5
Model View Controller

These days, **Model View Controller** (**MVC**) is a buzzword in the ASP.NET community, thanks to the upcoming ASP.NET MVC framework that Microsoft is expected to launch soon (at the time of writing of this book, only Preview 5 was available). This chapter is dedicated to MVC design and the ASP.NET MVC framework.

In this chapter, we will learn about MVC design patterns, and how Microsoft has made our lives easier by creating the ASP.NET MVC framework for easier adoption of MVC patterns in our web applications. The following are some highlights of this chapter:

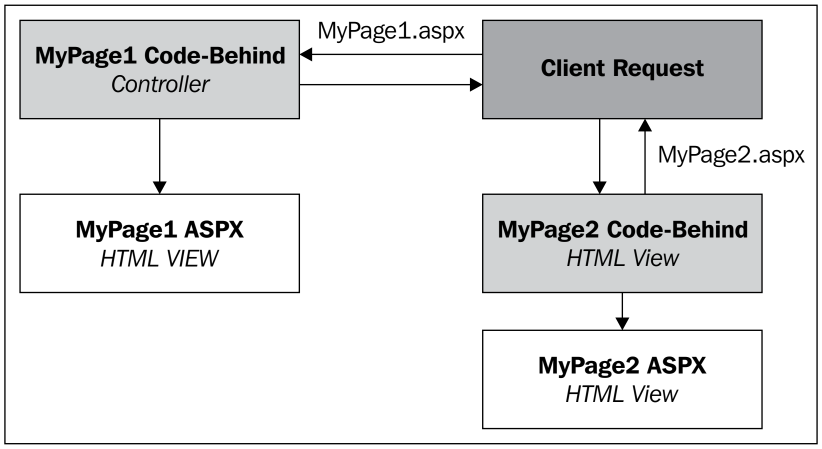
- Understanding the Page Controller pattern
- Understanding the need for the MVC design pattern
- Learning the basics of MVC design
- Understanding the Front Controller design pattern
- Understanding REST architecture
- Understanding the ASP.NET MVC framework
- Implementing the ASP.NET MVC framework in a sample application

Page Controller Pattern in ASP.NET

So far, all web pages we have created in our coding samples are based on the page controller pattern, which is the default architecture in the ASP.NET web forms. Let us understand page controller in detail.

In Chapter 2, we noticed that inline coding samples in ASP and ASP.NET had HTML and code scripts mixed together, creating a hard-to-maintain code base. Then we studied how code-behind classes "modularized" the architecture by separating the logic from the HTML. This code-behind architecture is a page controller based design, where by controller we mean the components that control the rendering of the HTML, which in the case of ASP.NET web forms are the code-behind classes.

Each page has a code-behind class, and the URL requested by the client is directly handled by individual pages. Any button or server control causing postbacks (such as a DropDownList control) is handled directly by the page code-behind class. So understanding the page life cycle is very important in a page controller based architecture. Here is a diagram that shows how a page controller pattern works in ASP.NET:



So for every page, its code-behind will act as a controller and handle all requests, and return processed HTML to the client browser.

Problems with Page Controller Design

In the page controller design we have a controller for each distinct page in our application (a separate code-behind class having all of the logic that fires sequentially as each page loads according to the ASP.NET page life cycle). So for big projects, there could potentially be a lot of code in the code-behind files, creating problems in code maintenance and support.

GUI Unit Testing

Separating business logic and data access code from the GUI is one of the steps leading towards a better design. In the previous chapters, we saw how to implement a basic n-tier architecture using tiers and layers to achieve loose coupling. But testing the application, especially the GUI and the code-behind classes in a page controller based model, is very difficult because the only way to test something like a button click's code-behind event handler is to click the button itself! This means that if we put more and more code in code-behind classes (which inevitably becomes the case in large web applications with lots of UI controls),