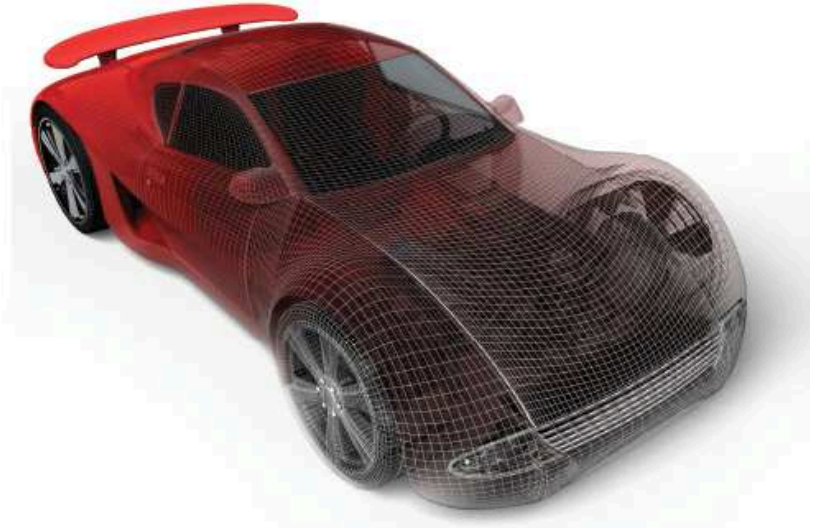
Extracted from:

Test-Drive ASP.NET MVC

The Pragmatic Programmers

Covers ASP.NET MVC 2.0

Test-Drive ASP.NET MVC



Jonathan McCracken

Edited by Susannah Davidson Pfalzer

If at first the idea is not absurd, then there is no hope for it.
► Albert Einstein

Preface

It's testable. It's lightweight. It's open source. It's . . . Microsoft? Yes, ASP.NET MVC is an open source web application framework created by Microsoft to cater to the needs of agile software developers. Since its official release in early 2009, it has been downloaded by almost 1 million developers, and it is rapidly being adopted by many organizations because of its efficient development model. Simply put, it's C# on the Web done right.

With this book's test-driven approach to ASP.NET MVC, you'll gain the cutting-edge skills to build your next web application and become a more agile developer in the process.

What Makes ASP.NET MVC Special?

Microsoft offers two web presentation frameworks: ASP.NET Web Forms and ASP.NET MVC. ASP.NET itself is the common set of libraries and features that both ASP.NET Web Forms and ASP.NET MVC work on top of. This supports customers' existing needs with the older ASP.NET Web Forms and their future needs with ASP.NET MVC. Although ASP.NET MVC shares many of the same underpinnings of ASP.NET, it overcomes its brother's weaknesses. ASP.NET MVC was designed using the latest innovations and lessons learned on how to build web applications. This adds up to big productivity improvements for your teams.

Here's what ASP.NET MVC offers that ASP.NET Web Forms doesn't.

Full Control Over Markup

If you've ever developed an ASP.NET Web Forms website, you'll know what a struggle it is to build a site for anything other than Internet Explorer. This is partly because ASP.NET Web Forms was designed for intranet applications where a single browser could be more easily mandated. For most companies, supporting only one browser isn't an

option anymore. Many companies are focusing on enabling their partners and customers to perform their work through web applications, so they need to support multiple browsers.

The Achilles' heel of ASP.NET Web Forms is its bloated HTML. It generates complex markup through a string of embedded web and user controls. ASP.NET MVC comes to the rescue with a much simpler solution. Its default view engine, which is confusingly named the *Web Forms view engine*, gives you full control over your markup. No more strange **id** tags with **$** and underscores in them. This pays off when dealing with client-side scripting such as JavaScript. You'll find out more about the Web Forms view engine in Chapter 7, *Composing Views with Ajax and Partials*, on page 153.

Testability

A web application framework that has out-of-the-box testing saves you a lot of time. Most developers building ASP.NET Web Forms applications had to use their own design patterns, such as Model-View-Presenter (MVP), to accomplish this. For developers who don't know much about unit testing, it's less obvious how to approach testing. ASP.NET MVC solves this with a clear way to test your code. I'll be focusing on this point heavily throughout the book to walk you through how to write a well-tested ASP.NET MVC application.

Convention Over Configuration

Following convention saves time. ASP.NET MVC's timesaving conventions keep you out of configuration files, and some conventions give you added benefits, such as search engine optimization. For example, in ASP.NET MVC, URLs to your site become more readable by engines. Instead of http://yourblog.com/Blog/Entry.aspx?id=108 in ASP.NET Web Forms, ASP.NET MVC can do much better, such as http://yourblog.com/Blog/Entry/108/MVC-Makes-Search-Engines-Happy. You can achieve the same thing with ASP.NET Web Forms, but it's less straightforward.[1] With ASP.NET MVC, you get it for free. You'll see more of these conventions throughout Part II, "Building an Application."

Extensible Architecture

Striking a balance between conventions and extensibility is tricky for web frameworks. If too many conventions are prescribed, they can

1. http://weblogs.asp.net/scottgu/archive/2009/10/13/url-routing-with-asp-net-4-web-forms-vs-2010-and-net-4-0-series.aspx

restrict you from extending the framework when you need to do so. The opposite is also true: if no conventions are set, then your team has to continue to reinvent the wheel.

ASP.NET MVC strikes a pretty good balance. It comes with a powerful default view engine but makes it easy to extend or create your own. You'll learn about this in Section 6.2, *Building a Custom HTML Helper*, on page 137. ASP.NET MVC has a feature called *action filters* that you can extend to provide helpful features such as transaction support. You'll tackle this in Section 9.4, *Creating a Custom Action Filter*, on page 202. Because ASP.NET MVC's architecture has a single point of creation for all the controllers, you can extend it with *dependency injection*. Dependency injection decouples object behaviors, or, more specifically, the implementation of those behaviors. We pass the behavior to the constructor, effectively "injecting" it into the object. You'll see how to do this in Section 5.1, *IControllerFactory: Where Controllers Are Born*, on page 100.

Finally, ASP.NET MVC isn't tied to any single persistence framework (see the *Joe Asks...* on page 21 for more on persistence frameworks). In fact, it doesn't come bundled with one at all. This leaves room for you to choose the right tool for the job. In this book, you'll be using NHibernate, one of the most popular open source persistence frameworks. You'll see how to use NHibernate in Chapter 8, *Persisting Your Models*, on page 174.

Why Test-Driven Development?

Test-driven development (TDD) is a simple programming technique that *drives* your development by starting with a failing unit test. It's quickly becoming a standard practice on projects because TDD helps you feel more confident about your code. If you've never used TDD before, then Chapter 2, *Test-Driven Development*, on page 33 will show you how. With TDD, you'll spend much more time coding and much less time fiddling around with the debugger.

The other key advantage to this method is that it helps you learn a framework faster. Tests, when they pass, confirm that you've written a bit of code correctly, and you can even dig into the tests that the framework offers. Because ASP.NET MVC is open source, you're free to browse all of its unit tests to help you gain an even better understanding of it.

And if you're a seasoned test-driven developer who's embarking on learning ASP.NET MVC, this book will be your guide on how and what to test.

Who Should Read This Book?

This book was written for two audiences: Microsoft developers and non-Microsoft developers. The goal for both is the same: to learn how to build an ASP.NET MVC application based on development best practices.

For Microsoft developers with a long history of building applications using Microsoft frameworks, the emphasis on TDD might be unfamiliar to you. Almost all the code examples in this book have been written with TDD and are explained so that you can understand both how the tests work and how the ASP.NET MVC code works. Also, you'll learn about some tools and open source projects that can save you time when developing your ASP.NET MVC applications.

For non-Microsoft developers, you'll find the methods of testing familiar, but learning the language and the framework will be your primary focus. Although this book assumes a basic knowledge of the C# language, each tutorial explains line by line what the code is doing and why it is important.

Although you can develop VB .NET web applications with ASP.NET MVC, all the samples in this book are written in C#. If you're comfortable reading C# and translating for yourself, then you'll be fine using this book as your guide to ASP.NET MVC.

What's in This Book?

Part I of this book shows you how to build an ASP.NET MVC application and introduces you to the TDD approach.

Part II focuses on building a sample application. You will work through test-driving core components of ASP.NET MVC, as well as other essential frameworks that integrate with it. In Chapter 7, *Composing Views with Ajax and Partials*, on page 153, you will focus on working with jQuery.

Part III builds on the same application but introduces how to work with other frameworks. The database access in ASP.NET MVC is flexible, and

you'll find out about NHibernate in Chapter 8, *Persisting Your Models*, on page 174. Also, you'll learn how to use the Castle Windsor container in Chapter 9, *Integrating Repositories with Controllers*, on page 190. To integrate with other applications, you'll also learn how to create Representational State Transfer (REST) web services in Chapter 10, *Building RESTful Web Services*, on page 212.

Part IV focuses on deployment, something that many of us struggle with. Chapter 12, *Build and Deployment*, on page 249 is dedicated to this subject. You'll also learn about nonfunctional requirements in Chapter 11, *Security, Error Handling, and Logging*, on page 231.

To get the most out of this book, it's highly recommended that you code through the problems while reading. Not only will this help you learn the concepts of the framework and experience the subtle differences in each test, but, more important, you'll master the test-driven discipline. This is a skill you'll take with you to every language you program in. Whether you are programming in C#, Java, or Ruby, knowing how to write tests will help you write high-quality code in shorter periods of time.

What's New in ASP.NET MVC 2.0?

Since version 1.0 of ASP.NET MVC was released in March 2009, the development team in Redmond has been working tirelessly at improving the framework in the 2.0 release. More evolutionary than revolutionary, these changes make view and model development easier. Let's talk quickly about the new features.

Strongly Typed HTML Helpers

These new helpers reduce errors at compile time as well as the number of lines of code in your views. The helper methods are an improvement over checking properties at runtime. For example, we'd do this in ASP.NET MVC 1.0 to render a textbox for a person:

```
Html.TextBox("Name");
```

This standard Html helper renders a textbox. It's linked to the Name property of the model so that when it's filled out, the model itself is updated. In ASP.NET MVC 2.0, you do it like this:

```
Html.EditorFor(person => person.Name);
```

Here the EditorFor() renders a textbox for the Person model and checks for the presence of the Name property at compile time. Compile-time checking alerts you early to typos that break your code. It also helps if you rename properties of models that are referenced in views.

```
Html.EditorFor<Person>(person => person);
```

EditorFor() can also check for all the properties of the Person and render them all for editing. In this case, the lambda expression we pass is the whole model, not just a single property. You'll get to use the DisplayFor() helper methods in Section 1.3, *MVC in Five Minutes: Building Quote-O-Matic*, on page 26.

Templated Views

Templated views build on what strongly typed view helpers allow us to do. With ASP.NET MVC 2.0, you can now create generic view templates that let you postpone customizing views. This works well for prototyping applications, such as when your pages need just enough information to get feedback from your customer to know whether you're on the right track. Building your own templates is as simple as creating a view under the View/Shared directory named after the controller's action. Instead of creating a view per model to show or create details, ASP.NET MVC can fall back on your templated views. You'll look at this feature in Section 4.3, *Adding Thoughts with Templated Views*, on page 85.

Data Annotations

Data annotations are a way to mark up your models with validation rules. For example, if you wanted to make sure that a user's name was no longer than twenty-five characters, you could add this attribute:

```
[StringLength(25, ErrorMessage="Invalid Length")]
public string Name {get; set;}
```

The attribute **StringLength** specifies a length of a maximum of twenty-five, and the ErrorMessage value will be the message you display to the user if they input a name that is too long or short. You'll see more of this in Section 6.4, *Adding Validations Using ModelStateDictionary*, on page 145.

Other Features

Areas, asynchronous controllers, and Html.RenderAction() are other useful new features in ASP.NET MVC 2.0. Because they're more advanced or specialized, they won't be covered in this book. Areas extend the way

files are organized in an ASP.NET MVC project and are aimed at larger web applications (see Phil Haack's blog[2] for a tutorial on how to use them). Asynchronous controllers are for long-running tasks that can be run in parallel. Finally, Html.RenderAction() provides a more efficient way for HTML to be written to the response.

Online Resources

At the website for this book, http://pragprog.com/titles/jmasp, you'll find the following:

- You'll find the source code for all the snippets used in this book, including the full codebase for the sample application from Parts II and III. You can find the final solution in the GetOrganizedFinal folder when you unzip it.
- You'll find an errata page, where you can post errors you find in the current edition.
- You'll find a discussion forum where you can communicate with me and other ASP.NET MVC developers directly.

In addition, once you get to the end of the book, Section 12.3, *That's All, Folks*, on page 269 will give you some additional online resources to sites where you can further your learning.

Feel free to use the source code in your own applications. However, keep in mind that not all the examples in the book are fit for production code, because some are there to help you learn only. If you're reading the ebook version of this book, you can download and play with the code by clicking the little gray rectangle before the code listings.

Let's get started with a high-level overview in Chapter 1, *Getting Started with ASP.NET MVC*, on page 20, where we'll build a simple web application. Following that, in Chapter 2, *Test-Driven Development*, on page 33, we'll learn the basics of this more efficient form of development. With that knowledge, we'll be able to tackle building a full-featured end-to-end sample application for the rest of the book.

2. http://haacked.com/archive/2010/01/12/ambiguous-controller-names.aspx

The Pragmatic Bookshelf

The Pragmatic Bookshelf features books written by developers for developers. The titles continue the well-known Pragmatic Programmer style and continue to garner awards and rave reviews. As development gets more and more difficult, the Pragmatic Programmers will be there with more titles and products to help you stay on top of your game.

Visit Us Online

Home Page for Test-Drive ASP.NET MVC
http://pragprog.com/titles/jmasp
Source code from this book, errata, and other resources. Come give us feedback, too!

Register for Updates
http://pragprog.com/updates
Be notified when updates and new books become available.

Join the Community
http://pragprog.com/community
Read our weblogs, join our online discussions, participate in our mailing list, interact with our wiki, and benefit from the experience of other Pragmatic Programmers.

New and Noteworthy
http://pragprog.com/news
Check out the latest pragmatic developments, new titles and other offerings.

Buy the Book

If you liked this eBook, perhaps you'd like to have a paper copy of the book. It's available for purchase at our store: pragprog.com/titles/jmasp.

Contact Us

Online Orders:	www.pragprog.com/catalog
Customer Service:	support@pragprog.com
Non-English Versions:	translations@pragprog.com
Pragmatic Teaching:	academic@pragprog.com
Author Proposals:	proposals@pragprog.com
Contact us:	1-800-699-PROG (+1 919 847 3884)