CHAPTER 1

Putting ASP.NET MVC in Context

ASP.NET MVC is a Web development framework from Microsoft that combines the effectiveness and tidiness of model-view-controller (MVC) architecture, the most up-to-date ideas and techniques from agile development, and the best parts of the existing ASP.NET platform. It is a complete alternative to traditional ASP.NET Web Forms, delivering advantages for all but the most trivial of Web development projects. In this chapter, you'll learn why Microsoft created ASP.NET MVC, how it compares to its predecessors and alternatives, and, finally, what's new in ASP.NET MVC 5 and what's covered in this book.

Understanding the History of ASP.NET

ASP.NET was a huge shift when it first arrived in 2002. Figure 1-1 illustrates Microsoft's technology stack as it appeared then.

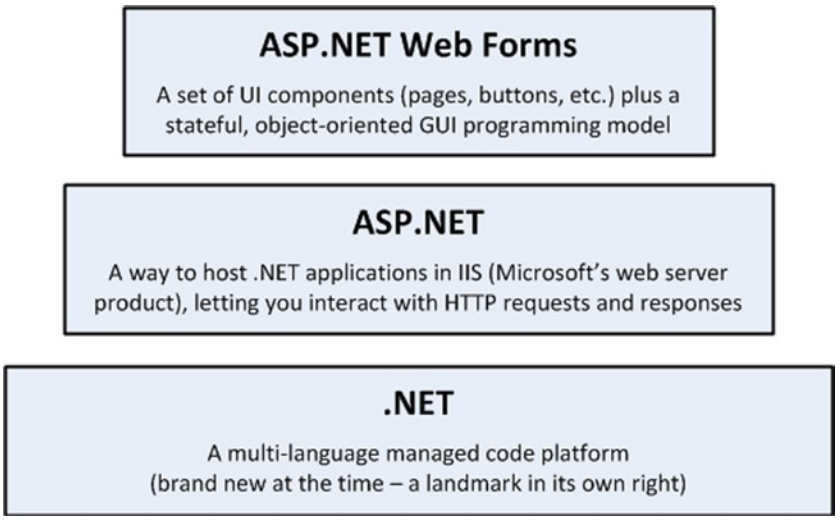

Figure 1-1. *The ASP.NET Web Forms technology stack*

With Web Forms, Microsoft attempted to hide both HTTP (with its intrinsic statelessness) and HTML (which at the time was unfamiliar to many developers) by modeling the user interface (UI) as a hierarchy of server-side control objects. Each control kept track of its own state across requests (using the *View State* facility), rendering itself as HTML when needed and automatically connecting client-side events (for example, a button click) with the corresponding server-side event handler code. In effect, Web Forms is a giant abstraction layer designed to deliver a classic event-driven graphical user interface (GUI) over the Web.

The idea was to make Web development feel just the same as Windows Forms development. Developers didn't need to work with a series of independent HTTP requests and responses. They could think in terms of a stateful UI, and Microsoft could seamlessly transition the army of Windows desktop developers into the new world of web applications.

What Is Wrong with ASP.NET Web Forms?

Traditional ASP.NET Web Forms development was great in principle, but reality proved more complicated:

- *View State weight*: The actual mechanism for maintaining state across requests (known as View State) results in large blocks of data being transferred between the client and server. This data can reach hundreds of kilobytes in even modest Web applications, and it goes back and forth with *every* request, leading to slower response times and increasing the bandwidth demands of the server.
- *Page life cycle*: The mechanism for connecting client-side events with server-side event handler code, part of the page life cycle, can be extraordinarily complicated and delicate. Few developers have success manipulating the control hierarchy at runtime without getting View State errors or finding that some event handlers mysteriously fail to execute.
- *False sense of separation of concerns*: ASP.NET Web Forms' *code-behind* model provides a means to take application code out of its HTML markup and into a separate code-behind class. This has been widely applauded for separating logic and presentation, but, in reality, developers are encouraged to mix presentation code (for example, manipulating the server-side control tree) with their application logic (for example, manipulating database data) in these same monstrous code-behind classes. The end result can be fragile and unintelligible.
- *Limited control over HTML*: Server controls render themselves as HTML, but not necessarily the HTML you want. In early versions of ASP.NET, the HTML output failed to meet with Web standards or make good use of Cascading Style Sheets (CSS), and server controls generated unpredictable and complex ID attribute values that are hard to access using JavaScript. These problems are much improved in recent Web Forms releases, but it can still be tricky to get the HTML you expect.
- *Leaky abstraction*: Web Forms tries to hide HTML and HTTP wherever possible. As you try to implement custom behaviors, you frequently fall out of the abstraction, which forces you to reverse-engineer the postback event mechanism or perform obtuse acts to make it generate the desired HTML. Plus, all this abstraction can act as a frustrating barrier for competent Web developers.
- *Low testability*: The designers of Web Forms could not have anticipated that automated testing would become an essential component of software development. Not surprisingly, the tightly coupled architecture they designed is unsuitable for unit testing. Integration testing can be a challenge, too.

Web Forms isn't all bad and Microsoft has put a lot of effort into improving standards compliance, simplifying the development process, and even taking some features from ASP.NET MVC. Web Forms excels when you need quick results, and you can have a reasonably complex web app up and running within a day. But unless you are careful during development, you will find that the application you create is hard to test and hard to maintain.

■ **Note** For complete details of ASP.NET Web Forms, see my *Pro ASP.NET 4.5 in C#* book, also published by Apress. I cover the complete framework and provide best-practice guidance for avoiding the most serious pitfalls.

Web Development Today

Outside Microsoft, Web development technology has been progressing rapidly and in several different directions since Web Forms was first released.

Web Standards and REST

The drive for Web standards compliance has increased in recent years. Web sites are consumed on a greater variety of devices and browsers than ever before, and Web standards (HTML, CSS, JavaScript, and so forth) remain the great hope for enjoying a consistent browsing experience. Modern web platforms can't afford to ignore the business case and the weight of developer enthusiasm for Web standards compliance.

HTML5 has begun to enter mainstream use and provides the Web developer with rich capabilities that allow the client to perform work that was previously the exclusive responsibility of the server. These new capabilities and the increasing maturity of JavaScript libraries such as AngularJS, jQuery, jQuery UI, and jQuery Mobile means that standards have become ever more important and form the critical foundation for ever richer Web apps.

■ **Note** I touch on HTML5, jQuery, and its cousins in this book, but I don't go into depth because these are topics in their own right. If you want more complete coverage, then Apress publishes my books on these subjects: *Pro AngularJS*, *Pro jQuery 2.0*, *Pro JavaScript for Web Apps,* and *The Definitive Guide to HTML5*.

At the same time, Representational State Transfer (REST has become the dominant architecture for application interoperability over HTTP, completely overshadowing SOAP (the technology behind ASP.NET's original approach to Web services). REST describes an application in terms of resources (URIs) representing real-world entities and standard operations (HTTP methods) representing available operations on those resources. For example, you might `PUT` a new `http://www.example.com/Products/Lawnmower` or `DELETE` `http://www.example.com/Customers/Arnold-Smith`.

Today's Web applications don't serve just HTML. Often, they must also serve JSON or XML data to client technologies such as AJAX and native smartphone applications. This happens naturally with REST, which eliminates the distinction between Web services and Web applications, but requires an approach to HTTP and URL handling that has not easily been supported by ASP.NET Web Forms.

Agile and Test-Driven Development

It is not just Web development that has matured. Software development as a whole has shifted toward *agile methodologies*. This can mean a lot of different things, but it is largely about running software projects as adaptable processes of discovery and resisting excessive forward planning. Enthusiasm for agile methodologies tends to go hand-in-hand with a set of development practices and tools (usually open source) that promote and assist these practices.

Test-driven development (TDD), and its close relative, *behavior-driven development (BDD)*, are two examples. The idea is to design your software by first describing examples of desired behaviors (known as *tests* or *specifications*), so at any time you can verify the stability and correctness of your application by executing your suite of tests against the implementation. There's no shortage of .NET tools to support TDD/BDD, but these tend to not work well with Web Forms:

- *Unit testing tools* let you specify the behavior of individual classes or other small code units in isolation. These can be effectively applied only to software that has been designed as a set of independent modules, so that each test can be run in isolation. Unfortunately, few Web Forms applications can be tested this way.
- *UI automation tools* let you simulate a series of user interactions against a complete running instance of your application. These can be used with Web Forms, but they can break down whenever you make a slight change to your page layout. Without special attention, Web Forms change the HTML structures and element IDs, breaking test suites.

The .NET open source and independent software vendor (ISV community has produced no end of top quality unit testing frameworks (NUnit and xUnit), mocking frameworks (Moq and Rhino Mocks), inversion-of-control containers (Ninject and AutoFac), continuous integration servers (Cruise Control and TeamCity), object-relational mappers (NHibernate and Subsonic), and the like. Traditional ASP.NET Web Forms is not amenable to these tools and techniques because of its monolithic design, and so Web Forms gets little respect from these projects.

Ruby on Rails

In 2004, Ruby on Rails was a quiet, open source contribution from an unknown player. Suddenly fame hit, transforming the rules of Web development. It's not that Ruby on Rails contained revolutionary technology but that the concept took existing ingredients and blended them in such a compelling and appealing way as to put existing platforms to shame.

Ruby on Rails (or just Rails, as it is commonly called) embraced an MVC architecture, which I describe in Chapter 3. By applying MVC and working in tune with the HTTP protocol, by promoting conventions instead of the need for configuration, and by integrating an object-relational mapping (ORM tool into its core, Rails applications more or less fell into place without much effort. It was as if this was how Web development should have been all along. Rails showed that Web standards compliance and RESTfulness don't need to be hard. It also showed that agile development and TDD work best when the framework is designed to support them. The rest of the Web development world has been catching up ever since.

Node.js

Another significant trend is the movement toward using JavaScript as a primary programming language. AJAX first showed that JavaScript is important; jQuery showed us that it could be powerful and elegant; and Google's open source V8 JavaScript engine showed us that it could be fast. Today, JavaScript is becoming a serious server-side programming language. It serves as the data storage and querying language for several non-relational databases, including CouchDB and Mongo, and it is used as a general-purpose language in server-side platforms such as Node.js. Node.js has been around since 2009 and gained acceptance quickly. Its key innovations are as follows:

- *Using JavaScript*: Developers need to work only in a single language, from client-side code, through server-side logic, and even into data-querying logic via CouchDB or the like.
- *Being completely asynchronous*: Node.js's core API doesn't expose any way of blocking a thread while waiting for input/output (I/O) or any other operation. All I/O is implemented by beginning the operation and then later receiving a callback when the I/O is completed. This means that Node.js makes extremely efficient use of system resources and may handle tens of thousands of concurrent requests per CPU. (Alternative platforms tend to be limited to about one hundred concurrent requests per CPU.)

Node.js remains a niche technology. Its biggest contribution to web app development has, rather oddly, been to provide a consistent JavaScript engine on which development tools can be written. Many emerging client-side JavaScript frameworks, such as AngularJS, have good tooling support based on the use of Node.js.

Node.js adoption for deploying web apps has been slower. Most businesses building real applications in limited time frames typically need the infrastructure in full-stack frameworks such as Ruby on Rails and ASP.NET MVC. Node.js is mentioned here only to put some of ASP.NET MVC's design into context against industry trends. For example, ASP.NET MVC includes *asynchronous controllers* (which I describe in Chapter 19). This is a way to handle HTTP requests with non-blocking I/O and scale up to handle more requests per CPU.

Key Benefits of ASP.NET MVC

In October 2007, Microsoft announced a new MVC Web development platform, built on the core ASP.NET platform, clearly designed as a direct response to the evolution of technologies such as Rails and as a reaction to the criticisms of Web Forms. The following sections describe how this new platform overcame the Web Forms limitations and brought ASP.NET back to the cutting edge.

MVC Architecture

It is important to distinguish between the MVC architectural pattern and the ASP.NET MVC Framework. The MVC pattern is not new—it dates back to 1978 and the Smalltalk project at Xerox PARC—but it has gained enormous popularity today as a pattern for Web applications, for the following reasons:

- User interaction with an MVC application follows a natural cycle: the user takes an action, and in response the application changes its data model and delivers an updated view to the user. And then the cycle repeats. This is a convenient fit for Web applications delivered as a series of HTTP requests and responses.
- Web applications necessitate combining several technologies (databases, HTML, and executable code, for example), usually split into a set of tiers or layers. The patterns that arise from these combinations map naturally onto the concepts in MVC.

The ASP.NET MVC Framework implements the MVC pattern and, in doing so, provides greatly improved separation of concerns. In fact, ASP.NET MVC implements a modern variant of the MVC pattern that is especially suitable for Web applications. You will learn more about the theory and practice of this architecture in Chapter 3.

By embracing and adapting the MVC pattern, the ASP.NET MVC Framework provides strong competition to Ruby on Rails and similar platforms, and brings the MVC pattern into the mainstream of the .NET world. By capitalizing on the experience and best practices discovered by developers using other platforms, ASP.NET MVC has, in many ways, pushed forward beyond what even Rails can offer.

Extensibility

The MVC Framework is built as a series of independent components that satisfy a .NET interface or that are built on an abstract base class. You can easily replace components, such as the routing system, the view engine, and the controller factory, with a different one of your own implementation. In general, the MVC Framework gives you three options for each component:

- Use the *default* implementation of the component as it stands (which should be enough for most applications).
- Derive a *subclass* of the default implementation to tweak its behavior.
- *Replace* the component entirely with a new implementation of the interface or abstract base class.

You'll learn all about the various components, and how and why you might want to tweak or replace each of them, starting in Chapter 14.

Tight Control over HTML and HTTP

ASP.NET MVC produces clean, standards-compliant markup. Its built-in HTML helper methods produce standards-compliant output, but there is a more significant philosophical change compared with Web Forms. Instead of generating out swathes of HTML over which you have little control, the MVC Framework encourages you to craft simple, elegant markup styled with CSS.

Of course, if you do want to throw in some ready-made widgets for complex UI elements such as date pickers or cascading menus, ASP.NET MVC's "no special requirements" approach to markup makes it easy to use best-of-breed UI libraries such as jQuery UI or the Bootstrap CSS library. ASP.NET MVC meshes so well with jQuery, for example, that Microsoft ships jQuery as a built-in part of the default Visual Studio ASP.NET MVC project template, along with other popular libraries, such as Bootstrap, Knockout and Modernizr.

■ **Tip** I don't get into the detail of these "blessed" JavaScript libraries in this book because they are not part of the core MVC Framework and do their work within the browser. Client-side development for MVC Framework applications is an important topic, however, and you can learn more in my book *Pro ASP.NET MVC 5 Client*, which will be published by Apress in 2014. There are some libraries, however, that provide support for core features such as validation and Ajax requests and I describe these in Part 2 of this book. I describe Knockout in Chapter 27 and I use Bootstrap (albeit without a detailed introduction) throughout the book.

ASP.NET MVC–generated pages don't contain any View State data, so they are smaller than typical pages from ASP.NET Web Forms. Despite today's fast connections, this economy of bandwidth still gives an enormously improved end-user experience and helps reduce the cost of running a popular web application.

ASP.NET MVC works in tune with HTTP. You have control over the requests passing between the browser and server, so you can fine-tune your user experience as much as you like. AJAX is made easy, and there aren't any automatic postbacks to interfere with client-side state.

Testability

The MVC architecture gives you a great start in making your application maintainable and testable because you naturally separate different application concerns into independent pieces. Yet the ASP.NET MVC designers didn't stop there. To support unit testing, they took the framework's component-oriented design and made sure that each separate piece is structured to meet the requirements of unit testing and mocking tools.

They added Visual Studio wizards to create unit test projects on your behalf, which can be integrated with open source unit test tools such as NUnit and xUnit as well as the test tools that are included in Visual Studio, which I introduce in Chapter 6. Even if you have never written a unit test before, you will be off to a great start.

In this book, you will see examples of how to write clean, simple unit tests for ASP.NET MVC controllers and actions that supply fake or mock implementations of framework components to simulate any scenario, using a variety of testing and mocking strategies.

Testability is not only a matter of unit testing. ASP.NET MVC applications work well with UI automation testing tools, too. You can write test scripts that simulate user interactions without needing to guess which HTML element structures, CSS classes, or IDs the framework will generate, and you do not have to worry about the structure changing unexpectedly.

Powerful Routing System

The style of URLs has evolved as Web application technology has improved. URLs like this one:

```
/App_v2/User/Page.aspx?action=show%20prop&prop_id=82742
```

are increasingly rare, replaced with a simpler, cleaner format like this:

```
/to-rent/chicago/2303-silver-street
```

There are some good reasons for caring about the structure of URLs. First, search engines give weight to keywords found in a URL. A search for "rent in Chicago" is much more likely to turn up the simpler URL. Second, many Web users are now savvy enough to understand a URL, and appreciate the option of navigating by typing it into their browser's address bar. Third, when someone understands the structure of a URL, they are more likely to link to it, share it with a friend, or even read it aloud over the phone. Fourth, it doesn't expose the technical details, folder, and file name structure of your application to the public Internet, so you are free to change the underlying implementation without breaking all your incoming links.

Clean URLs were hard to implement in earlier frameworks, but ASP.NET MVC uses a feature known as *URL routing* to provide clean URLs by default. This gives you control over your URL schema and its relationship to your application, offering you the freedom to create a pattern of URLs that is meaningful and useful to your users, without the need to conform to a predefined pattern. And, of course, this means you can easily define a modern REST-style URL schema if you wish. You'll find a thorough description of URL routing in Chapters 15 and 16.

Built on the Best Parts of the ASP.NET Platform

Microsoft's existing ASP.NET platform provides a mature, well-proven set of components and facilities for developing effective and efficient Web applications. First and most obviously, as ASP.NET MVC is based on the .NET platform, you have the flexibility to write code in any .NET language and access the same API features—not just in MVC itself but in the extensive .NET class library and the vast ecosystem of third-party .NET libraries.

Second, ready-made ASP.NET platform features—such as authentication, membership, roles, profiles, and internationalization—can reduce the amount of code you need to develop and maintain any Web application, and these features are just as effective when used in the MVC Framework as they are in a classic Web Forms project. The underlying ASP.NET platform provides a rich set of tools on which to build web applications with the MVC Framework.

■ **Note** I describe the most commonly used ASP.NET Platform features as they relate to MVC development in this book, but the platform is a topic in its own right. For complete details of the rich features that the ASP.NET platform provides, see my forthcoming *Pro ASP.NET MVC 5 Platform*, which will be published by Apress in 2014.

Modern API

Microsoft's .NET platform has evolved with each major release, supporting – and even defining – the state-of-the-art aspects of modern programming. ASP.NET MVC 5 is built for .NET 4.5.1, so its API can take full advantage of recent language and runtime innovations, including the `await` keyword, extension methods, lambda expressions, anonymous and dynamic types, and Language Integrated Query (LINQ). Many of the MVC Framework's API methods

and coding patterns follow a cleaner, more expressive composition than was possible with earlier platforms. Don't worry if you are not up to speed on the latest C# language features: I provide a summary of the most important C# features for MVC development in Chapter 4.

ASP.NET MVC Is Open Source

Unlike previous Microsoft Web development platforms, you are free to download the original source code for ASP.NET MVC, and even modify and compile your own version of it. This is invaluable when your debugging trail leads into a system component and you want to step into its code (and even read the original programmers' comments). It is also useful if you are building an advanced component and want to see what development possibilities exist, or how the built-in components actually work.

Additionally, this ability is great if you do not like the way something works, if you find a bug, or if you just want to access something that's otherwise inaccessible, because you can simply change it yourself. However, you'll need to keep track of your changes and reapply them if you upgrade to a newer version of the framework. ASP.NET MVC is licensed under the Microsoft Public License (Ms-PL, http://www.opensource.org/licenses/ms-pl.html), an Open Source Initiative (OSI)–approved open source license. This means that you can change the source code, deploy it, and even redistribute your changes publicly as a derivative project. You can download the MVC source code from http://aspnetwebstack.codeplex.com.

What Do I Need to Know?

To get the most from this book, you should be familiar with the basics of web development, have an understanding of how HTML and CSS work and a working knowledge of C#. Don't worry if you are a little hazy on the client-side details. My emphasis is on server-side development in this book and you can pick up what you need through the examples. In Chapter 4, I provide a summary of the most useful C# language features for MVC development, which you'll find useful if you are moving to the latest .NET versions from an earlier release.

What Is the Structure of This Book?

This book is split into 2 parts, each of which covers a set of related topics.

Part 1: Introducing ASP.NET MVC 5

I start this book by putting the ASP.NET MVC Framework in context. I explain the benefits and practical impact of the MVC pattern, the way in which the MVC Framework fits into modern web development and describe the tools and C# language features that every MVC Framework programmer needs.

In the next chapter you will dive right in and create a simple web application and get an idea of what the major components and building blocks are and how they fit together. Most of this part of the book, however, is given over to the development of a project called SportsStore, through which I show you a realistic development process from inception to deployment, touching on the major features of the ASP.NET MVC Framework.

Part 2: ASP.NET MVC in Detail

In Part 2, I explain the inner workings of the MVC Framework features that I used to build the SportsStore application. I show you how each feature works, explain the role it plays in the MVC Framework and show you the configuration and customization options that are available. Having set the broad context in Part 1, I dig right into the details in Part 2.

What's New in this Edition?

Version 5 of the MVC Framework is a relatively minor upgrade and a lot of the changes are really to do with the way that ASP.NET projects are created and managed in Visual Studio. Table 1-1 briefly describes the new MVC Framework features and details where you can find more information about them in this book.

Table 1-1. *The New Features in MVC 5*

Feature	Description	See Chapter
Authentication Filters	A new kind of filter that can be used to include different types of authentication within the same controller	18
Filter Overrides	A new kind of filter that is applied to action methods to prevent filters defined globally or on the controller from taking effect	18
Attribute Routing	A set of attributes that allow URL routes to be defined within the controller class	15, 16

ASP.NET version 4.5.1, on which the MVC Framework 5 is built, has been enhanced as well. The most important change is the addition of the ASP.NET Identity API, which replaces the Membership system for managing user credentials. I don't cover ASP.NET Identity in this book, although I do explain how authentication and authorization are applied to MVC Framework applications through the use of features like filters.

■ **Note** I will be covering ASP.NET Identity in my *Pro ASP.NET MVC 5 Platform* book, which will be published in 2014 and cover all of the facilities that the ASP.NET platform provides. That said, I don't want you to have to buy a second book to learn about something as important as user security, and so Apress has agreed to distribute the security-related chapters from that book from its web site for download without charge when that book is published. Those chapters won't be available immediately because I have not written the platform book yet, but it is my next major writing project after this book and my hope is that the delay won't be too long.

A new edition is a chance to go beyond writing about new features and I have made some other changes for this book. I have expanded the SportsStore example to show the basics of responsive and mobile web application development, I added quick references to the start of all the in-depth chapters so you can find easily specific examples, and I added a chapter that shows how one of the open source libraries that Microsoft has embraced—Knockout—can be combined with the Web API feature to create Single-Page Applications (SPAs).

Where Can I Get the Example Code?

You can download all of the examples for all of the chapters in this book from `Apress.com`. The download is available without charge and includes all of the Visual Studio projects and their contents. You don't have to download the code, but it is the easiest way of experimenting with the examples and cutting and pasting techniques into your own projects.

What Software Do I Need for This Book?

The only software you need for MVC development is Visual Studio 2013, which contains everything you need to get started, including a built-in application server for running and debugging MVC applications, an administration-free edition of SQL Server for developing database-driven applications, tools for unit testing and, of course, a code editor compiler and debugger.

There are several different editions of Visual Studio, but I will be using the one that Microsoft makes available free of charge, called *Visual Studio Express 2013 for Web*. Microsoft adds some nice features to the paid-for editions of Visual Studio, but you will not need them for this book and all of the figures that you see throughout this book have been taken using the Express edition, which you can download from http://www.microsoft.com/visualstudio/eng/products/visual-studio-express-products. There are several different versions of Visual Studio 2013 Express, each of which is used for a different kind of development. Make sure that you get the Web version, which supports ASP/NET applications.

Once you have installed Visual Studio, you are ready to go. Microsoft has improved the scope of the features in the Visual Studio Express in recent years and there is nothing else you need to follow along with this book. I do rely on additional software packages, but these are installed through Visual Studio and don't require separate downloads and installations (and are available without cost).

■ **Tip** I have used Windows 8.1 throughout this book, but you can use Visual Studio 2013 and develop MVC applications quite happily on earlier versions of Windows. See the system requirements for Visual Studio 2013 for details of which versions and patch levels are supported).

Credits

In Chapter 10, I use a feature of the Bootstrap CSS library called *Glyphicons Halflings*, which are a set of icons that are not usually available for free, but for which the creator has given an open license for their inclusion in Bootstrap. The only request is that the creator's URL be quoted when it is possible to do so, which seems like a fair and reasonable thing to do. Here is it: http://glyphicons.com.

Summary

In this chapter, I explained the context in which the MVC Framework exists and how it compares to Web Forms. I described the benefits of using the MVC framework, the structure of this book and the software that you will require to follow the examples.

You saw how the ASP.NET MVC platform addresses the weaknesses of ASP.NET Web Forms, and how its modern design delivers advantages to developers who want to write high-quality, maintainable code. In the next chapter, you'll see the MVC Framework in action in a simple demonstration of the features that deliver these benefits.