# The Model View Framework (MVC)

In the web development world, you often hear people talking about the MVC framework and the MVVM framework and so on. So, what exactly are these frameworks? How are they useful in doing web development?

Let's talk about that briefly next. In the software engineering world, you often hear people talking about design patterns. What exactly they mean is to stop reinventing the wheel every single time. Design pattern is well-documented solution to a recurring problem. Very often, you see yourself repeatedly solving similar problems. If we have a well-specified documentation of how to solve these problems, why keep reinventing the wheel every single time? So that is where the software engineering's design pattern concept originates. Sometimes, you also see people referring to this as an architecture pattern.

So, to summarize, software design patents in particular are a reusable solution to commonly occurring problems that are solved in software. Now in this context, you often hear people talking about the Gang of four. This was a group of four authors that wrote this seminal book called Design Patterns, Elements of Reusable Object-Oriented Software. In this book, they identified a large set of commonly used design patterns in software engineering. This was one of the first well-documented exploration of design patterns and hence, became the gold standard for anybody working in software engineering, especially concerned about design patterns.

This software engineering pattern enables us to isolate domain logic from the user interface. So, you're basically separating the users view of the information from the actual logic and how the information is stored and manipulated. Now, the separation of concerns concepts. You going to be hearing over and over again in this context. The separation of concerns is what facilitates independent development of each of these three parts of our application, and also enables testing and maintenance of these different parts.

Now, we can divide our entire application into three parts. The View, that is primarily concerned with presenting the information to the user. The Model, that stores the domain state and the domain logic and also provides the way of manipulating this state from the rest of the application, and the Controller that mediates between the view and the model. We'll talk about each of these three parts in a bit more detail next.

In the MVC framework, the model manages the behavior and data of the application domain. And the model responds to requests for information about its current state. So, typically when the view wants to render, or the view wants to update itself, it might query the model in order to obtain information, so that it can be rendered appropriately to the user. The model also will respond the requests for change of its state. This is usually done through the Controller. In an event-driven system, the model also can be configured to notify observers.

So, viewers can register themselves as observers for the model and so when the model is updated, the views will be automatically triggered to update themselves based on the change to the model state. The view itself, is concerned with presenting the information to the users in a user interface element. In such a way that it facilitates both the presentation of information to the user and also enables the user to interact with the application. So, the view may represent one representation of the model state. So, from a single model, you can easily drive multiple ways of presenting this information to the user. For example, depending upon the viewport size. So, a small size viewport like on a mobile application, the information may be presented in a different way as opposed to a larger viewport that is facilitated on a desktop computer.

So, in an MVC framework, all the display of formation has a one-to-one correspondence with their model state. The third piece of puzzle in the MVC framework, is the controller. The job of the controller is to receive information from that view. So, any user interaction that is formed will be captured and then passed on to the controller in order to act on these user interactions. And it is the job of the controller then to initiate a change of the state of the model if it is required in this particular situation.

So, the controller will appropriately cost the change of the state of the model. So, to summarize, the controller can accept input from the user in terms of the user interactions that have taken place, and then it will instruct the model to change the state. Simultaneously, the controllers may also cause the view to change the way information has been shown in the view. So that is the reason why in this picture, you have two arrows going from the controller. One towards the model and the other towards the view. Sometimes, you hear people talking about the Model View View-Model approach.

The Model View View-Model approach is in some sense, a derivative of the Model-View-Controller approach. Sometimes, you also hear people referring to it as the Model-View-Binder approach. In here, you have the model that represents the business logic and the data for your application. From the model, you derive a View-Model, which encapsulates that part of the information that is required for rendering a specific view. So, the View-Model is the abstraction of the view, that exposes the public properties and the various comments that are available. So, this provides a declarative data binding. With this quick understanding of the MVC and MVVM framework, let's now proceed forward to learn more about React.