TRAINING & REFERENCE

murach's ASP.NET Core MVC

2nd Edition

(Chapter 1)

Thanks for downloading this chapter from ***Murach's ASP.NET Core MVC (2nd Edition)***. We hope it will show you how easy it is to learn from any Murach book, with its paired-pages presentation, its "how-to" headings, its practical coding examples, and its clear, concise style.

To view the full table of contents for this book, you can go to our **website**. From there, you can read more about this book, you can find out about any additional downloads that are available, and you can review our other books on related topics.

Thanks for your interest in our books!

Mike Murach & Associates, Inc.
1-800-221-5528 • (559) 440-9071 • Fax: (559) 440-0963
murachbooks@murach.com • **www.murach.com**

What developers have said about the previous edition

"Not only is this a great book on ASP.NET Core MVC, but it's the best programming book I've read on any subject. I love the format with the paired pages. The code is error-free and the numerous example applications are helpful to study. The explanations are clear and avoid some of the overly complicated word salads other books like to give."

Posted at an online bookseller

"I normally don't write in or leave reviews on products, but I felt compelled to do so with this book as it's the first one I went through that was actually fun from start to finish (and I've read many, many programming books before)."

Shannon Fairchild, Senior Software Developer,
Kingston, Ontario, Canada

"I have been programming in ASP.NET Webforms and MVC for over 15 years and this book is hands-down the best book programming book I've ever purchased."

Posted at an online bookseller

"This book is a solid introduction into ASP.NET Core and can really help a beginner get up to speed in this framework. You can get a lot out of doing the exercises."

Posted at an online bookseller

"Your launch of the ASP.NET Core MVC text is a much-needed contribution to the topic of ASP.NET Core."

K.S., College Instructor, Illinois

"At last, a dot net MVC book that explains everything clearly!"

Posted at an online bookseller

Section 1

Get off to a fast start

This section presents the essential skills for coding, testing, and debugging ASP.NET Core MVC (Model-View-Controller) web apps. After chapter 1 introduces you to the concepts and terms that you need to know for ASP.NET Core MVC programming, chapter 2 shows you how to develop a simple web app that consists of a single page. This includes writing the C# code for the model and controller classes as well as writing the HTML, CSS, and Razor code for the view files that are used to generate the user interface and present it to the user. That gets you off to a fast start!

Chapter 3 shows you how to use the CSS classes that are available from an open-source library known as Bootstrap to make an ASP.NET Core MVC app work well for all screen sizes. This is known as responsive design. Then, chapter 4 shows you how to develop a three-page Movie List app that works with data in a database. Finally, chapter 5 shows you how to manually test and debug ASP.NET Core MVC apps.

When you finish all five chapters, you'll be able to develop real-world apps of your own. You'll have a solid understanding of how ASP.NET Core MVC works. And you'll be ready for rapid progress as you read the other sections of the book.

1

An introduction to web programming and ASP.NET Core MVC

This chapter introduces you to the basic concepts of web programming and ASP.NET Core MVC. Here, you'll learn how web apps work, how ASP.NET Core MVC apps work, and what software you need for developing these apps. When you finish this chapter, you'll have the background that you need for learning how to develop ASP.NET Core MVC apps.

An introduction to web apps

A *web app*, also known as a *web application*, consists of a set of *web pages* that are generated in response to user requests. The internet has many different types of web apps such as search engines, online stores, news sites, social sites, music streaming, video streaming, and games.

The components of a web app

The diagram in figure 1-1 shows that web apps consist of *clients* and a *web server.* The clients are the computers and mobile devices that use the web apps. They often access the web pages through programs known as *web browsers.* The web server holds the files that generate the pages of a web app.

A *network* is a system that allows clients and servers to communicate. The *internet* is a large network that consists of many smaller networks. In a diagram like the one in this figure, the "cloud" represents the network or internet that connects the clients and servers.

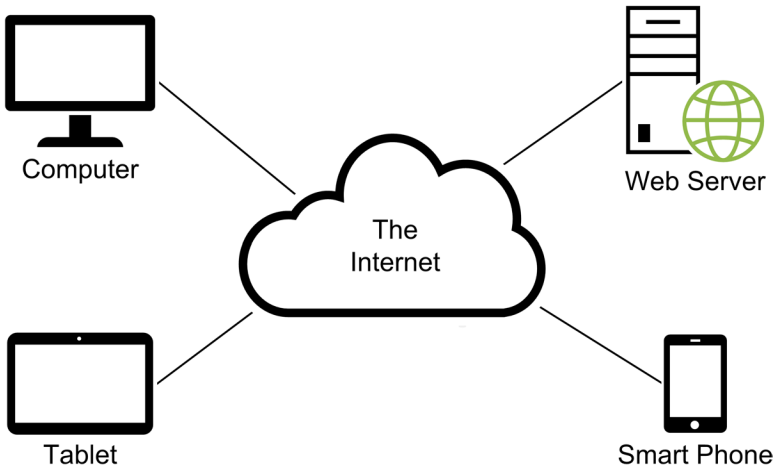
Networks can be categorized by size. A *local area network* (*LAN*) is a small network of computers that are near each other and can communicate with each other over short distances. Computers in a LAN are typically in the same building or adjacent buildings. This type of network is often called an *intranet*, and it can run web apps that are used throughout a company.

By contrast, a *wide area network* (*WAN*) consists of multiple LANs that have been connected. To pass information from one client to another, a router determines which network is closest to the destination and sends the information over that network. A WAN can be owned privately by one company or it can be shared by multiple companies.

An *internet service provider* (*ISP*) is a company that owns a WAN that is connected to the internet. An ISP leases access to its network to companies that need to be connected to the internet. When you develop production web apps, you will often implement them through an ISP.

To access a web page from a browser, you can type a *URL* (*Uniform Resource Locator*) into the browser's address area and press Enter. The URL starts with the *protocol*, which is usually HTTPS. It is followed by the *domain name* and the folder or directory *path* to the file that is requested. If the filename is omitted from the URL, the web server looks for a default file in the specified directory. The default files usually include index.html, index.htm, default.html, and default.htm.

The components of a web app



The components of an HTTP URL

```
https://www.murach.com/shop-books/web-development-books/index.html
```

protocol | domain name | path | filename

Description

- A *web app*, also known as a *web application*, consists of clients, a web server, and a network.
- The *clients* often use *web browsers* to request web pages from the web server. Today, the clients are often computers, smart phones, or tablets.
- The *web server* returns the pages that are requested to the browser.
- A *network* connects the clients to the web server.
- To request a page from a web server, the user can type the address of a web page, called a *URL (Uniform Resource Locator)*, into the browser's address area and then press the Enter key.
- A URL consists of the *protocol* (usually, HTTPS), *domain name*, *path*, and *filename*. If you omit the protocol, HTTPS is assumed. If you omit the filename, the web server typically looks for a file named index.html, index.htm, default.html, or default.htm.
- An *intranet* is a *local area network* (*LAN*) that connects computers that are near each other, usually within the same building.
- The *internet* is a network that consists of many *wide area networks* (*WANs*), and each of those consists of two or more LANs.
- The *cloud* refers to software and services that run on the internet instead of locally on your computer. This term implies that you don't have to understand how it works to be able to use it.
- An *internet service provider* (*ISP*) owns a WAN that is connected to the internet.

Figure 1-1 The components of a web app

How static web pages are processed

A *static web page* is stored in a file that contains hard-coded content that doesn't change each time it is requested. This type of web page is sent directly from the web server to the web browser when the browser requests it. Static web pages were common in the early days of web programming but are less common today.

The diagram in figure 1-2 shows how a web server processes a request for a static web page. This process begins when a client requests a web page in a web browser. To do that, the user can either enter the URL of the page in the browser's address bar or click a link in the current page that specifies the next page to load.

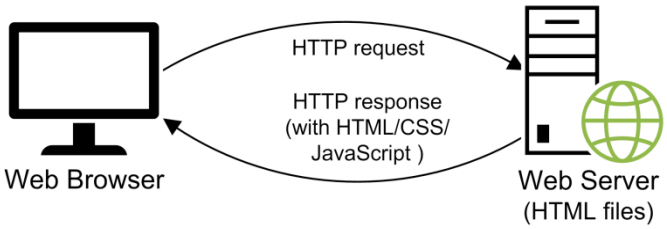
In either case, the web browser builds a request for the web page and sends it to the web server. This request, known as an *HTTP request*, is formatted using the *Hypertext Transfer Protocol* (*HTTP*), which lets the web server know which file is being requested. To keep the communication over the network secure, most web apps use an extension of HTTP known as *Hypertext Transfer Protocol Secure* (*HTTPS*).

When the web server receives the HTTP request, it retrieves the requested file from the disk drive. This file contains the *HTML* (*Hypertext Markup Language*) for the requested page with references to any CSS or JavaScript files needed by the page. Then, the web server sends the HTML, CSS, and JavaScript back to the browser as part of an *HTTP response*.

When the browser receives the HTTP response, it *renders* (translates) the HTML, CSS, and JavaScript into a web page that is displayed in the browser. Then, the user can view the content. If the user requests another page, either by clicking a link or entering another URL into the browser's address bar, the process begins again.

In this figure, both the HTTP request and response use HTTP version 1.1. However, almost all browsers today support HTTP version 2, which is more efficient and secure than HTTP/1.1. HTTP/2 requests and responses are sent in binary, though, so they don't appear as shown here.

How a web server processes a static web page



A simple HTTP request

```
GET / HTTP/1.1
Host: www.example.com
```

A simple HTTP response

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Length: 136

<html>
<head>
    <title>Example Web Page</title>
</head>
<body>
    <p>This is a sample web page</p>
</body>
</html>
```

Three protocols that web apps depend upon

- *Hypertext Transfer Protocol* (*HTTP*) is the protocol that web browsers and web servers use to communicate. It sets the specifications for HTTP requests and responses.
- *Hypertext Transfer Protocol Secure* (*HTTPS*) is an extension of the Hypertext Transfer Protocol (HTTP). It is used for secure communication over a network.
- *Transmission Control Protocol/Internet Protocol* (*TCP/IP*) is a suite of protocols that let two computers communicate over a network.

Description

- *Hypertext Markup Language* (*HTML*) is used to design the pages of a web app.
- A *static web page* is built from an HTML file that's stored on the web server and doesn't change. The filenames for static web pages usually have .htm or .html extensions.
- When the user requests a static web page, the browser sends an *HTTP request* to the web server that includes the name of the file that's being requested.
- When the web server receives the request, it retrieves the HTML, CSS, and JavaScript for the requested file and sends it back to the browser as part of an *HTTP response*.
- When the browser receives the HTTP response, it *renders* the HTML, CSS, and JavaScript into a web page that is displayed in the browser.

Figure 1-2 How static web pages are processed

How dynamic web pages are processed

A *dynamic web page* is a page that's generated by a program on an *application server*. This program uses the data that's sent with the HTTP request to generate the HTML that's returned to the server. For example, if the HTTP request includes a product code, the program can retrieve the data for that product from a *database server*. Then, it can insert that data into a web page and return it as part of an HTTP response.

The diagram in figure 1-3 shows how a web server processes a dynamic web page. The process begins when the user requests a page in a web browser. To do that, the user can click a link that specifies the dynamic page to load or click a button that submits a form that contains the data that the dynamic page should process.

In either case, the web browser builds an HTTP request and sends it to the web server. This request includes whatever data the app needs for processing the request. If, for example, the user has entered data into a form, that data is included in the HTTP request.

When the web server receives the HTTP request, the server examines the request to identify the application server that should process the request. The web server then forwards the request to that application server.

Next, the application server retrieves the appropriate program. It also loads any form data that the user submitted. Then, it executes the program. As the program executes, it generates the HTML, CSS, and JavaScript for the web page. If necessary, the program also requests data from a database server and uses that data as part of the web page it is generating.

When the program is finished, the application server sends the dynamically generated HTML back to the web server. Then, the web server sends the HTML, CSS, and JavaScript for the page back to the browser in an HTTP response.

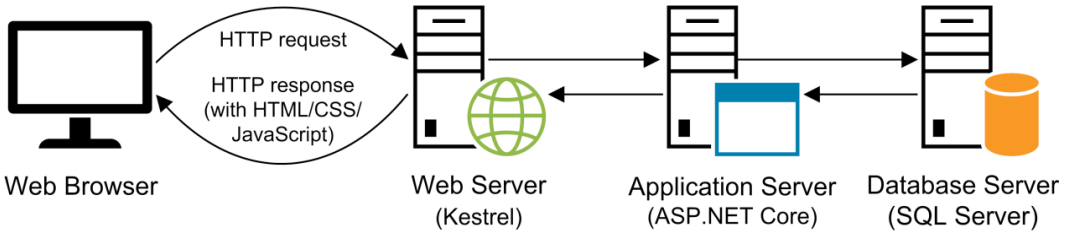
When the web browser receives the HTTP response, it renders the HTML/CSS/JavaScript and displays the web page. To do that, the web browser doesn't need to know whether this HTML is for a static page or a dynamic page. Either way, it just renders the HTML/CSS/JavaScript.

When the page is displayed, the user can view the content. Then, when the user requests another page, the process begins again. The process that begins with the user requesting a web page and ends with the server sending a response back to the client is called a *round trip*.

ASP.NET Core apps use ASP.NET Core as the application server. In addition, they typically use a cross-platform web server known as *Kestrel* and a *DBMS* (*database management system*) named Microsoft SQL Server running on the database server.

Prior to ASP.NET Core, ASP.NET apps typically used a Windows-only web server known as *Internet Information Services* (*IIS*). And, as you would expect, they used ASP.NET as the application server.

How a web server processes a dynamic web page



Description

- A *dynamic web page* is a web page that's generated by a program running on a server.
- When a web server receives a request for a dynamic web page, it examines the request to determine what *application server* should be used to process it and then passes the request to that application server.
- When the application server receives the request, it runs the appropriate program. Often, this program uses data that's sent in the HTTP request to get related data from a *database management system* (*DBMS*) running on a *database server*.
- When the application server finishes processing the data, it generates the HTML, CSS, and JavaScript for a web page and returns it to the web server. Then, the web server returns the HTML, CSS, and JavaScript to the web browser as part of an HTTP response.
- The process that starts when a client requests a page and ends when the page is returned to the browser is called a *round trip*.
- *Kestrel* is a cross-platform web server that also functions as an application server for ASP.NET Core apps.
- *Internet Information Services* (*IIS*) is an older Windows-only web server that also functions as an application server for older ASP.NET apps.
- SQL Server is typically used as the database server for ASP.NET Core and ASP.NET apps.

Figure 1-3 How dynamic web pages are processed

An introduction to the MVC pattern

The *MVC* (*Model-View-Controller*) *pattern* is commonly used to structure web apps that have significant processing requirements. Most modern web development frameworks today use some form of the MVC pattern. As you'll learn in this book, ASP.NET Core provides extensive support for the MVC pattern.

The MVC pattern divides an app into separate components where each component has a specific area of concern. This is known as *separation of concerns*. Although it requires a little more work to set up an app like this, it leads to many benefits. For example, it makes it easier to have different members of a team work on different components, it makes it possible to automate testing of individual components, and it makes it possible to swap out one component for another component. In short, it makes apps easier to code, test, debug, and maintain.

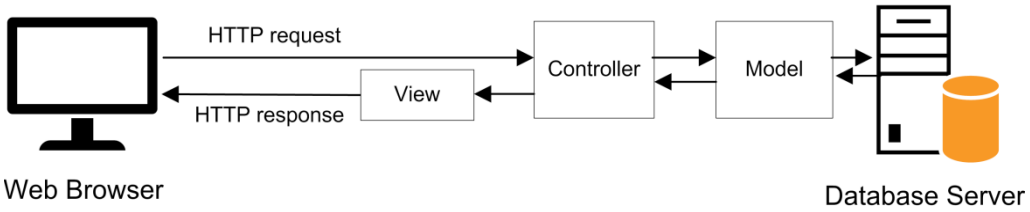
Figure 1-4 presents a diagram that shows the components of the MVC pattern and how they work together. To start, the *model* is in charge of data. Specifically, it gets and updates the data in a data store such as a database, it applies business rules to that data, and it validates that data.

The *view* is in charge of the user interface. Specifically, it creates the HTML, CSS, and JavaScript that the app sends to the browser in response to the browser's HTTP request.

The *controller* is in charge of controlling the flow of data between the model and the view. Specifically, it receives the HTTP request from the browser, determines what data to get from the model, and then sends the data from the model to the appropriate view.

It's important to note that each component should stick to its own area of concern and be as independent as possible. For example, the model should retrieve and validate data but it shouldn't have anything to do with formatting or displaying it. Similarly, the controller should move data between the model and the view but it shouldn't apply any business rules to it.

The MVC pattern



Components of the MVC pattern

- The *model* consists of the code that provides the data access and business logic.
- The *view* consists of the code that generates the user interface and presents it to the user.
- The *controller* consists of the code that receives requests from users, gets the appropriate data from the model, and passes that data to the appropriate view.

Benefits of the MVC pattern

- Makes it easier to have different members of a team work on different components.
- Makes it possible to automate testing of individual components.
- Makes it possible to swap out one component for another component.
- Makes the app easier to maintain.

Drawbacks of the MVC pattern

- Requires more work to set up.

Description

- The *MVC* (*Model-View-Controller*) *pattern* is commonly used to structure web apps that have significant processing requirements.
- The MVC pattern breaks web apps into separate component parts. This is known as *separation of concerns*, and it leads to many benefits including making the app easier to code, test, debug, and maintain.

Figure 1-4 An introduction to the MVC pattern

An introduction to ASP.NET Core MVC

Now that you understand some concepts that apply to most web apps, you're ready to learn some concepts that are more specific to ASP.NET Core MVC apps.

Four ASP.NET programming models for web apps

Since 2002, Microsoft has developed many ASP.NET programming models. Figure 1-5 summarizes four of the most popular. Of these programming models, Web Forms is the oldest and most established, and Core MVC and Razor Pages are the newest and most cutting edge.

In 2002, Microsoft released ASP.NET Web Forms. It provides for *RAD* (*Rapid Application Development*) by letting developers build web pages by working with a design surface in a way that's similar to the way that developers build desktop apps with Windows Forms.

This approach made a lot of sense back in 2002 when many developers were switching from desktop development to web development. However, this approach has many problems including poor performance, inadequate separation of concerns, lack of support for automated testing, and limited control over the HTML/CSS/JavaScript that's returned to the browser. In addition, Web Forms uses the ASP.NET Framework, which is proprietary and only runs on Windows.

In 2007, Microsoft released ASP.NET MVC. It uses the MVC pattern that's used by many other web development platforms. It fixes many of the perceived problems with web forms to provide better performance, separation of concerns, support for automated testing, and a high degree of control over the HTML/CSS/JavaScript that's returned to the browser. However, it uses the same proprietary, Windows-only ASP.NET Framework as Web Forms. As a result, it can't run on internet servers that use other operating systems such as Linux, which is widely used for internet servers.

In 2015, Microsoft released ASP.NET Core MVC and ASP.NET Core Razor Pages. ASP.NET Core MVC uses an MVC service to implement the MVC pattern. It provides all of the functionality of ASP.NET MVC, but with better performance, more modularity, and cleaner code. In addition, it's built on the new ASP.NET Core platform that's open source and can run on multiple platforms including Windows, macOS, and Linux. As a result, it can run on most internet servers. Razor Pages is a model that's built on top of ASP.NET Core MVC.

While developing the different versions of MVC and Core MVC, Microsoft used different names and version numbers. For example, ASP.NET Core 1.0 was originally going to be called ASP.NET 5. But then Microsoft decided ASP.NET Core was a fundamentally different technology that should have a new name and number. Information on the internet sometimes uses the old names and version numbers that were used during development, not the official names and numbers

ASP.NET Web Forms

- Released in 2002.
- Provides for *RAD* (*Rapid Application Development*). Lets developers build web pages by working with a design surface in a way that's similar to Windows Forms.
- Has many problems including poor performance, inadequate separation of concerns, lack of support for automated testing, and limited control over the HTML/CSS/JavaScript that's returned to the browser.
- Uses the ASP.NET Framework, which is proprietary and only runs on Windows.

ASP.NET MVC

- Released in 2007.
- Uses the MVC pattern that's used by many other web development platforms.
- Fixes many of the perceived problems with web forms to provide better performance, separation of concerns, support for automated testing, and a high degree of control over the HTML/CSS/JavaScript that's returned to the browser.
- Uses the same proprietary, Windows-only ASP.NET Framework as Web Forms.

ASP.NET Core MVC

- Released in 2015.
- Uses a service to implement the MVC pattern that's used by many other web development platforms.
- Provides all of the functionality of ASP.NET MVC but with better performance, more modularity, and cleaner code.
- Is built on the open-source ASP.NET Core platform that can run on multiple platforms including Windows, macOS, and Linux.

ASP.NET Core Razor Pages

- Provides the same features as ASP.NET Core MVC, but accesses those features using a model that's built on top of MVC.

Description

- Since 2002, Microsoft has developed many ASP.NET programming models. Of these programming models, ASP.NET Core MVC and ASP.NET Core Razor Pages are the newest.
- While developing the different versions of MVC and Core MVC, Microsoft used different names and version numbers.
- Information on the internet sometimes uses the old names and version numbers that were used during development, not official names and numbers of the final release.
- ASP.NET Core has been stable since version 3.1, so most of the code for newer versions of ASP.NET Core should be compatible with version 3.1.

Figure 1-5 Four ASP.NET programming models for web apps

of the final release. So, be aware of this when you search for information about ASP.NET on the internet.

Although there were breaking changes between ASP.NET and ASP.NET Core and different versions of ASP.NET Core such as between 2.0 and 3.0, ASP.NET Core has been stable since version 3.1. As a result, most of the code for newer versions of ASP.NET Core should be compatible with code for ASP.NET Core 3.1.

Some web components of .NET Framework and .NET Core

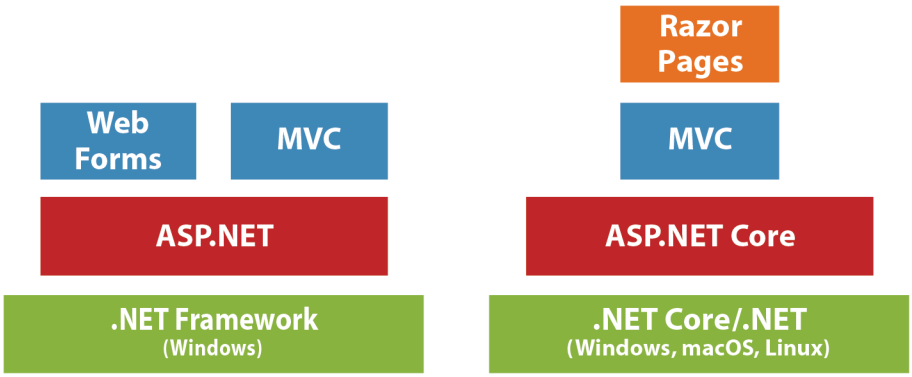
Figure 1-6 shows some web components of the older *.NET Framework*. In addition, it shows some components of the newer *.NET Core* platform, now just called *.NET*.

There are several important differences between the .NET Framework and .NET. Most importantly, .NET is open source and supports multiple operating systems including Windows, macOS, and Linux. By contrast, the .NET Framework is proprietary and only supports the Windows operating system. If you want to develop Web Forms applications, of course, you have to use the .NET Framework. But for new program development, we don't recommend that. Instead, we recommend using ASP.NET Core MVC directly or using Razor Pages. That will result in improved performance and greater flexibility in where you can deploy your app.

In this book, you'll learn how to use the MVC model to develop ASP.NET Core MVC apps. This model is older and more established than the Razor Pages model, and many websites already use it. As a result, it's a skill that's likely to remain valuable for many years to come. However, the Razor Pages model is a newer approach that some programmers prefer. So, if you're developing a new website from scratch, you may want to learn more about Razor Pages before deciding which model to use.

Before going on, you should realize that the first version of .NET, .NET 5, was released in November of 2020. This was the latest release of .NET Core after 3.1. It was named .NET 5 to distinguish it from version 4.8 of the .NET Framework. In addition, Core was dropped from its name to indicate that it will be the main platform going forward. Since then, .NET 6 has been released, and it is the current version as of this writing. Note that to distinguish .NET from the .NET Framework, we sometimes refer to .NET as .NET Core in this book.

Some components of .NET Framework and .NET Core (.NET)



Description

- The *.NET Framework* only supports the Windows operating system.
- The *.NET Core* platform, now just called *.NET*, is open source and supports multiple operating systems including Windows, macOS, and Linux.
- ASP.NET Web Forms apps use services of the ASP.NET Framework, which uses services of the .NET Framework.
- ASP.NET MVC apps work by using services of the ASP.NET Framework.
- ASP.NET Core MVC apps work by using services of ASP.NET Core, which uses services of .NET.
- Although you can use MVC directly to develop ASP.NET Core MVC apps as shown in this book, another option is to use Razor Pages. When you use Razor Pages to develop ASP.NET Core apps, they use MVC under the hood.
- In November of 2020, Microsoft released a new version of .NET Core, called .NET 5. The previous version of .NET Core was 3.1, but version 4 was skipped so it would not be confused with .NET Framework 4.8.
- In November of 2021, Microsoft released .NET 6, which is the current version as of this writing.
- Although ASP.NET Core is based on .NET, it's name still includes "Core" to distinguish it from ASP.NET MVC.

Figure 1-6 Some web components of .NET Framework and .NET Core

An introduction to ASP.NET Core middleware

An ASP.NET Core app allows you to configure the *middleware* components that are in the HTTP request and response *pipeline*. This gives developers a lot of flexibility in how an app works.

Each middleware component can modify the HTTP request before passing it on to the next component in the pipeline. Similarly, each middleware component can modify the HTTP response before passing it to the next component in the pipeline.

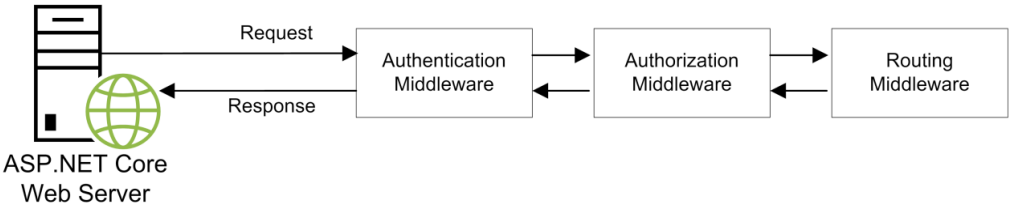
The diagrams in figure 1-7 should give you an idea of how middleware works. To start, the authentication middleware authenticates the request by confirming the identity of the client making the request. To do that, it may need to edit the content of the request. Then, it passes the request on to the authorization middleware.

If the authorization middleware determines that the client is authorized to make the request, it passes the request on to the routing middleware. This middleware uses a controller to generate the content for a response by working with the model and view layers. Then, it passes the response back to the web server. Along the way, the authorization and authentication middleware can edit the content of the response.

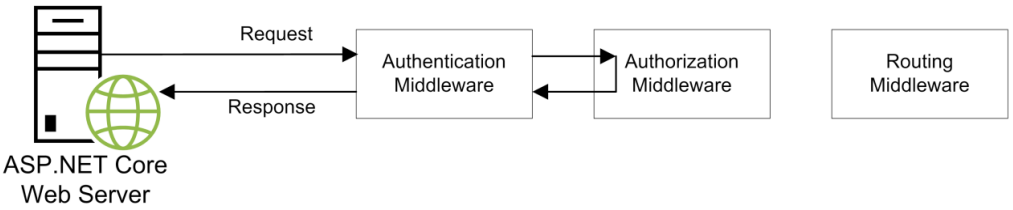
If the authorization middleware determines that the client is not authorized to make the request, it short circuits the request by passing a response back to the web server. Along the way, the authentication middleware can edit the content of the response.

An ASP.NET Core app typically has a number of middleware components configured such as static files middleware, authorization middleware, and routing middleware. Near the end of this chapter, you'll be introduced to the code that configures the routing middleware for a simple ASP.NET Core MVC app. Most of the apps presented in this book configure this middleware. As you progress through this book, you'll learn how to configure additional middleware that's necessary to support the most important features of ASP.NET Core.

A request that makes it through all middleware in the pipeline



A request that's short circuited by a middleware component in the pipeline



Middleware can…

- Generate the content for a response
- Edit the content of a request
- Edit the content of a response
- Short circuit a request

Description

- An ASP.NET Core app allows you to configure the *middleware* components that are in the HTTP request and response *pipeline*.

Figure 1-7 An introduction to ASP.NET Core middleware

How state works in a web app

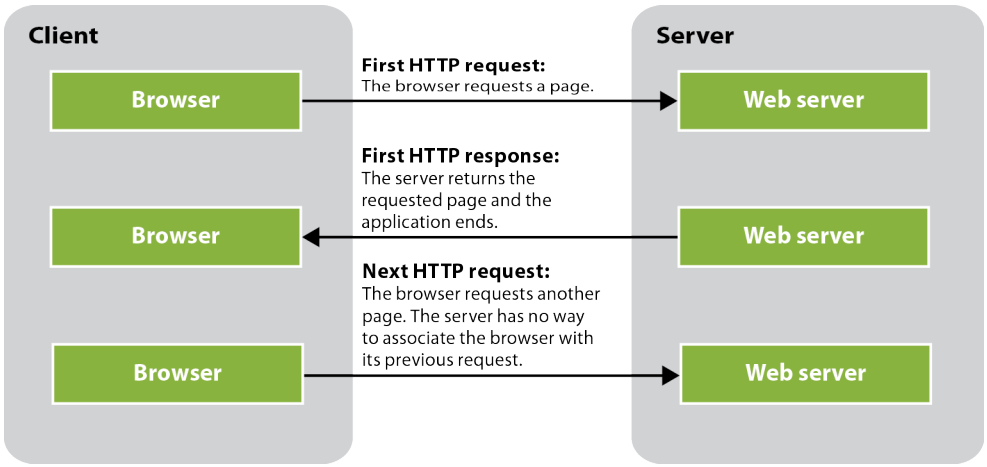
State refers to the current status of the properties, variables, and other data maintained by an app for a single user. As an app runs, it must maintain a separate state for each user currently accessing the app.

Unlike some other protocols, HTTP is a *stateless protocol*. That means that it doesn't keep track of state between round trips. Once a browser makes a request and receives a response, the app terminates and its state is lost as shown by the diagram in figure 1-8. As a result, when the web browser makes a subsequent request, the web server has no way to associate the current request with the previous request.

ASP.NET Web Forms attempted to hide the stateless nature of a web app from developers by automatically maintaining state. This often resulted in large amounts of data being transferred with each request and response, which led to poor performance.

ASP.NET Core MVC does not attempt to hide the stateless nature of a web app from the developer. Instead, it provides features to handle state in a way that gives developers control over each HTTP request and response. When used wisely, this can lead to excellent performance.

Why state is difficult to track in a web app



Concepts

- *State* refers to the current status of the properties, variables, and other data maintained by an app for a single user.
- HTTP is a *stateless protocol*. That means that it doesn't keep track of state between round trips. Once a browser makes a request and receives a response, the app terminates and its state is lost.

Description

- ASP.NET Web Forms attempted to hide the stateless nature of a web app from developers by automatically maintaining state. This led to poor performance.
- ASP.NET Core MVC does not attempt to hide the stateless nature of a web app from developers. Instead, it provides features to handle state in a way that gives developers control over each HTTP request and response.

Figure 1-8 How state works in a web app

Tools for working with ASP.NET Core MVC apps

Now that you know some of the concepts behind ASP.NET Core MVC, you're ready to learn more about the tools that you can use to develop ASP.NET Core MVC apps. Some developers prefer to use an *integrated development environment* (*IDE*), which is a tool that provides all of the functionality that you need for developing an app in one place. Other developers prefer to use a code editor to enter and edit code and a command line to compile and run it. Each approach has its pros and cons. Fortunately, an excellent IDE and code editor are both available for free to ASP.NET Core developers.

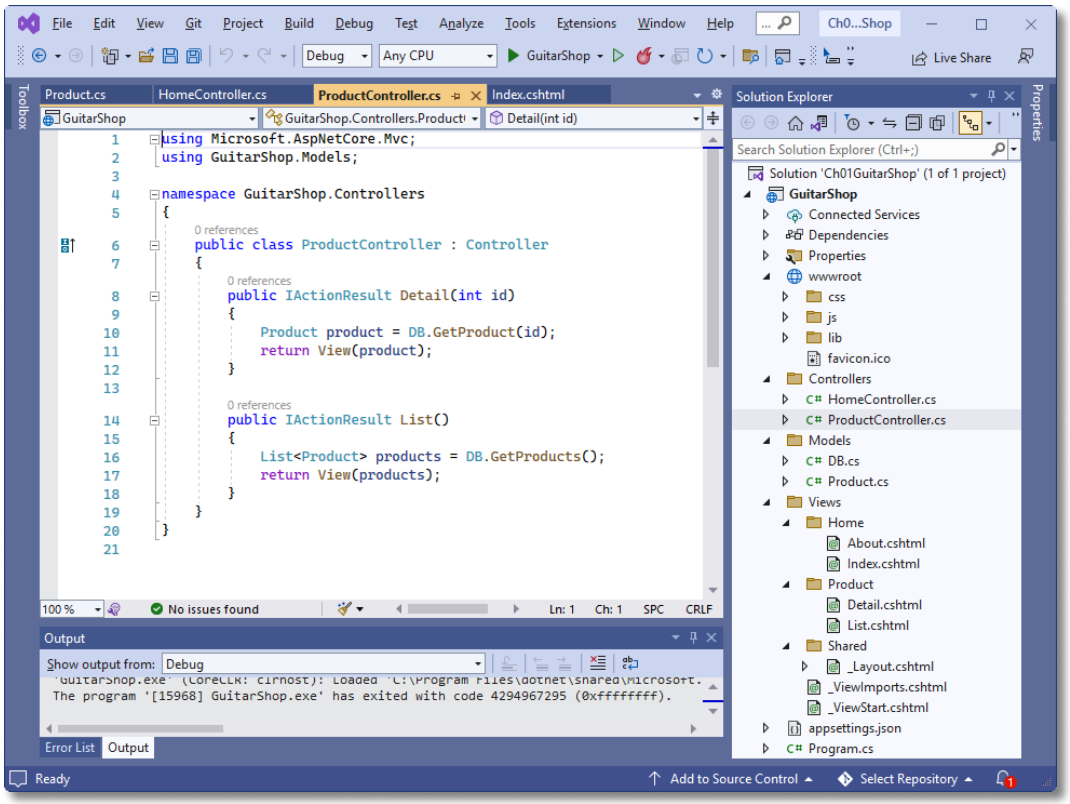
An introduction to Visual Studio

Visual Studio, also known just as *VS*, is the most popular IDE for developing ASP.NET Core apps. Microsoft provides a Community Edition that's available for free and runs on Windows, as well as a free version that runs on macOS.

Figure 1-9 shows Visual Studio after it has opened an ASP.NET Core MVC app and displayed the code for one of its controllers. In addition, this figure lists some of the features provided by Visual Studio. For example, you can compile and run an app with a single keystroke.

Visual Studio is an established product that has been around for decades, and it's a great tool for learning. That's why we present it throughout this book, starting in the next chapter.

Visual Studio with an ASP.NET Core MVC app



Features

- IntelliSense code completion makes it easy to enter code.
- Automatic compilation allows you to compile and run an app with a single keystroke.
- Integrated debugger makes it easy to find and fix bugs.
- Runs on Windows and macOS.

Description

- An *Integrated Development Environment* (*IDE*) is a tool that provides all of the functionality that you need for developing web apps.
- *Visual Studio*, also known as *VS*, is the most popular IDE for ASP.NET Core web development.
- Starting in the next chapter, this book shows how to use Visual Studio to develop ASP.NET Core MVC apps.

Figure 1-9 An introduction to Visual Studio

An introduction to Visual Studio Code

Visual Studio Code, also known as *VS Code,* is a *code editor* that's becoming popular with developers. Like Visual Studio, VS Code can be used to develop all types of .NET apps, including ASP.NET Core apps. Since it doesn't provide as many features as an IDE like Visual Studio, some developers find VS Code easier to use. In addition, VS Code typically starts and runs faster than Visual Studio, especially on slower computers.

Figure 1-10 shows VS Code after it has opened the same app as the previous figure. If you compare these two figures, you'll see that they look very similar. In short, both provide an Explorer window that lets you navigate through the files for an app, and both provide a code editor for editing the code for an app. The main difference is that Visual Studio provides more features than VS Code. That's either good or bad, depending on how you look at it.

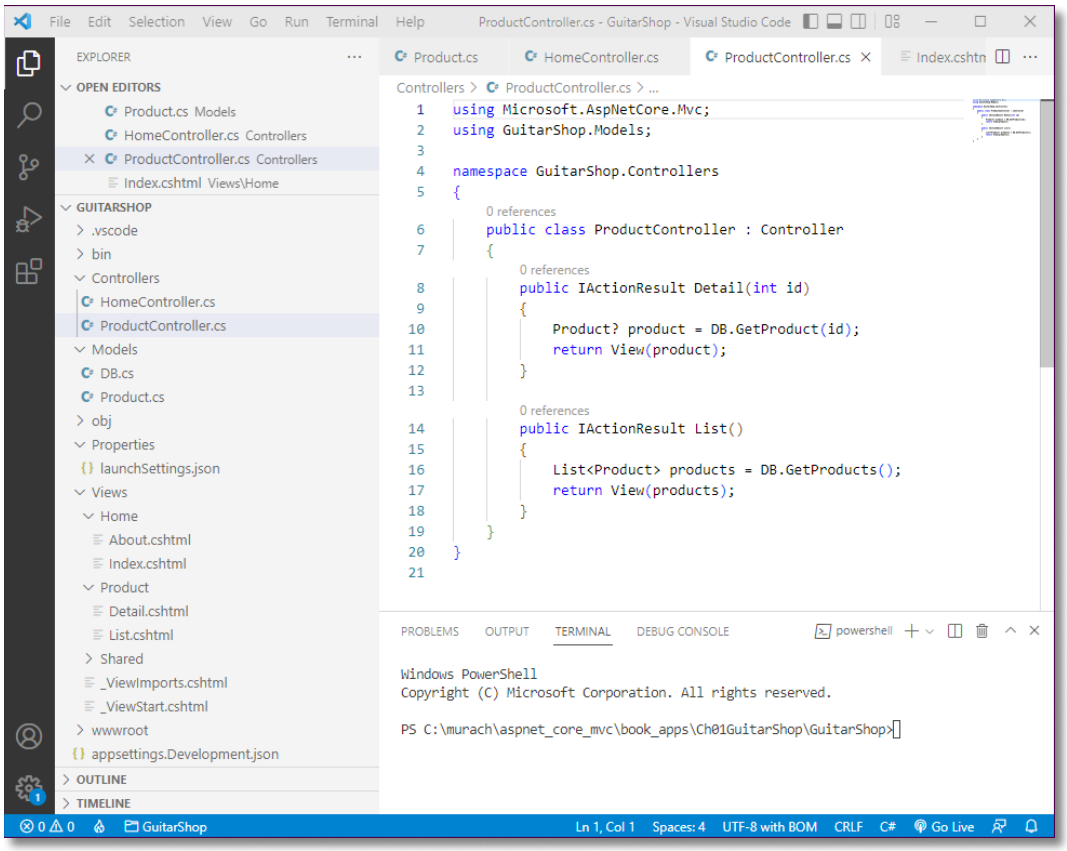
This figure also lists some of the features provided by VS Code. If you compare this list of features with the list of features in the previous figure, you'll see that they're mostly the same. The main difference is that VS Code runs on Linux, and Visual Studio does not. So, even though Visual Studio provides more features, VS Code provides all of the most essential features that make it easy to develop ASP.NET Core apps.

When you use VS Code, you can use its Terminal window to use a *command line* to enter and execute the commands that build and run an app. With Windows, for example, you typically use the command line known as Windows PowerShell. In this figure, the Windows PowerShell command line is displayed across the bottom of the code editor window.

When you choose between Visual Studio and VS Code, you may want to consider that VS Code has a less restrictive license than the Community Edition of Visual Studio and adheres to a truly open-source, open-use model. As a result, you may want or need to use VS Code if licensing for commercial development becomes an issue.

As mentioned in the previous figure, this book shows how to use Visual Studio, starting in the next chapter. However, chapter 18 shows how to use VS Code. If you prefer using a code editor and command line to using a more full-featured IDE, you can use VS Code. Then, you can refer to chapter 18 whenever you need help with a task. This should make it easy to use VS Code with this book.

Visual Studio Code with an ASP.NET Core MVC app



Features

- IntelliSense code completion makes it easy to enter code.
- Automatic compilation allows you to compile and run an app with a single keystroke.
- Integrated debugger makes it easy to find and fix bugs.
- Runs everywhere (Windows, macOS, and Linux).

Description

- *Visual Studio Code*, also known as *VS Code,* is a *code editor* that you can use to work with ASP.NET Core apps.
- When you use VS Code, you can use its Terminal window to use a *command line* to enter and execute the commands that build and run an app.
- VS Code has a less restrictive license than the Community Edition of Visual Studio and adheres to a truly open-source, open-use model.
- Chapter 18 shows how to use VS Code to develop ASP.NET Core apps.

Figure 1-10 An introduction to Visual Studio Code

How an ASP.NET Core MVC app works

Now that you've been introduced to some general concepts and tools for working with ASP.NET Core MVC, you're ready to learn more about how an ASP.NET Core MVC app works. To do that, the next few figures present the overall structure of an ASP.NET Core MVC app as well as some key snippets of code. Then, in the next chapter, you'll learn all of the details for coding such an app.

How coding by convention works

ASP.NET Core MVC uses a software design paradigm known as *convention over configuration*, or *coding by convention*. This reduces the amount of configuration that developers need to do if they follow certain conventions.

That's why figure 1-11 shows some of the folders and files for an MVC web app that follow the standard MVC conventions. And that's why this figure lists some of these conventions.

To start, the top-level folder for a web app is known as its *root folder*, also known as the *root directory*. In this figure, the GuitarShop folder is the root folder. Within the root folder, you typically use .cs files to store the C# classes that define controllers and models.

All controller classes should be stored in a folder named Controllers or one of its subfolders. In this figure, the Controllers folder contains the files for two classes named HomeController and ProductController. Both of these classes have a suffix of "Controller". This isn't required for controllers to work, but it's a standard convention that makes it easy for other programmers to quickly identify these classes as controller classes.

All model classes should be stored in a folder named Models or one of its subfolders. In this figure, the Models folder contains a single file for a model class named Product.

All view files should be stored in a folder named Views or one of its subfolders. In addition, the subfolders of the Views folder should correspond to the controller classes. For example, the Views/Home folder should store the view files for the HomeController class, and the Views/Product folder should store the view files for the ProductController class. These view files (.cshtml files) contain *Razor views* that define views for the app. In this figure, the views for the app are defined by view files named Home/Index.cshtml, Home/About.cshtml, Product/Detail.cshtml, and Product/List.cshtml files.

All static files such as image files, CSS files, and JavaScript files should be stored in a folder named wwwroot. The static files for an app can include CSS libraries such as Bootstrap or JavaScript libraries such as jQuery. In addition, they can include custom CSS or JavaScript files that override the code in these libraries.

Some of the folders and files for a web app

```
GuitarShop
    /Controllers
        /HomeController.cs
        /ProductController.cs
    /Models
        /Product.cs
    /Views
        /Home
            /About.cshthml
            /Index.cshthml
        /Product
            /Detail.cshthml
            /List.cshthml
    /wwwroot
        /css
            site.css
        /images
        /js
            custom.js
        /lib
            /boostrap
            /jquery
     /Program.cs
```

Some naming conventions for an ASP.NET Core MVC app

- All controller classes should be stored in a folder named Controllers or one of its subfolders.
- All model classes should be stored in a folder named Models or one of its subfolders.
- All view files should be stored in a folder named Views or one of its subfolders.
- All static files such as image files, CSS files, and JavaScript files should be stored in a folder named wwwroot or one of its subfolders.
- All controller classes should have a suffix of "Controller".

Description

- ASP.NET Core MVC uses a software design paradigm known as *convention over configuration*, or *coding by convention*. This reduces the amount of configuration that developers need to do if they follow certain conventions.
- The top-level folder for a web app is known as its *root folder* or *root directory*.
- You typically use C# classes (.cs files) to define controllers and models.
- You typically use *Razor views* (.cshtml files) to define views.
- The static files for an app can include CSS libraries such as Bootstrap or JavaScript libraries such as jQuery. In addition, they can include custom CSS or JavaScript files that override the code in these libraries.

Figure 1-11 How ASP.NET Core MVC uses coding by convention

How a controller passes a model to a view

In ASP.NET Core MVC, a *model* is a C# class that defines the data objects and business rules for the app. Figure 1-12 begins by showing the code for a model named Product. It contains three properties named ProductID, Name, and Price. This class is stored in the GuitarShop.Models namespace, and it defines a simple Product object.

In ASP.NET Core MVC, a *controller* is a C# class that typically inherits the Microsoft.AspNetCore.Mvc.Controller class. In this figure, the ProductController class inherits this class and defines two actions. In ASP.NET Core MVC, an *action* is a method of a controller that returns an *action result*. To do that, a method can use the built-in View() method to return a type of action result known as a *view result* that's created by merging the model (if there is one) into the HTML code specified in the view file.

In this figure, for example, the Detail() method is an action. It starts by using the parameter named id to get a Product object that corresponds to that id from the database. This Product object is the model for the view. Then, it passes this model to the view so the view can display its data.

Similarly, the List() method starts by getting the model, which is a List of Product objects, from the database. Then, it passes that List of Product objects to the view for display.

The code for a model class named Product

```
namespace GuitarShop.Models
{
    public class Product
    {
        public int ProductID { get; set; }
        public string Name { get; set; } = string.Empty;
        public decimal Price { get; set; }
    }
}
```

The code for a controller class named ProductController

```
using Microsoft.AspNetCore.Mvc;
using GuitarShop.Models;

namespace GuitarShop.Controllers
{
    public class ProductController : Controller
    {
        public IActionResult Detail(int id)
        {
            Product product = DB.GetProduct(id);
            return View(product);  // passes model to Product/Detail view
        }

        public IActionResult List()
        {
            List<Product> products = DB.GetProducts();
            return View(products); // passes model to Product/List view
        }
    }
}
```

Description

- A *model* is a C# class that defines the data objects and business rules for the app.
- With ASP.NET Core MVC, a *controller* is a C# class that typically inherits the Microsoft.AspNetCore.Mvc.Controller class.
- With ASP.NET Core MVC, an *action* is a method of a controller that returns an *action result*.
- An action method can use the View() method to return a type of action result known as a *view result* that's created by merging the model (if there is one) into the corresponding view file.

Figure 1-12 How a controller passes a model to a view

How a view uses Razor code, tag helpers, and Bootstrap CSS classes

Most of a typical view file consists of HTML elements. In figure 1-13, for example, the code uses an <h1> element to display a level-1 heading. It uses the <table>, <tr>, and <td> elements to display a table that has three rows and two columns. And it uses an <a> element to display a link that's formatted to look like a button.

However, a view file can also contain *Razor code* that allows you to use C# to get data from the model, format it, and display it. Razor code begins with an @ sign. In this figure, all of the Razor code is highlighted. As you can see, there are several different ways to work with Razor code.

To start, the @model directive specifies the class for the model, and the @Model property allows you to access a model object that's created from the specified class. In this figure, the @model directive at the top of the class specifies that this view uses the Product class as the model. Then, it uses the @Model property to get data from the ProductID, Name, and Price properties and insert this data into the HTML elements. In addition, it uses the ToString() method of the Price property to format the number as currency with 2 decimal places.

After the @model directive, there's an @ sign followed by braces ({}) that identifies a block of C# statements. In this figure, the block of statements contains a single statement that uses the built-in ViewData object to set the title of the web page. However, if necessary, you could add other C# statements to this code block.

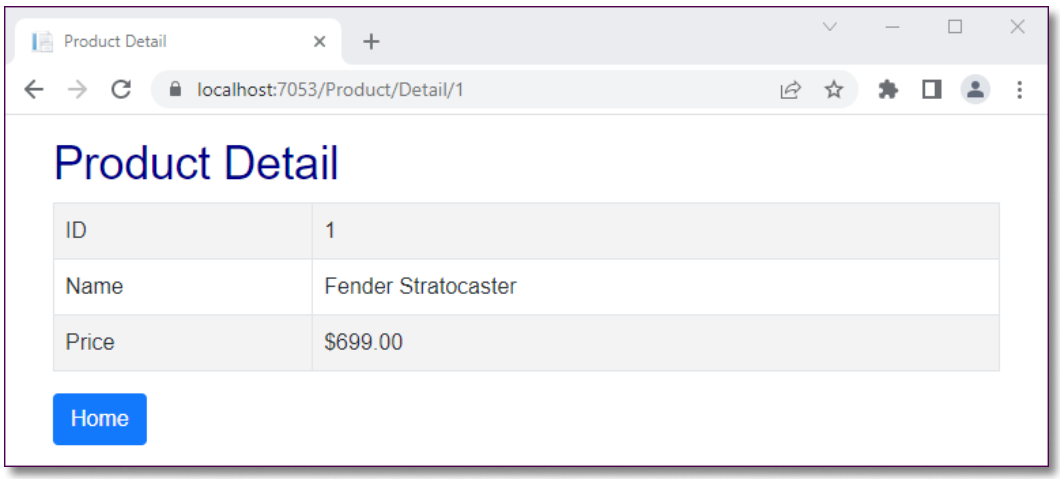
In a view, all HTML attributes that start with "asp-" are *tag helpers* that can make it easier to code HTML attributes. Tag helpers are defined by C# classes, and many are available from ASP.NET Core. As a result, you can use them to make coding HTML attributes easier. In this figure, for example, the <a> element uses the asp-controller and asp-action tag helpers to specify the controller and action method for the link. Alternately, you could code an href attribute that specified a URL to the correct controller and action. However, this isn't as flexible and easy to maintain as using the tag helpers.

In a view, it's common to use the class attribute of an HTML element to specify CSS classes from *Bootstrap*, a popular open-source CSS library that's often used with ASP.NET Core apps. In this figure, for example, the <table> element uses the class attribute to specify three CSS classes: table, table-bordered, and table-striped. That's why the table that's displayed in the browser looks so professional. Similarly, the <a> element uses the class attribute to specify two CSS classes: btn and btn-primary. That's why the link that's displayed in the browser looks like a button.

The code for a view file named Product/Detail.cshtml

```
@model Product
@{
    ViewData["Title"] = "Product Detail";
}
<h1>Product Detail</h1>
<table class="table table-bordered table-striped">
    <tr>
        <td>ID</td><td>@Model.ProductID</td>
    </tr>
    <tr>
        <td>Name</td><td>@Model.Name</td>
    </tr>
    <tr>
        <td>Price</td><td>@Model.Price.ToString("C2")</td>
    </tr>
</table>
<a asp-controller="Home" asp-action="Index"
   class="btn btn-primary">Home</a>
```

The view displayed in a browser



Description

- Most of a typical view file consists of standard HTML elements.
- The @model directive specifies the class for the model, and the @Model property allows you to access the model object that's passed to the view from the controller.
- The @ sign followed by braces ({}) identifies a block of C# statements. Within the block, you can code one or more C# statements.
- All HTML attributes that start with "asp-" are *tag helpers*. Tag helpers are defined by C# classes and make it easier to work with HTML elements. Many tag helpers are built into ASP.NET Core.
- The class attribute of an HTML element can specify CSS classes from *Bootstrap*, a popular open-source CSS library that's often used with ASP.NET Core.

Figure 1-13 How a view uses Razor code, tag helpers, and Bootstrap CSS classes

How the Program.cs file configures the middleware for an app

Figure 1-14 begins by showing the code that's generated for the Program.cs file when you create an ASP.NET Core MVC Web app. This is the code that's used for the Guitar Shop app, and it configures the middleware that's used by the app. This builds the middleware pipeline for the app. You'll learn more about this file in the next chapter, but here are some highlights.

The Program.cs file begins by creating a WebApplicationBuilder object. Then, it contains code that adds services to the app and then builds the app. For a simple ASP.NET Core MVC app like the Guitar Shop app, you only need to use the AddControllersWithViews() method to add services that support controllers, models, views, tag helpers, and so on.

Next, the Program.cs file contains the code that configures the services that have been added. In this figure, the if statement specifies the exception pages that the middleware should display if the page is run in a production environment rather than a development environment. The first statement after that indicates that the app should use a secure connection (HTTPS). The second statement specifies that the app can use the static files that are in the wwwroot directory such as the Bootstrap library. The third statement marks the position in the middleware pipeline where a routing decision is made for a URL. The fourth statement indicates that the app should use authorization middleware. And the fifth statement marks the position in the pipeline where the routing decision is executed.

This fifth statement consists of a MapControllerRoute() method that maps the controllers and their actions to a pattern for request URLs that's known as the default route. This route identifies the Home controller as the default controller and the Index() action method as the default action. As a result, if the request URL specifies the root folder (/), the app executes the Index() action method of the Home controller. However, if a request URL specifies /Product/Detail, the app executes the Detail() action method of the Product controller.

The default route also identifies a third segment of the URL that's a parameter named id, and it uses a question mark (?) to indicate that this parameter is optional. As a result, if a request URL specifies /Product/Detail/1, the app passes an id parameter of 1to the Detail() action method of the Product controller. This allows the Product controller to retrieve data about the product that has an ID of 1.

The Program.cs file

```
var builder = WebApplication.CreateBuilder(args);

// Add services to the container.
builder.Services.AddControllersWithViews();

var app = builder.Build();

// Configure the HTTP request pipeline.
if (!app.Environment.IsDevelopment())
{
    app.UseExceptionHandler("/Home/Error");
    // The default HSTS value is 30 days. You may want to change this
    // for production scenarios, see https://aka.ms/aspnetcore-hsts.
    app.UseHsts();
}

app.UseHttpsRedirection();
app.UseStaticFiles();

app.UseRouting();

app.UseAuthorization();

app.MapControllerRoute(
    name: "default",
    pattern: "{controller=Home}/{action=Index}/{id?}");

app.Run();
```

How request URLs map to controllers and actions by default

Request URL	Controller	Action
http://localhost	Home	Index
http://localhost/Home	Home	Index
http://localhost/Home/About	Home	About
http://localhost/Product/List	Product	List
http://localhost/Product/Detail	Product	Detail

Description

- The Program.cs file contains the code that configures the middleware that's used by the app. In other words, it builds the middleware pipeline for the app.
- The Program.cs file contains the code that identifies which services to use and provides additional configuration if necessary.
- By convention, the routing system identifies the Home controller as the default controller and the Index() action method as the default action.

Figure 1-14 How the Program.cs file configures the middleware for an app

Perspective

Now that you've read this chapter, you should have a general understanding of how ASP.NET Core MVC apps work and what software you need for developing these apps. With that as background, you're ready to gain valuable hands-on experience by learning how to develop an ASP.NET Core MVC app as shown in the next chapter.

Terms

web app
web application
web page
client
web browser
web server
network
URL (Uniform Resource Locator)
protocol
domain name
path
filename
intranet
internet
LAN (local area network)
WAN (wide area network)
cloud
ISP (internet service provider)
HTTP (Hypertext Transfer Protocol)
HTTPS (Hypertext Transfer Protocol Secure)
TCP/IP (Transmission Control Protocol/Internet Protocol)
HTML (Hypertext Markup Language)
static web page
HTTP request
HTTP response
dynamic web page
application server
database management system (DBMS)
database server
round trip
Kestrel
Internet Information Services (IIS)
MVC (Model-View-Controller) pattern
model
view
controller
separation of concerns
RAD (Rapid Application Development)
.NET
.NET Framework
.NET Core
middleware
pipeline
state
stateless protocol
IDE (Integrated Development Environment)
VS (Visual Studio)
VS Code (Visual Studio Code)
code editor
command line
convention over configuration
coding by convention
root folder
root directory
Razor view
action
action result
view result
tag helper
Bootstrap

Summary

- A *web app*, also known as a *web application*, consists of a set of *web pages* that are run by clients, a web server, and a network. *Clients* often use *web browsers* to request web pages from the web server. The *web server* returns the requested pages.
- A *local area network* (*LAN*), or *intranet*, connects computers that are near each other. By contrast, the *internet* consists of many *wide area networks* (*WANs*).
- One way to access a web page is to type a *URL* (*Uniform Resource Locator*) into the address area of a browser and press Enter. A URL consists of the *protocol* (usually, HTTPS), *domain name*, *path*, and *filename*.
- To request a web page, the web browser sends an *HTTP request* to the web server. Then, the web server gets the HTML/CSS/JavaScript for the requested page and sends it back to the browser in an *HTTP response*. Last, the browser *renders* the HTML/CSS/JavaScript into a web page.
- A *static web page* is a page that is the same each time it's retrieved. By contrast, the HTML/CSS/JavaScript for a *dynamic web page* is generated by a server-side program, so its HTML/CSS/JavaScript can change from one request to another. Either way, HTML/CSS/JavaScript is returned to the browser.
- For ASP.NET Core MVC apps, the application server is ASP.NET Core, the web server is usually *Kestrel*, and the database server usually runs a *database management system* (*DBMS*) like SQL Server.
- One way to develop ASP.NET apps is to use Web Forms. This is similar to using Windows Forms and encourages *Rapid Application Development* (*RAD*). Web Forms are supported by the .NET Framework but not by .NET Core.
- Another way to develop ASP.NET apps is to use ASP.NET Core *MVC* (*Model-View-Controller*). It provides better *separation of concerns*, which leads to many benefits including making the app easier to code, test, debug, and maintain.
- The older *.NET Framework* only supports the Windows operating system. The newer *.NET Core* platform, now just called .NET, is open source and supports Windows, macOS, and Linux.
- An ASP.NET Core app allows you to configure the *middleware* components that are in the HTTP request and response *pipeline*.
- HTTP is called a *stateless protocol* because it doesn't keep track of the data (*state*) between *round trips*. However, ASP.NET Core provides features to handle state in a way that gives developers control over each HTTP request and response.

- *Visual Studio*, also known as just *VS*, is an *integrated development environment* (*IDE*) that that provides all of the functionality that you need for developing web apps.
- *Visual Studio Code*, also known as *VS Code,* is a *code editor* that you can use to develop ASP.NET Core apps.
- With ASP.NET Core MVC, you typically use *Razor views* (.cshtml files) to define the user interface and present it to the user.
- A *model* is a C# class that defines the data objects and business rules for the app.
- A *controller* is a C# class that controls the flow of data between the model and the view. A controller typically inherits the Microsoft.AspNetCore.Mvc.Controller class.
- With ASP.NET Core MVC, an *action* is a method of a controller that returns an *action result*. To do that, it's common to use the View() method to return a type of action result known as a *view result* that's typically created by merging the model into the corresponding view file.
- All HTML attributes that start with "asp-" are *tag helpers*. Tag helpers are defined by C# classes and make it easier to work with HTML elements. Many tag helpers are built into ASP.NET Core.
- The class attribute of an HTML element can specify CSS classes from *Bootstrap*, a popular open-source CSS library that's often used with ASP.NET Core.

Before you do the exercises for this book...

Before you do the exercises for this book, you should install the software that's required for this book, and you should download the source code for this book. Appendixes A (Windows) and B (macOS) show how to do that.

Exercise 1-1 Use Visual Studio to run the Guitar Shop app

In this exercise, you'll run the Guitar Shop app. This will test whether you've successfully installed the software and source code for this book.

Start Visual Studio and open the Guitar Shop app

1. Start Visual Studio.
2. From the menu system, select the File→Open→Project/Solution item. Or, if you are in the Start window for Visual Studio, you can click the "Open a project or solution" button. In the dialog box that's displayed, navigate to this folder:

   `/aspnet_core_mvc/book_apps/Ch01GuitarShop`

   Then, select the Ch01GuitarShop.sln file and click the Open button.

Run the Guitar Shop app

3. Press Ctrl+F5 to run the app. That should display the Home page for the Guitar Shop app in Visual Studio's default web browser. If you get messages about trusting and installing an SSL certificate, you can click Yes. And if a web page is displayed indicating that the connection is not private, you can click the link to proceed.
4. Click the "View Fender Stratocaster" link. This should display the Product Detail page for that product.
5. Click the Home button to return to the Home page.
6. Click the "View Products" link. This should display a list of products.
7. Click the View link for the product named Gibson Les Paul. This should display the Product Detail page for that product.
8. Close the browser tab or window for the app, and then switch back to Visual Studio.
9. In the Solution Explorer, expand the Controllers, Models, and Views folders and review some of the code.

Close the Guitar Shop app and exit Visual Studio

10. Use the File→Close Solution command to close the solution.
11. Exit Visual Studio.

Exercise 1-2 Use Visual Studio Code to run the Guitar Shop app (optional)

This exercise is optional. However, if you want to use Visual Studio Code instead of Visual Studio, or if you just want to see how VS Code compares to VS, you can use this exercise to take VS Code for a test drive. But first, you should install and configure VS Code as described in chapter 18.

Start VS Code and open the Guitar Shop app

1. Start VS Code.
2. From the menus, select the File→Open Folder item. In the dialog box that's displayed, navigate to this folder:

   `/aspnet_core_mvc/book_apps/Ch01GuitarShop/GuitarShop`

   Then, click the Select Folder button.

   If the "Do you trust the authors of the files in this folder" dialog is displayed, select Yes. This should open the GuitarShop project that's in this folder.
3. If you get any error messages, click on the appropriate buttons to fix them. VS Code should be able to do this for you.

Run the Guitar Shop app

4. Press Ctrl+F5 to run the app. That should display the Home page for the Guitar Shop app in your default web browser.
5. Click the "View Fender Stratocaster" link. This should display the Product Detail page for that product.
6. Click the Home button to return to the Home page.
7. Click the "View Products" link. This should display a list of products.
8. Click the View link for the product named Gibson Les Paul. This should display the Product Detail page for that product.
9. Close the browser tab or window for the app, and then switch back to VS Code.
10. In the Explorer window, expand the Controllers, Models, and Views folders and review some of the code.

Close the Guitar Shop app and exit VS Code

11. Select the File→Close Folder item to close the folder. This should close the Guitar Shop project that's in this folder.
12. Exit VS Code.

How to build your ASP.NET MVC web programming skills

Mike Murach, Publisher

The easiest way is to let **Murach's ASP.NET Core MVC (2nd Edition)** be your guide! So if you've enjoyed this chapter, I hope you'll get your own copy of the book today. You can use it to:

- Teach yourself how to develop professional, responsive web apps that follow the MVC pattern and work with databases
- Pick up new skills whenever you want to or need to by focusing on material that's new to you
- Look up coding details or refresh your memory on forgotten details when you're in the middle of developing a web app
- Loan to your colleagues who are always asking you questions about ASP.NET Core MVC programming

To get your copy, you can order online at **www.murach.com** or call us at 1-800-221-5528 (toll-free in the U.S. and Canada). And remember, when you order directly from us, this book comes with my personal guarantee:

100% Guarantee

When you buy directly from us, you must be satisfied. Try our books for 30 days or our eBooks for 14 days. They must outperform any competing book or course you've ever tried, or return your purchase for a prompt refund.…no questions asked.

Thanks for your interest in Murach books!