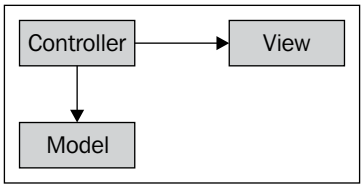
In terms of ASP.NET web applications, the model, view, and controller participants can be identified as:

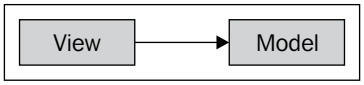
- **View:** This refers to HTML markup in ASPX pages, minus the code-behind logic. This view is rendered in the presentation tier (the browser).
- **Controller:** This refers to the special controller classes that decide which model needs to be shown to which particular view.
- **Model:** This refers to the data coming from the data layer, which may be processed by the business layer.

Before moving ahead, an important point to understand is that the MVC design is not a replacement to the n-tier architecture. MVC is more focused on how to keep the UI separate from the logic and the model; the model itself can be broken into separate tiers.

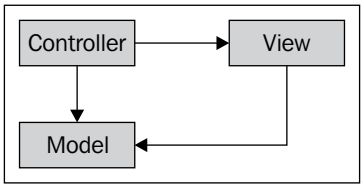
In the MVC design, the model, the view, and the controller are not related directly to the layers, or to the physical tiers; they are logical components that operate together in a certain pattern. The controller is related directly to the model and the view. Based on user actions (in the view), it fetches the data (the model) and populates the view. The relationship between the controller, the model, and the view can be depicted as:



The view is based on the model, which means that its job is to simply render the model that the controller passes to it:



So the net relationship between the three components can be described as:



A few important points to note from the above diagram:

- We can see that the model depends neither on the view nor on the controller, which is logical. Think of it like this: we have some data in the database tables; we use DAL code to handle this data and BL code to operate on this data as per certain business rules. Now, it is up to the UI to present and show this data. But the data itself is not dependent on the graphical user interface (GUI). So the model is independent of the view and the controller.
- The view does not depend on the controller; rather, the controller is associated with the view. That means we have a separation between the view and the controller, allowing us to change views independent of the controller.
- The view depends on the model, and is updated when the model's state has changed. As the view cannot contain any logic (which is stored inside the model), the view depends on the model; that is, the model is in charge of updating the contents or displaying the view.

Now we will look at the practical aspects of implementing this MVC design using the ASP.NET MVC framework, which will help us implement our web applications. MVC will be ready in no time. But before going ahead with the actual code, we need to understand another important aspect of ASP.NET MVC framework, that is, REST!

REST: Representation State Transfer

REST means Representational State Transfer, an architectural pattern used to identify and fetch resources from networked systems such as the World Wide Web (WWW). The REST architecture was the foundation of World Wide Web. But the term itself came into being around the year 2000, and is quite a buzzword these days. The core principle of REST is to facilitate the sharing of resources via unique identifiers, just as we use Uniform Resource Identifiers (URIs) while accessing resources on the Web. In simple terms, REST specifies how resources should be addressed, including URI formats, and protocols such as HTTP. The term resources include files such as ASPX pages, HTML files, images, videos, and so on.

In the default page controller based design in ASP.NET, we don't follow a strict REST-based architecture. If we use a pure REST-based architecture, then all of the information required to access a particular resource would be in the URI. This means that we don't need to check if a postback happened or not, because each request is unique in itself and will be treated differently (via unique URLs). Whereas in ASP.NET, we can use the postback technique to make the same requests using the same URLs and do different processing based on whether it is a postback or not. Many a times, in numerous projects, we come across the following coding style in ASP.NET code-behind files: