Apple's MVC pattern

The MVC stands for Model, View and Controller. The Model encapsulates the data specific to an application and defines the logic and computation that manipulate and process that data. The View is an object in an application that users can see. A view object knows how to draw itself and can respond to user actions. A major purpose of view objects is to display data from the application’s model objects and to enable the editing of that data. The Controller acts as an intermediary between one or more of an application’s view objects and one or more of its model objects. Controller objects are thus a conduit through which view objects learn about changes in model objects and vice versa. Controller objects can also perform setup and coordinating tasks for an application and manage the life cycles of other objects.

This pattern is not perfect, but it is not the worst. Here are some pros of Apple’s MVC pattern, easy to use which means compared with other modes is the smallest amount of code and easier to maintain even for less experienced developers. Although it has advantages, it also has cons such as poor testability and also a more bulky controller.

In my opinion Apple’s MVC pattern is a great architecture procedure because as a developer it is easy to apply in my development applications and even though it has disadvantages that it does not really affect me as a developer because it only affects the performance of the app.



Model: User actions in the view layer that create or modify data are communicated through a controller object and result in the creation or updating of a model object. When a model object changes (for example, new data is received over a network connection), it notifies a controller object, which updates the appropriate view objects.

View: learn about changes in model data through the application’s controller objects and communicate user-initiated changes—for example, text entered in a text field—through controller objects to an application’s model objects.

Controller: Interprets user actions made in view objects and communicates new or changed data to the model layer. When model objects change, a controller object communicates that new model data to the view objects so that they can display it.