Microsoft

ASP.NET Core Application Development

Building an application in four sprints

Professional

James Chambers
David Paquette
Simon Timms

FREE SAMPLE CHAPTER

SHARE WITH OTHERS

Microsoft

ASP.NET Core Application Development: Building an application in four sprints (Developer Reference)

James Chambers
David Paquette
Simon Timms

PUBLISHED BY
Microsoft Press
A division of Microsoft Corporation
One Microsoft Way
Redmond, Washington 98052-6399

Library of Congress Control Number: 2016958907
ISBN: 978-1-5093-0406-6

Printed and bound in the United States of America.

First Printing

Microsoft Press books are available through booksellers and distributors worldwide. If you need support related to this book, email Microsoft Press Support at mspinput@microsoft.com. Please tell us what you think of this book at http://aka.ms/tellpress.

Acquisitions Editor: Laura Norman
Developmental Editor: Troy Mott
Editorial Production: Ellie Volckhausen
Technical Reviewer: Rachel Appel
Copyeditor: Rachel Jozsa
Indexer: Julie Grady
Cover: Chuti Prasertsith

I would like to dedicate this book to my loving wife. Thank you for your support and I look forward to spending more time outdoors with you once I finally emerge from this office.

—David Paquette

I would like to dedicate this book to my children who have been forced to play without me while I work feverishly pressing buttons in front of a glowing box and my wife who taught them to sing Cat's In The Cradle outside my office door. Without their support I would be but a quarter of a person. I love them more than I have words to express.

—Simon Timms

I dedicate this book to my incredible, intelligent and striking wife who has supported me for over twenty years while I chase dreams and sometimes our pet dog who gets out in the middle of the night. I dedicate it to my children that I love endlessly and completely. I can't wait to spend more time with them, even if they need more showers and should ask for rides less.

—James Chambers

This page intentionally left blank

Contents at a Glance

Table of Contents

What do you think of this book? We want to hear from you!

Microsoft is interested in hearing your feedback so we can improve our books and learning resources for you. To participate in a brief survey, please visit:

http://aka.ms/tellpress

What do you think of this book? We want to hear from you!

Microsoft is interested in hearing your feedback so we can improve our books and learning resources for you. To participate in a brief survey, please visit:

http://aka.ms/tellpress

Introduction

ASP.NET Core MVC is Microsoft's latest web framework for .NET developers. It is the next version of the now-familiar MVC Framework and aspires to cross boundaries by enabling cross-platform development and deployment. It leverages a wide assortment of open source libraries and is, itself built as open source software. ASP.NET Core MVC helps developers to separate concerns like business logic, routing, services, and views and provides new systems for configuration and extensibility. It uses the C# programming language and the Razor view engine. If you are an experienced .NET developer or a newcomer to the .NET platform, ASP.NET Core MVC is likely what your projects will be built from.

This book follows the first few sprints of an application being redeveloped by a team at a fictional company named Alpine Ski House. Each chapter contains a little bit of information about the challenges the team is facing and how they work to overcome them. Despite having a short story element to each chapter, the book dives deep to cover not only the features of ASP.NET Core MVC, but also the tooling around it that developers will use to build, maintain and deploy their applications.

In addition to its story elements and technical information around ASP.NET Core MVC, the book discusses the new version of Entity Framework, package management systems, and peripheral technologies that are used by modern web developers. Beyond the explanatory content, the book also comes with an accompanying project—the very same project that the developers at Alpine Ski House have built.

Who should read this book

The book takes a programmer through all the steps necessary to build a brand new application on ASP.NET Core and push it out so it is available on the Internet. There is still a great population of programmers who have yet to journey onto the web or have done so only using webforms, much less using the full gamut of tooling that is available today. This book will help put the skills and confidence needed in place to build modern applications on an emerging framework. It will help you explore application architecture, deployment and building applications that are designed for the cloud.

Assumptions

Readers should know how to program at an intermediate to senior level. Readers should be proficient in C#, have a background in web development, and understand fundamentals of working in Visual Studio. Experience with previous versions of MVC will be beneficial, but not required. Familiarity in working with a command line interface will be an asset. After completing this book you will be able to build a meaningful and relevant database-driven application and deploy it to a cloud-based infrastructure.

This book might not be for you if…

This book might not be for you if you are an advanced ASP.NET MVC developer who has been closely following or contributing to the development of ASP.NET Core MVC.

Organization of this book

This book offers the innovative approach of taking developers through individual sprints in the development of an application. It will cover not just the technology, but also the process of recovering from mistakes and adjusting to user feedback, starting from a blank canvas and ending with a real working product.

This book is divided into four sections:

- Part 1, "Alpine Ski House," Covers background information that sets up the example app and fictional characters in the story used throughout the book
- Part 2, "Sprint Retro: A Journey of 1000 Steps," focuses on the features required to get our application out the door, configuring the pipeline so that deployment happens on-the-fly in a way that the whole team understands.
- Part 3, "Sprint Retro: The Belly of the Beast," focuses on the core features needed to get the business running with our sample application. Here we introduce data access using the Entity Framework Core, creating views using Razor, Configuration and Logging, Security and User Management, and finally Dependency Injection.
- Part 4, "Sprint Retro 3: Home Stretch" covers JavaScript and dependency management, along with building on top of the previous foundations.

Postfix covers some important topics such as testing, refactoring and extensibility.

Finding your best starting point in this book

The different sections of ASP.NET Core Application Development: Building an application in four sprints cover a wide range of technologies associated with the ASP.NET Core framework. Depending on your needs, and your existing understanding of Microsoft's web stack, you may wish to focus on specific areas of the book. Use the following table to determine how best to proceed through the book.

If you are	Follow these steps
New to ASP.NET Core development, or an existing ASP.NET Core developer	Focus on Parts I, II and III, or read through the entire book in order.
Familiar with earlier releases of ASP.NET	Briefly skim Chapter 1 and Chapter 2 if you need a refresh on the core concepts. Read up on the new technologies throughout the renaminder of the book.
Interested in client side development	Read Chapters 15, 16 and 17 in Part IV. Skim the section on JavaScript services in Chapter 20.
Interested in cross-platform development	The entire book is applicable to cross platform developemnt but Chapter 8 and 9 are specifically deidcated to the topic.

Most of the book's chapters include hands-on samples that let you try out the concepts just learned. No matter which sections you choose to focus on, be sure to download and install the sample applications on your system.

Conventions and features in this book

This book presents information using conventions designed to make the information readable and easy to follow.

- The book includes samples for C# programmers and syntaxes such as HTML, CSS, SCSS and Razor.
- Boxed elements with labels such as "Note" provide additional information or alternative methods for completing a step successfully.
- A plus sign (+) between two key names means that you must press those keys at the same time. For example, "Press Alt+Tab" means that you hold down the Alt key while you press the Tab key.
- **A vertical bar** between two or more menu items (e.g. File | Close), means that you should select the first menu or menu item, then the next, and so on.

System requirements

You will need the following hardware and software to run the sample application accompanying this book:

- .NET Core 1.0 or newer, available cross platform from https://dot.net.
- Your code editor of choice. We use Visual Studio 2015 (any edition) or newer on Windows and Visual Studio Code on Windows, Mac and Ubuntu Linux.
- SQL Server LocalDB (included with Visual Studio 2015 or newer on Windows). Linux or Mac users will need access to a SQL Server database hosted either on a Windows machine or in Microsoft Azure.
- Computer that has a 1.6GHz or faster processor
- At least 1 GB of RAM
- 4 GB of available hard disk space
- Internet connection to download software and sample project

Depending on your Windows configuration, you might require Local Administrator rights to install or configure Visual Studio 2015.

Downloads: Sample Project

Most of the chapters in this book include snippets from the sample project. The sample project is available on GitHub:

https://github.com/AspNetMonsters/AlpineSkiHouse

Follow the instructions on the GitHub repository to download and run the sample project.

Note In addition to the sample project, your system should have .NET Core 1.0 or newer installed.

Errata, updates, & book support

We've made every effort to ensure the accuracy of this book and its companion content. You can access updates to this book—in the form of a list of submitted errata and their related corrections—at:

https://aka.ms/ASPCoreAppDev/errata

If you discover an error that is not already listed, please submit it to us at the same page.

Get all code samples, including complete apps, at: *https://aka.ms/ASPCoreAppDev/downloads*.

If you need additional support, email Microsoft Press Book Support at *mspinput@microsoft.com*.

Please note that product support for Microsoft software and hardware is not offered through the previous addresses. For help with Microsoft software or hardware, go to *http://support.microsoft.com*.

Free ebooks from Microsoft Press

From technical overviews to in-depth information on special topics, the free ebooks from Microsoft Press cover a wide range of topics. These ebooks are available in PDF, EPUB, and Mobi for Kindle formats, ready for you to download at:

http://aka.ms/mspressfree

Check back often to see what is new!

We want to hear from you

At Microsoft Press, your satisfaction is our top priority, and your feedback our most valuable asset. Please tell us what you think of this book at:

http://aka.ms/tellpress

We know you're busy, so we've kept it short with just a few questions. Your answers go directly to the editors at Microsoft Press. (No personal information will be requested.) Thanks in advance for your input!

Stay in touch

Let's keep the conversation going! We're on Twitter: *http://twitter.com/MicrosoftPress*

PART I

Alpine Ski House

Here is some background information that introduces the fictional aspect covered in this book, including the fictional characters that are creating the Alpine Ski House application.

Even the most fervent riders had to admit it: the season was at an end. It hadn't been the best season in memory but nor had it been the worst. It had been, in every way, unremarkable. There had been moments, a power outage in late February had forced the dusting off of an emergency plan which had long been practiced but never used. There had been reports on the local news station about children trapped on gondolas for hours but with the balmy weather nobody had ever truly been in danger. A smattering of free passes was all that was required to keep the skiers and riders coming.

The spring was a time for the permanent staff to regroup and the seasonal staff to return to wherever it is that lefties go in the summer. A rumor among the permanent staff was that at least half of the seasonal staff were rounded up by immigration as soon as they stepped off the hill and sent back to Australia. Dani-

elle couldn't imagine why the young seasonal staff would resist being sent back to Australia. One thing was for sure, it was much more exciting in Australia than in the sleepy mountain town that reemerged from hibernation each winter.

It was still too early to plan the next year and Danielle was looking forward to a month or two of down time before the cycle began anew. She had been the lone developer for Alpine Ski House for close to a decade and every year was about the same. Most of her time involved keeping legacy systems running and adjusting whatever small things were needed for the next year's activities. It wasn't the most exciting job in the world but over the winter months it was expected that everybody would sneak off for a couple of hours skiing on nice days and that was a perk she thoroughly enjoyed.

Opening the door to the low-rise that Alpine Ski House called home she was surprised to see that things were buzzing. People she wouldn't have expected to see in the office for a couple of hours were scattered about in huddles throughout the open plan office. Confused, Danielle dumped her bag and grabbed a coffee before looking for a huddle to join. The rest of the IT staff seemed to be orbiting Tim, the portly IT manager and her boss. Danielle headed over to join.

"Danielle! What do you think of the news, going to be an exciting time if you ask me," boomed Tim.

"What news is this?" asked Danielle.

"Where have you been?" asked Arjun, "We just bought out Thunder Valley and Mount Ballyhoo. They're merging operations and we're all going to lose our jobs!"

The two other ski areas were a few miles down the road. Thunder Valley was a small operation with only three lifts but a loyal following of ski bums. It was a favorite for the locals who wanted a break from the crowds of tourists in the winter months. It couldn't be more different from Mount Ballyhoo if it had been the output of Babbage's difference machine. Mount Ballyhoo was a huge ski hill spanning three mountains with countless lifts and enough on hill accommodation to house everybody in town twice over. Every weekend they had bands playing on the hill,

and it was not uncommon to see famous people like Scott Gu and John Skeet there rubbing shoulders with the commoners.

"Now Arjun," said Tim, "nobody has said anything about layoffs or redundancies or anything of the sort. Why at times like this the workload for IT usually increases because management wants systems integrated right away. We're just going to have to wait and find out what the plans are."

Danielle had to sit down. She was years still from retirement and didn't want to find another job. How many jobs would there be for programmers in a seasonal town like this? "This is silly," she told herself, "there is no use planning a move back to the big city based on this sort of uncertainty. Things will shake out in the next couple of weeks."

As it turned out nobody was waiting a couple of weeks.

As soon as lunch, a dandelion and walnut salad, with balsamic sweet potato crisps, was over, Tim came knocking at her cubicle.

"We're gathering in the big conference room. It sounds like the programmers from Thunder and Ballyhoo are here."

Gulping down the rest of her goat's milk, Danielle grabbed a pen and a big yellow legal pad and hurried towards the conference room. The pad and paper were just for show; she hadn't taken notes at a meeting in years. It was easier to just engage people around the small office in person than plan things out way in advance on a notepad. Better to make a good impression right away with layoffs on the horizon.

The big conference room was rarely used outside of potlucks because there simply weren't enough people to fill it. But today it was, if not packed, at least well used. Five young hipster looking individuals sat at one end of the table sipping on all manner of exotic looking smoothies. Danielle wondered how one would even go about importing rambutan and what sort of carbon footprint it would have. Still it was better than that time she had hacked her way into a durian only to have to defenestrate the offensive fruit.

Clearly divided from the hipsters were a group guys who would have been called "suits" in the big city. Out here they

looked like they had just stepped of a golf course. Somehow they were already tanned and relaxed looking.

Tim waited for everybody to settle and then addressed the room, "Good news, everybody, this is the team we're moving forward with. If you've made it to this room, then your job is safe and you can relax. I'm sure you all have questions about that and you can come see me individually after this meeting if you want to talk.

"Management has asked me to keep a larger number of programmers on staff after the merge because they have some exciting new initiatives that they want us to embark upon. Over the next few years we're going to be refreshing all the custom systems we have in place to run the hill. They recognize that this is a big undertaking and some of them have been reading CIO magazine and they've learned about agile and microservices. I can assure you that I've given instructions that all future copies of that magazine are to be burned before they reach management's hands but for now we're stuck with it."

Tim had always had a bit of a rocky relationship with management's great new ideas. He was the emergency brake on their crazy ideas. He continued on. "The first thing management want is a way for people to buy their lift tickets online. I'm told that it is 2016 and that we should have that in place already and that every other hill in the world has it." Tim seemed exasperated by management's generalizations; it must have been a fun discussion when these orders came down.

"Management wants to see a prototype in a month's time. I think I can buy another week if we can show that we're making progress."

A month's time! Danielle was beside herself. A month was how long Danielle liked to spend getting her head around a problem. She looked over at the hipster developers hoping they shared her ashen face. But the wheatgrass crew were nodding along happily.

Tim looked like he was coming to a conclusion and readying to step down from his soapbox. "Look guys, we need this to buy us some capital with management. I'll clear any roadblocks in your way. Use whatever technology you think is best buy whatever tools you need. I trust you to get this done."

CHAPTER 3

Models, Views, and Controllers

It was a surprise when Adrian popped over to Danielle's cubicle. Maybe more surprising was the furrowed brow he was sporting. "Got a minute?" he asked in a hushed tone, and then walked away without waiting for an answer. She nodded a confused yes, mostly to herself, and slowly got up to follow him to a side room. She scanned around the development pit, but didn't see anyone watching, much less interested, and couldn't really get a read on why he was assuming the role of secret agent in this software docu-drama.

He closed the door behind her as she stepped into the room. "Look, I'm a CSS guy. I'm not going to sugar coat it. I know my way around jQuery well enough to troubleshoot, but I'm no programmer." He looked tense and Danielle wondered what he'd been thinking. "Everyone else seems to be buying into this shift over to Core or whatever it's called...but I'm...I mean, come on, I run Linux on my laptop and I don't even use Office." The room quickly drew silent.

"Are you still nervous, Adrian? Marc had mentioned that you were worried about the cuts." Danielle was nervous too, truth be told. She had lost some good friends herself when the merge went through, but she wasn't sure that was what he needed to hear at the moment.

"Well yeah, I guess," he replied. "But my knowledge of MVC is that it stands for Model-View-Controller, and I haven't taken a look any deeper than that. And you guys keep referring to it as a framework. If it's a framework, but you have to make your own models, views, and controllers, then the MVC part seems more than a little misleading, don't you think?"

He had a point. "Well, yeah, that's actually pretty true," said Danielle.

"I'm coming from a different view here; I just want to wrap my head around it. I want to learn, but I'm not sure I know where to start, either." Adrian pounded back the rest of his coffee like it was about to expire. "I know they said we're safe if we're still here, but I don't want to get caught with my feet standing still if they think there's still room to shrink the team."

"Okay, look," said Danielle, "I've got a bit of time right now, let's run through the basics and we'll learn together. We're all going to be fine."

The M, the V, and the C of it

Let's face it, the MVC Framework is a pretty boring name. The acronym used in the title is from the well-known Model-View-Controller pattern, and it helps to organize a project. If you're familiar with it, the name literally spells out some of the original intent to separate concerns, and moves away from the other common pattern at the time known as Page-Controller. The name can also be misleading. The framework is much more than just models, views and controllers. ASP.NET Core MVC has a growing set of tooling and libraries available to that help developers create great applications, worthy of the modern web world.

Let's do a quick recap on the aspects you should already understand, and then move into some more interesting aspects of the framework, officially known as ASP.NET Core MVC.

Diving into Models

First up is the letter M, so we'll start with Models. The model represents the data that we need to properly render an experience, or part of an experience, for a customer. Customers are navigating to a page in the application that is data-driven, and models are the data part. However, as far as intent goes, the model in question is actually what you'll be using to support the rendering of the view, and not the entity or entities in question in which you persist to the database.

Let's consider this example from Alpine Ski House's possible database design that deals with user account summaries, as shown in Figure 3-1. When you want to indicate to the user that she has a current season pass, you don't want to return the list of season passes to the view and iterate over the collection to see if one exists that has not yet expired.

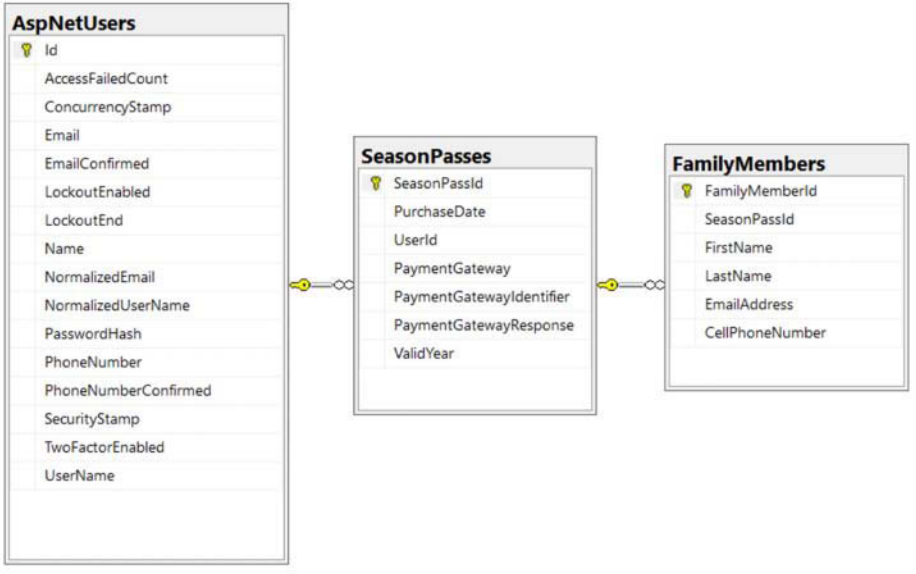

FIGURE 3-1 A screen shot showing a sampling of tables that might be used to model purchasing season passes

Returning all of this information to the view would be more than is required. Listing 3-1 contains a view model that might more closely approximate the information you would want to display to the user. As you can see, this is a POCO that sports the properties you can use to satisfy the view requirements without the view needing to make any decisions about what to display or any implementation of business logic. The view doesn't need to know what qualifies as a current season pass nor does it need to sift through any of the purchase details or iterate through child records in related tables to make sense of the data.

LISTING 3-1 The AccountSummaryViewModel Class

```
public class AccountSummaryViewModel
{
    public Guid UserId { get; set; }
    public int YearsOfMembership { get; set; }
    public bool IsCurrentSeasonPassHolder { get; set; }
    public List<string> IncludedFamilyMembers { get; set; }
}
```

The differentiation between what you craft for models on the front end, versus what you store in the database, is important not just for separating the concerns of the view and the business logic that supports it, but also for helping to prevent certain types of security issues. On the "write" side of things, when a view uses a database entity, the application becomes more likely to fall victim to overbinding bugs or attacks. Overbinding occurs when fields that weren't anticipated from an incoming request are present in form or querystring parameters. The model binder sees the properties, doesn't know that you hadn't intended for them to be there, and kindly fills in your data for you. As an example, consider the class representing some kind of a digital asset in Listing 3-2.

LISTING 3-2 The `AccountSummaryViewModel` Class

```
public class DigitalAsset
{
    public Guid AssetId { get; set; }
    public Guid AssetOwnerId { get; set; }
    public string Title { get; set; }
    public string Description { get; set; }
    public Uri AccessUri { get; set; }
}
```

This type of model can be used to display a list of resources made available to a user, and doing so is quite harmless. But, if you use the same object to receive edits for the record, a malicious user can exploit the fact that `AssetOwnerId` is a property and use that to take ownership of the asset. In fact, this is how Egor Homakov gained administrative privileges for the Ruby on Rails (RoR) repository on GitHub in 2012[1] (The technique in RoR is exploited through improperly checked models that make use of the

[1] GitHub reinstates Russian who hacked site to expose flaw, John Leyden, March 5, 2012, *http://www.theregister.co.uk*.

mass assignment feature, an analog to automatic model binding in ASP.NET Core MVC. Thankfully, Homakov's intentions were pure and no harm was done. We have learned from those binding conventions and habits of yore though. Today, we have many ways to protect ourselves, which we'll cover later in Chapter 13, "Identity, Security and Rights Management," but likely the easiest way is to make sure we're using models that are appropriate to the task at hand.

Most of the examples you find for view models will likely use an entity directly as the model type for the view; however, the approach does not facilitate other aspects of software development, such as testing, nor does it help with separating concerns in your controllers. Using the entity directly in a view means that you've achieved an undesirable level of coupling from the database all the way up to the view.

A model should be everything you need to render your page after you've taken care of business logic and often has a flattened view of a denormalized record from several tables in the database. For these reasons, and considering the intent of the object you're building up when you create a "model," you should likely think of it as the "view model" due to its close relationship and responsibility to the view.

Views

Here, the view in question happens to start with V and is indeed the view we're talking about in our new favorite acronym. Views in ASP.NET Core MVC are the files used to interleave parts of the model with the HTML needed in order to present the user with the intended user interface. If you create a new project from the default application template you will find all the views in the Views folder, or you can search Solution Explorer with the term ".cshtml," which is the extension used for Razor views.

Using the Razor view engine and syntax you've seen through the last few iterations of the MVC Framework, you can switch effortlessly between the syntax used to control flow or access our model or services, and the markup required to generate HTML.

In Listing 3-3 we have created an unordered list with values from the model's `IncludedFamilyMembers` collection. Razor lets you use C# inline with the HTML and is pretty smart about how it interprets what you throw at it. A simple `@` character is enough for the parser to know you're flipping into C#, and since angle brackets can't be used at the start of a valid C# statement, it can tell when you've switched back to HTML. We'll be covering Razor in greater detail in Chapter 11, "Razor Views."

LISTING 3-3 An example of mixing C# and HTML in Razor Syntax.

```
<ul>
    @foreach (var familyMember in Model.IncludedFamilyMembers)
    {
        <li>@familyMember</li>
    }
</ul>
```

Partial Views

Toolbars, authentication cues, shopping carts, parts of dashboards, and other similar components of your application often find themselves appearing on multiple pages, or even on all pages. In the name of Don't Repeat Yourself (DRY), you can create these components using a partial view, which can in turn be used repeatedly from any other page. You'll also see partial views referred to more simply as "partials." We'll use those terms interchangeably throughout the book.

Partials are not typically rendered on their own, but are used in composition of other views in your project. The first place you see this in any MVC application is likely to be in the _Layout.cshtml, where the view relies on partials to render the login status. Other common uses include using a partial view to render items in toolbars, shopping cart summaries like those you see at the top of an ecommerce site, or side bars with relevant data for the current page.

Child actions had to be rendered synchronously in previous versions of the MVC Framework, but the same ideas that made partials possible can now be used to construct view components and invoked asynchronously. We'll talk about View Components more in Chapter 18, "Reusable Components," which is important in certain scenarios to keep performance in check on the site. Complex generated views and partials that interact with services are examples of this, which we'll talk about later in this chapter.

Before users can get the output of a view, and in order for you to load any kind of model into the view engine, we must talk a little bit about Controllers in your project.

Controllers (...and Actions!)

Controllers are the traffic cops of MVC applications, ensuring the right types of bits travel to and from the correct places. Controllers typically inherit from the base `Controller` class, but if you don't need the functionality of the base class, you can also use the convention of ending your class name with "Controller," such as in `SeasonPassController`.

The default convention assumes that you are putting your controllers in a folder called "Controllers" in the root of the project. This is no longer required because Core MVC actually does an assembly scan using the naming and inheritance conventions, but it's still a recommended practice to organize your controllers in a recognized way. This helps other developers, including the future version of yourself, to easily manage and maintain the code base down the road.

As software developers, we use controllers as a container for related sets of handlers for incoming requests. These handlers are called actions and are implemented as methods in our controller class. Each method, or action, can accept zero or more parameters that are automatically filled in by the model binding step in the execution pipeline if they are presented by the incoming request.

As the authors of these "traffic cops," our goal is to code our controllers using some well-accepted practices. The primary responsibility of an action is to process a request, validating the incoming parameters and creating an appropriate response.

From time to time, this also requires creating or requesting an instance of a model class, or producing an appropriate HTTP status code based response. You should try to avoid having any business logic

in your controller, which is the responsibility of your model or other components, as well as keeping data access or external calls out of your actions, which should be part of your application services. This is represented in a high level in Figure 3-2.

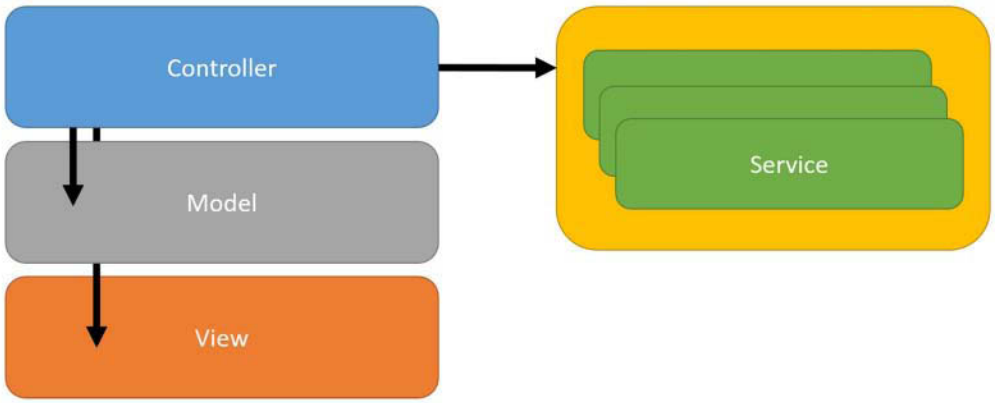

FIGURE 3-2 An illustration showing how controllers are responsible for invoking business logic that helps to generate an appropriate HTTP response

Keeping these services external might seem to make things more complex, or raise questions like, "Who will create these services for me?" This is a great question and one that we'll answer in the "Dependency Injection" section later in this chapter.

It's Not Just About MVC

As discussed earlier, there's actually a lot more going on than just the models, views and controllers themselves in your solution. We'll continue to explore these throughout the book, but here are some important ideas to have in the peripheral as you develop.

Middleware

Here is the secret about middleware in ASP.NET Core MVC: it's pretty much all middleware. All of it! During application start-up you have the opportunity to load your configuration, configure your services, and then configure the request pipeline, which is where the concept of middleware is called into play. You can see this in the `Configure` method of the `Startup` class in the default project template.

Often, the description of middleware and the interpretation by the reader overcomplicates a fairly simple premise. The purpose of middleware is to allow an application to say, "Have each request processed by the following components in the order that I specify." Middleware is a simplification over

previous incarnations of similar functionality in ASP.NET, namely HTTP Modules and Handlers. Middleware replaces both with a common approach in a fashion that is easier to implement.

There are several pieces of middleware that ship publically to handle most scenarios that you need to cover during the execution of your application, both in lower level environments such as staging and QA, as well as in production:

- **Diagnostics:** Provides exception handling and runtime helpers like database error pages and technical details to developers.
- **Static files:** Allows a short-circuit of the request pipeline to return a file from disk.
- **Identity and Authentication:** Allows applications to protect end points and assets of an application.
- **Routing:** Resolves which controller and action should be executed based on the incoming path and parameters.
- **CORS:** Handles injecting the correct headers for cross-origin resource sharing.
- **MVC itself:** Usually at the end of the configured middleware pipeline as it consumes requests.

Each middleware component has the option to execute code before and after the next component in the chain, or to short-circuit the execution and return a response. The name middleware likely comes from the idea that you can execute a piece of software in the middle of something else, as shown in Figure 3.3. In this instance, you see a series of different requests that are handled by the various middleware components in the default project template. In some cases, the request is handled by the static files middleware, returning an asset in `wwwroot`. At other times, the request is allowed to pass all the way through to the MVC execution pipeline where your controller is created and you can return a view.

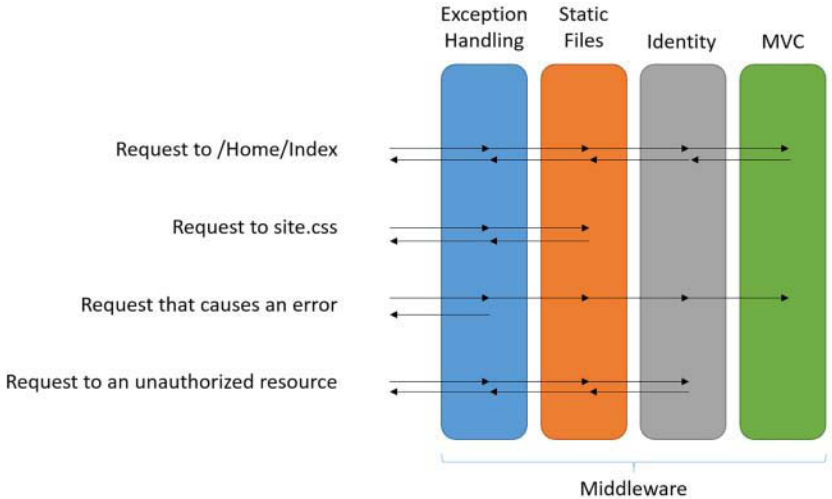

FIGURE 3-3 An illustration showing examples of different request types as handled by middleware

You can bring in other middleware from third parties, additional helpers that are provided by Microsoft, or you can write your own to handle cross-cutting concerns that are needed throughout your application. The middleware pipeline can also be branched based on paths or predicates to allow dynamic, flexible rules around processing requests.

Dependency Injection

There are many written volumes covering dependency injection, but we'll recap the basics here for completeness.

Generally speaking, it's likely going to be a good idea for your code to be obvious about the dependencies that it takes on. In C# you tend to do this by putting the components and services you need in your constructor, such that any creator of the class has to provide your dependencies for you.

Let's consider the constructor of the `HomeController` class in Listing 3-3. The class requires that any time it is being created an instance of an `ILogger` implementation would be provided for it.

LISTING 3-3 The `HomeController` Class Constructor

```
public class HomeController{
  ILogger _logger
  public HomeController (ILogger logger)
  {
    _logger = logger;
  }
}
```

`HomeController` doesn't need to know how to configure or create an `ILogger`, it doesn't need to know where to log to, or how it is to be done. But from any point after instantiation, `HomeController` is now able to add valuable information to your log files as required. This one simple parameter on the constructor explicitly defines your requirements and is referred to as the Explicit Dependencies Principle.

For this controller to be created by the pipeline, you need to have something in the runtime aware of how to resolve the `ILogger` requirement. You configure these services and components in a container, and then these types of dependencies are injected for you into the constructors at runtime. And voila, dependency injection! Being a broader topic, and also by virtue of ASP.NET Core MVC introducing some new ideas for Dependency I njection (DI), we're going to take a deeper look at the idea of

inverting control in Chapter 14, "Dependency Injection," where we'll also explore what is required when replacing the default container.

Other Gems

ASP.NET Core MVC contains some great improvements over previous versions and we are going to explore them throughout the book.

- **Configuration and Logging:** Long considered afterthoughts in the .NET space, these critical application aspects have been revamped, simplified, and made into first-class citizens. Read more about these in Chapter 12, "Configuration and Logging."
- **Tag Helpers:** Along with a few other aspects of simplifying front end development, in Chapter 18, "Reusable Components," we'll examine Tag Helpers and how they more closely resemble the HTML we're trying to return to the client.
- **Identity:** A user is more than just a name and a password. In Chapter 13, "Identity, Security & Rights," we'll explore the new features, uses, and components around security and rights management in ASP.NET Core MVC.

Summary

Each iteration of the MVC Framework has helped shape part of what it's become today. Some lessons learned along the way have helped to bring better assets to developers and the models, views, and controllers we have at the heart of our applications today take up only a small part of our development efforts.

> Before they knew it, the morning had escaped them and Danielle had all but plastered the whiteboard with dry erase marker. Adrian flopped back in his chair and said, "Geez, Danielle. Someone should write a book on this."

This page intentionally left blank

Index

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

S

T

U

V

W

X

Y