Chapter 1

An introduction to web programming and ASP.NET Core MVC

Objectives (part 1)

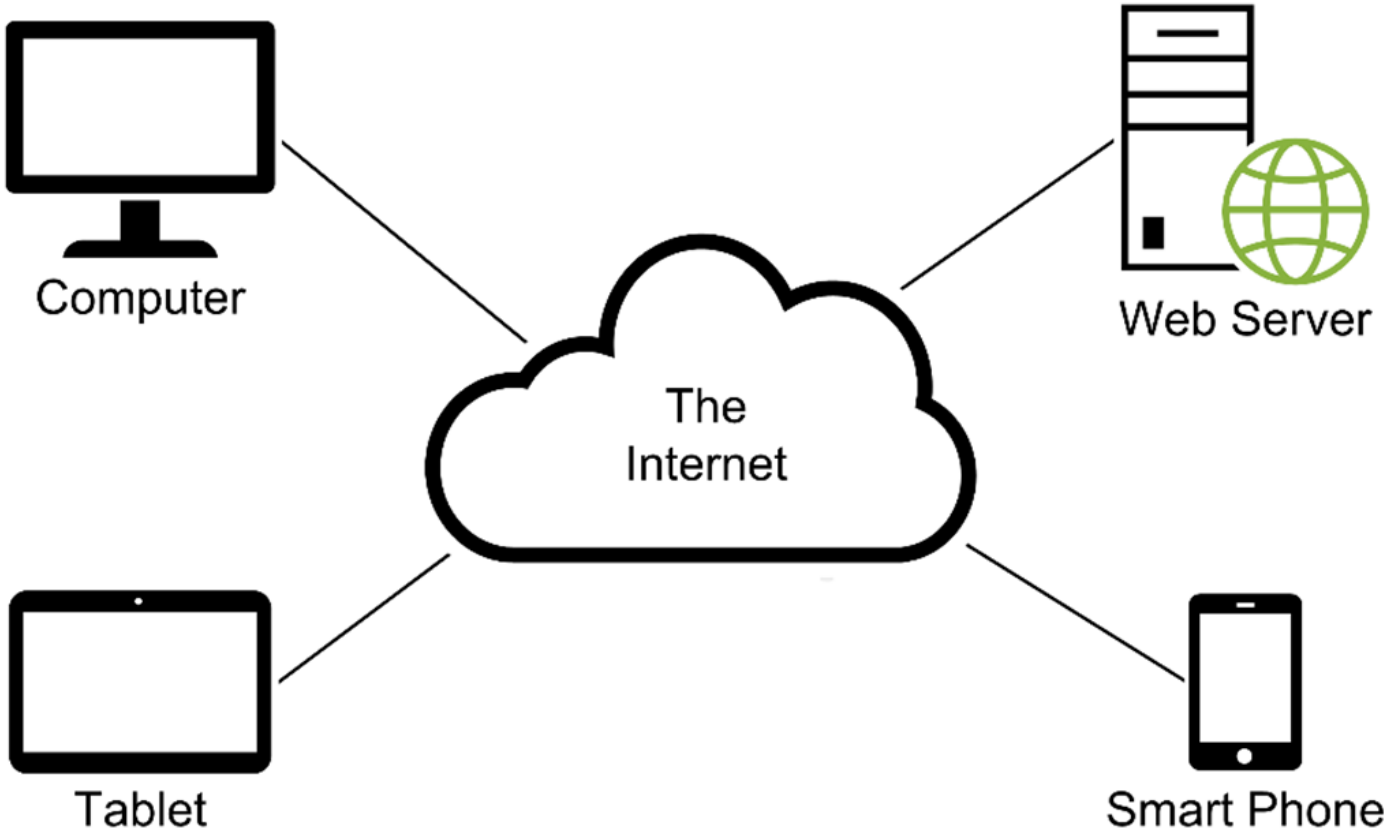
Knowledge

1. Describe the components of a web app.
2. Describe the four components of a URL.
3. Distinguish between static and dynamic web pages, with the focus on the web server, application server, and database server.
4. Distinguish between the internet and an intranet.
5. Describe these terms: HTTP request, HTTP response, and round trip.
6. Describe the model, view, and controller of the MVC pattern.
7. Explain how using the MVC pattern can improve app development.
8. Describe four programming models that can be used for developing ASP.NET apps.
9. Distinguish between .NET Framework and .NET Core.

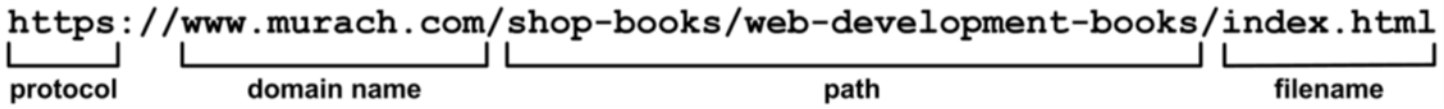
Objectives (part 2)

10. Describe how an ASP.NET Core app allows a developer to configure the middleware components in the HTTP request and response pipeline.
11. Define state and describe why it's hard to track in a web app.
12. Distinguish between the Visual Studio IDE and the code editor known as Visual Studio Code.
13. Describe how coding by convention works and how it can benefit developers.

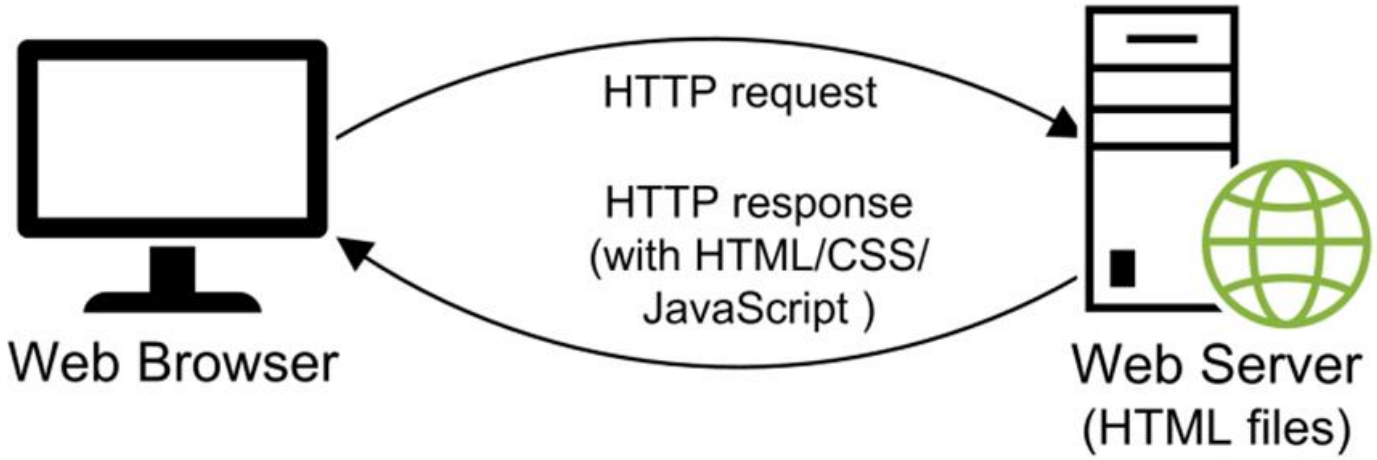
The components of a web app



The components of an HTTP URL



How a web server processes a static web page



A simple HTTP request

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

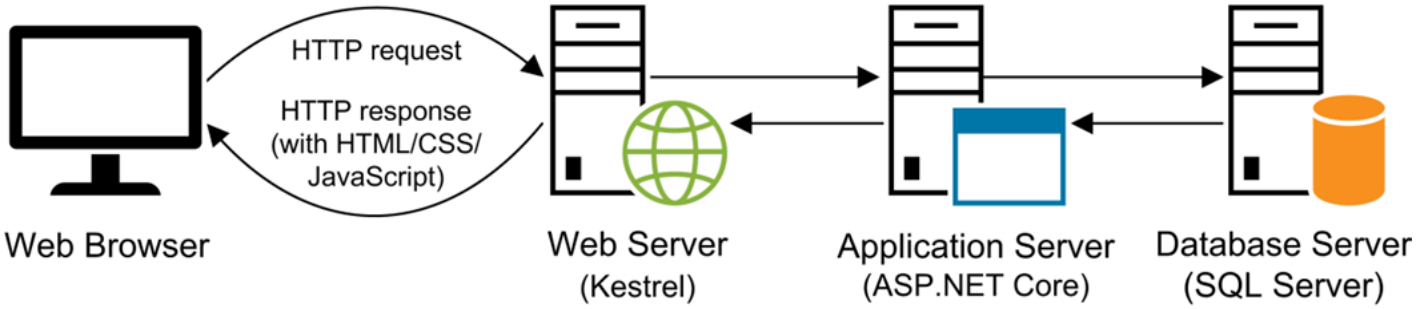
A simple HTTP response

```
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>
```

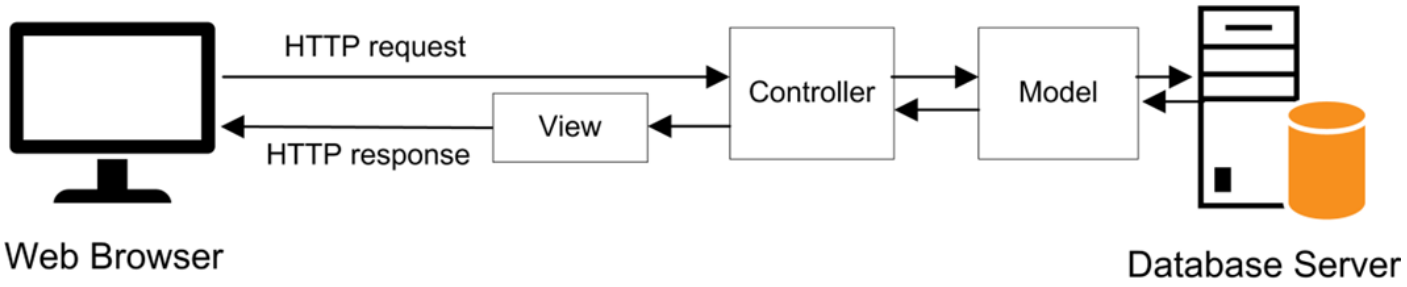
Three protocols that web apps depend upon

- *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.

How a web server processes a dynamic web page



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.

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.

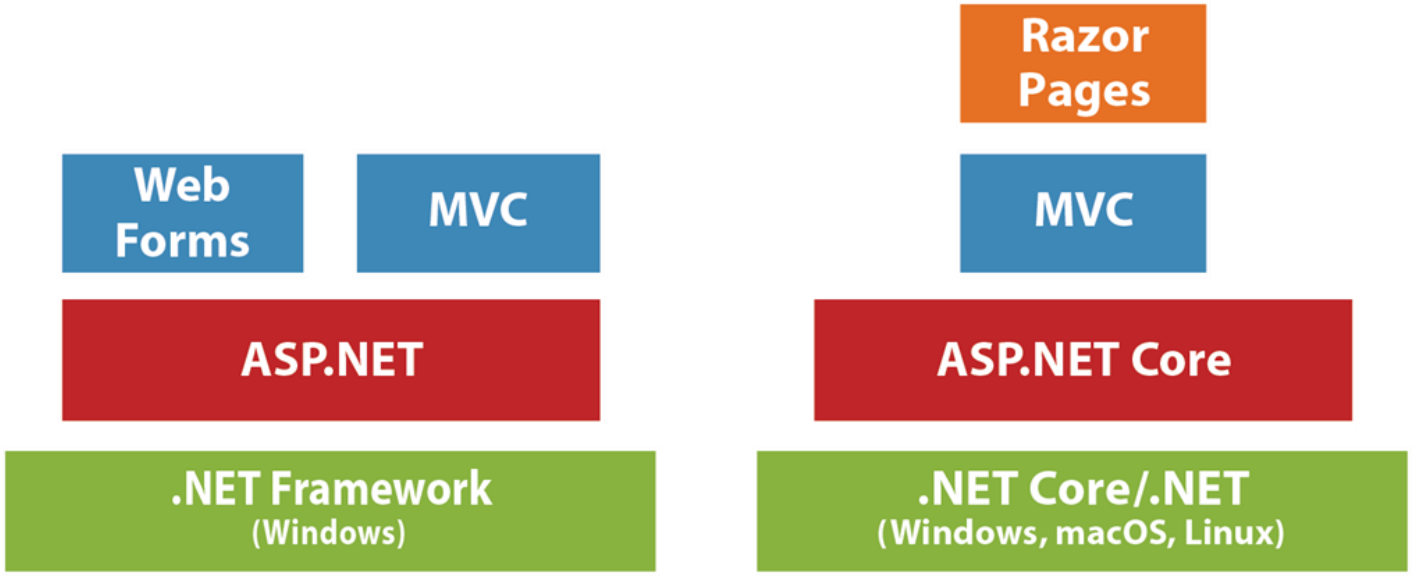
ASP.NET Core MVC

- 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.

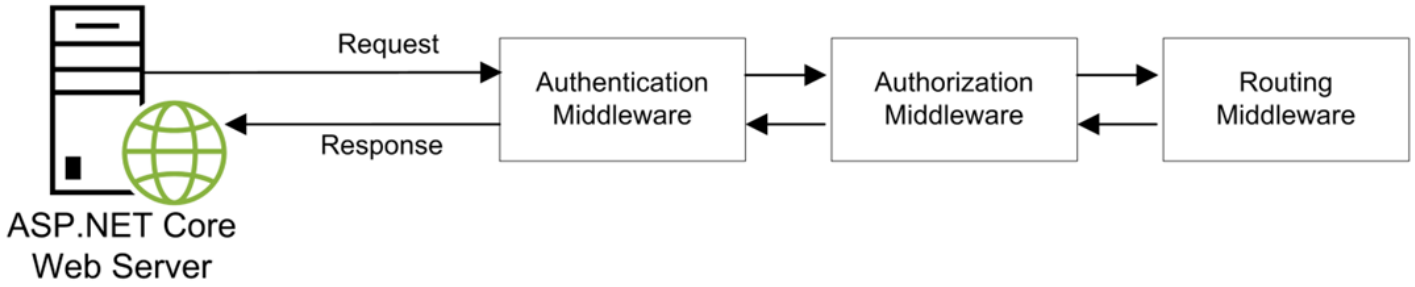
ASP.NET Core Razor Pages

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

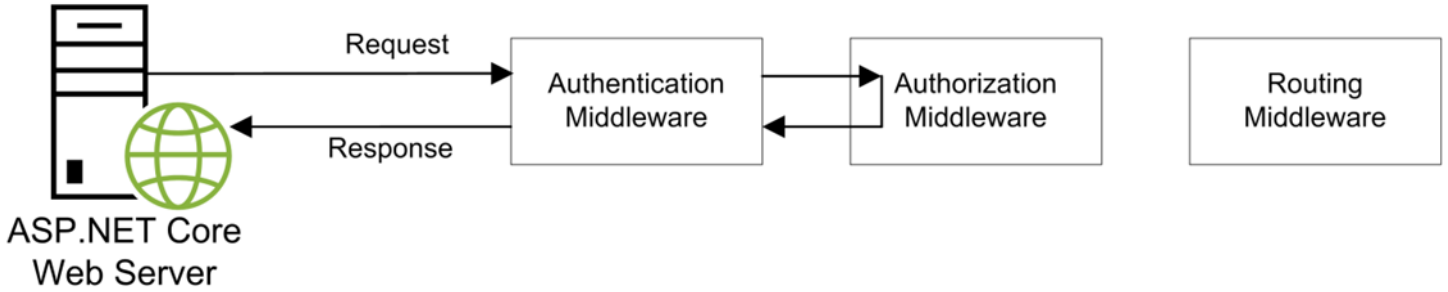
Some components of .NET Framework and .NET Core (.NET)



A request that makes it through all middleware in the pipeline



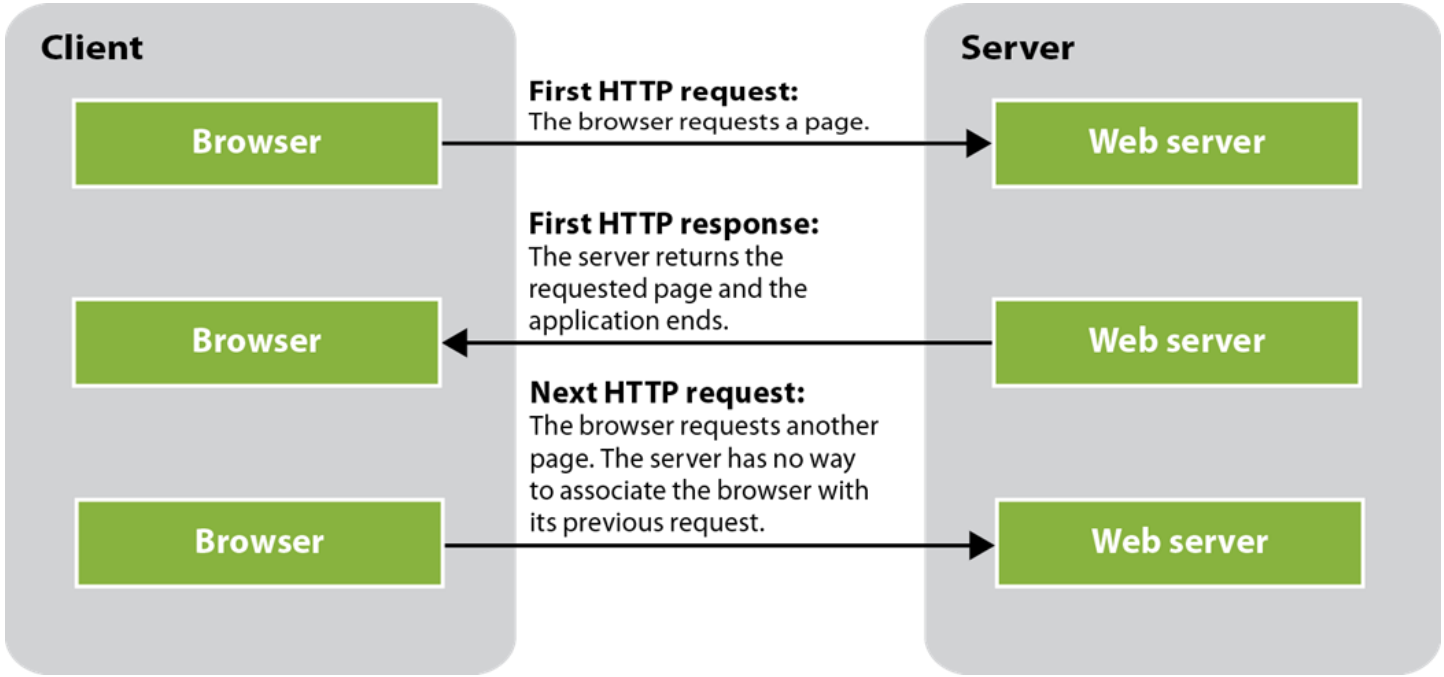
A request that's short circuited by a middleware component in the pipeline



Middleware can…

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

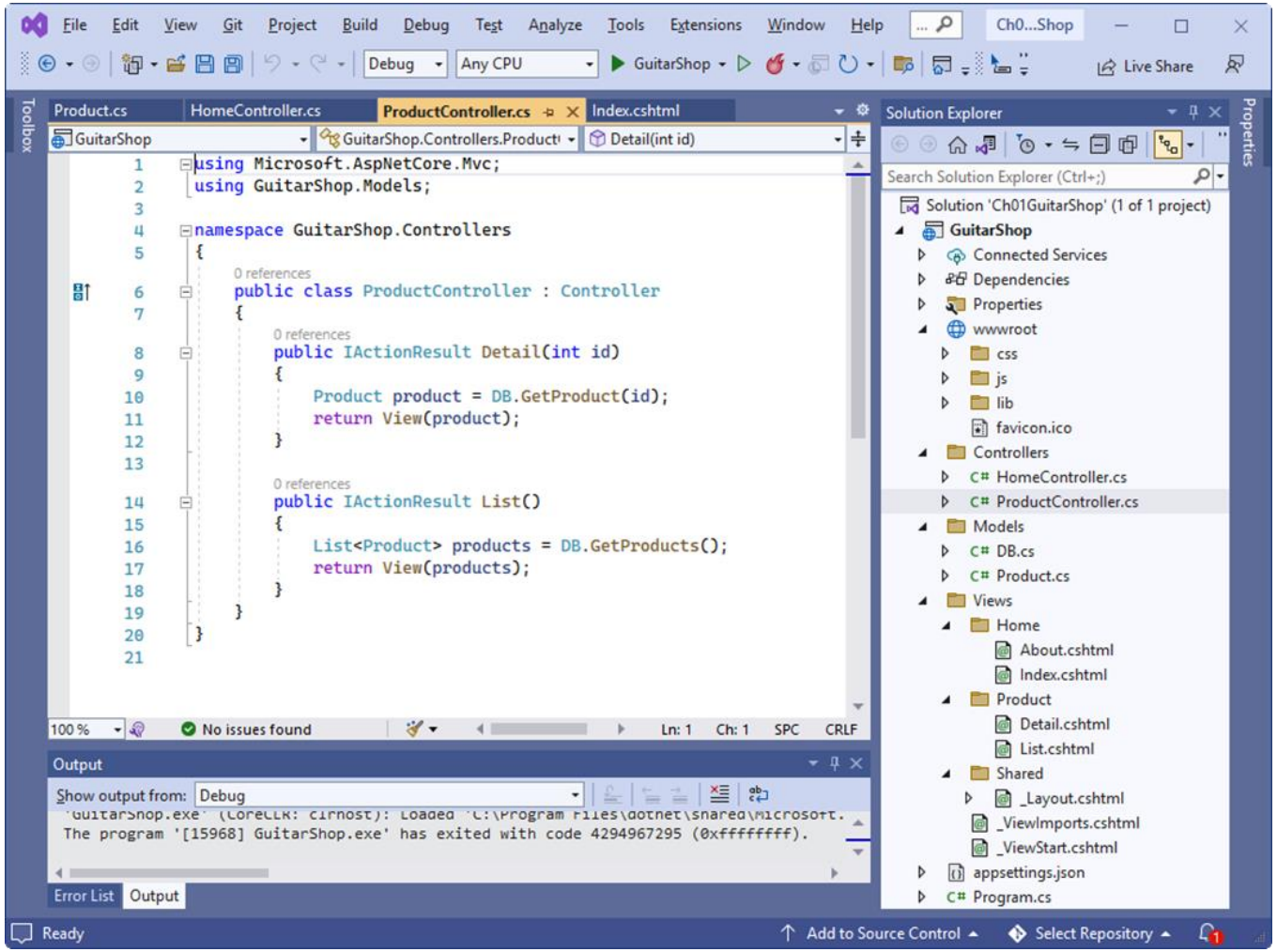
Why state is difficult to track in a web app



State 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.

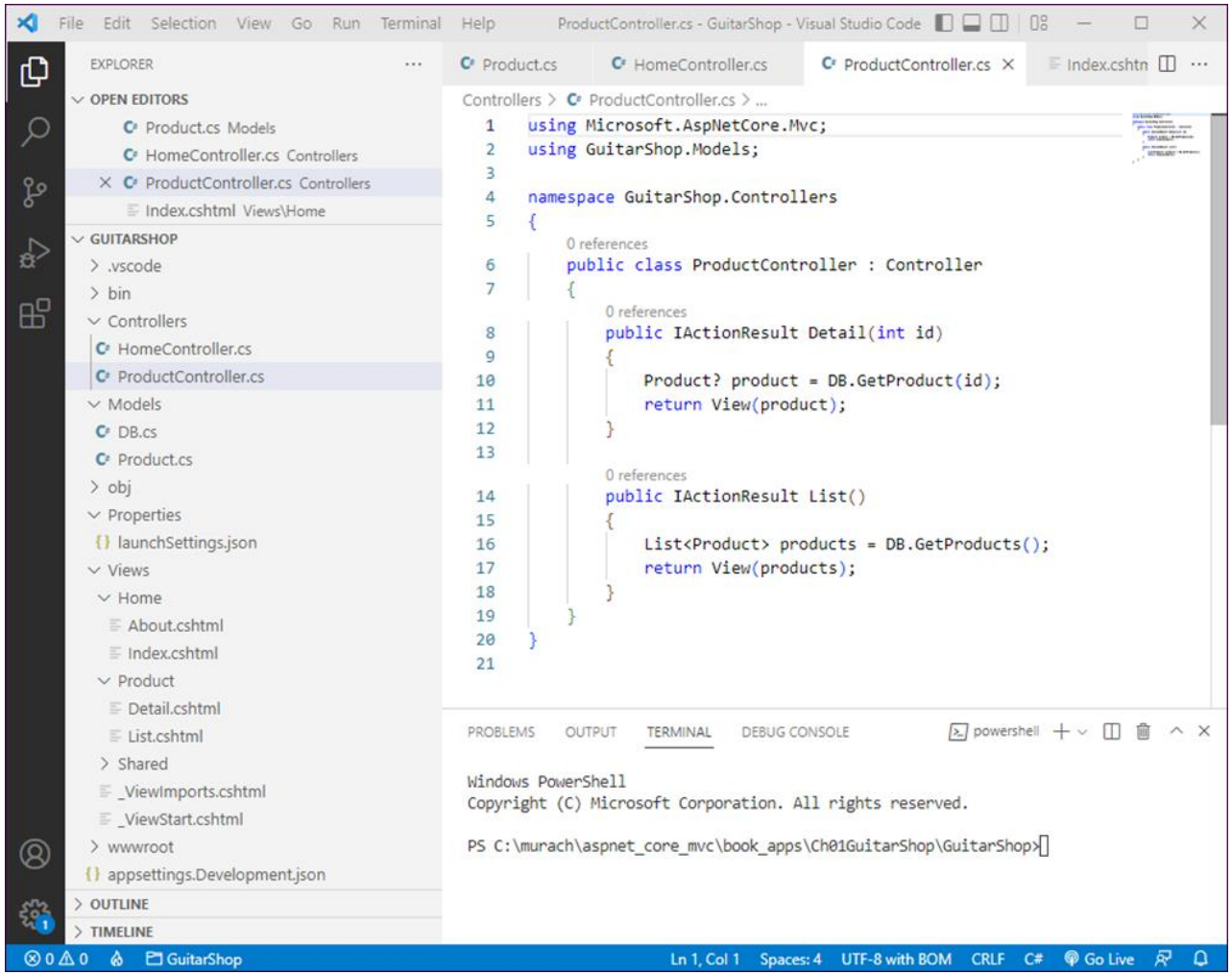
Visual Studio with an ASP.NET Core MVC app



Features of Visual Studio

- 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.

VS Code with an ASP.NET Core MVC app



Features of Visual Studio Code

- 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).

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".

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; }
    }
}
```

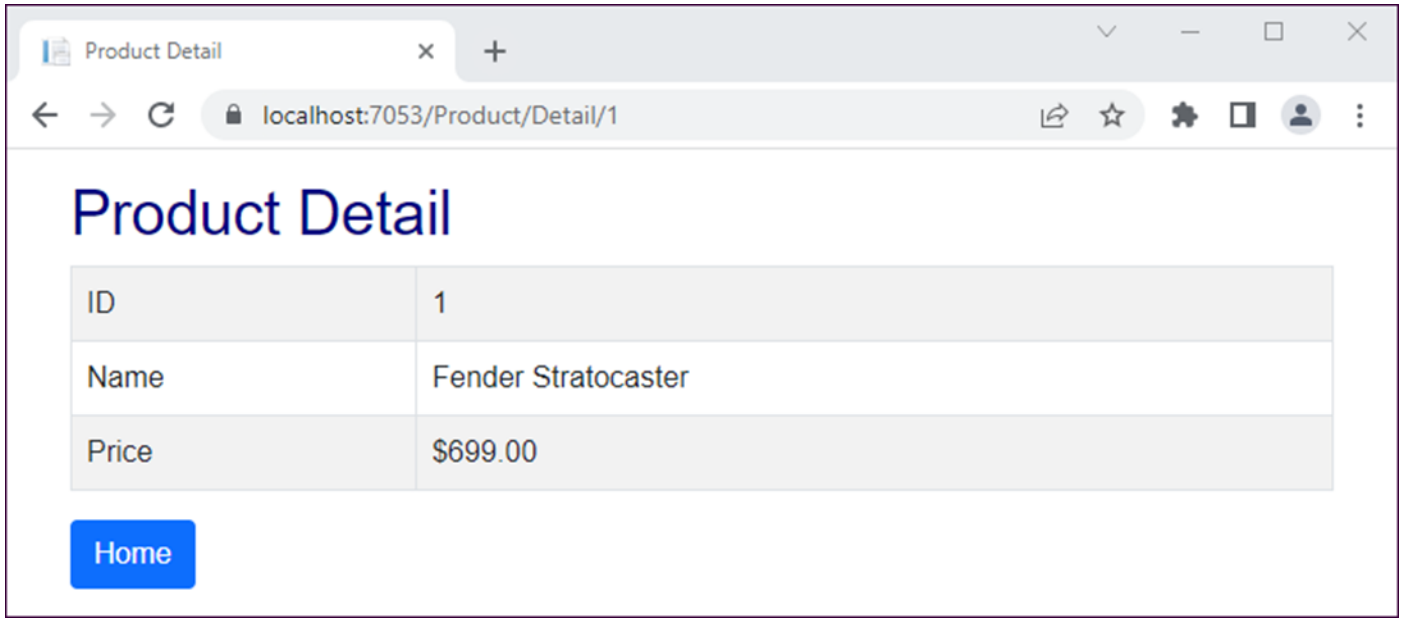
The code for the ProductController class

```
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);
        }

        public IActionResult List()
        {
            List<Product> products = DB.GetProducts();
            return View(products);
        }
    }
}
```

The code for the Product/Detail.cshtml view

```
@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



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`