simple classes. In Firebase and Parse Servers these models are represented as JSON objects.
* A real scenario. Your “To-do App”.

In your To-Do app, you have a number of Views in a list, one for every task (actually is a cell). Every View is backed by a Model that has the data for that task, like ‘title’ and ‘isCompleted’. Now the following happens:

* The user taps on a task *View*, and this causes the iPhone on-screen keyboard to pop up – the task title can now be edited.
* The user edits the task title, like changing it to "Get groceries", and taps the *Submit* button.
* The *Controller* now responds to the data change and updates the *Model* accordingly.

Likewise, this can happen too:

* The *Model* data for a task changes, because of data that comes in from a cloud back-end like Firebase.
* The *Model* then updates the *Controller*, sending a message that its data has changed.
* The *Controller* will then propagate that message and update the *View* accordingly.

# Summary

Don’t let your models talk to your views. Use views to display content to the user, use models to store data, and use controllers to handle any intermediary logic between the two.

MVC is a good starting point for every app you write. Yes, there are other architecture patterns that you can use but this is the easiest to use while starting off.

## MVC

### **1.1 How Apple does**

Visit:

<https://developer.apple.com/library/archive/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html>

### **1.2 Basic implementation**

Create a new single application. Open the Library (also: Shift + Cmd + L) and search for: View Controller. Click and drag it onto your view. Search for it and then drag it onto your view (or use the default one).

### **1.3 Create the folder structure based on MVC**

Additionally, add four labels in the storyboard.

### **1.4 Storyboard ID**

Use next snippets on your code.

The model:

// Model

**class** Dog {

**var** name: String?

**var** legs: Int?

}

The controller:

// Controller

**class** ViewController: UIViewController {

// View from the storyboard - both labels

**@IBOutlet** **var** petName: UILabel!

**@IBOutlet** **var** petLegs: UILabel!

// Controller continued

**override** **func** viewDidLoad() {

**super**.viewDidLoad()

**let** fido = Dog()

fido.name = "Fido"

fido.legs = 4

petName.text = fido.name!

petLegs.text = "\(fido.legs!)"

}

}

The View

A screenshot of a cell phone

Description automatically generated

Build and run.

Expectation: create the folder structure for your project.

**1.5 Switch language**

We always need to try the same behavior on Objective C.