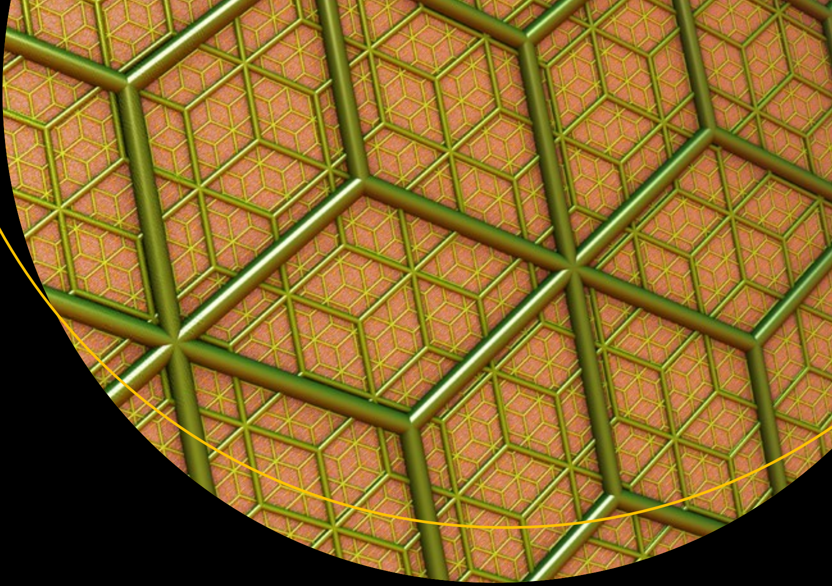


Creating ASP.NET Core Web Applications

Proven Approaches to Application Design and Development

Dirk Strauss

Apress®

Creating ASP.NET Core Web Applications

Proven Approaches to Application Design and Development

Dirk Strauss

Apress®

Creating ASP.NET Core Web Applications: Proven Approaches to Application Design and Development

Dirk Strauss
Uitenhage, South Africa

ISBN-13 (pbk): 978-1-4842-6827-8 ISBN-13 (electronic): 978-1-4842-6828-5
https://doi.org/10.1007/978-1-4842-6828-5

Managing Director, Apress Media LLC: Welmoed Spahr
Acquisitions Editor: Smriti Srivastava
Development Editor: Laura Berendson
Coordinating Editor: Shrikant Vishwakarma

Cover designed by eStudioCalamar

Cover image designed by Pexels

Distributed to the book trade worldwide by Springer Science+Business Media LLC, 1 New York Plaza, Suite 4600, New York, NY 10004. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springeronline.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a **Delaware** corporation.

For information on translations, please e-mail booktranslations@springernature.com; for reprint, paperback, or audio rights, please e-mail bookpermissions@springernature.com, or visit http://www.apress.com/rights-permissions.

Apress titles may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Print and eBook Bulk Sales web page at http://www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this book is available to readers on GitHub via the book's product page, located at www.apress.com/978-1-4842-6827-8. For more detailed information, please visit http://www.apress.com/source-code.

Printed on acid-free paper

For Adele, Irénéé, and Tristan. You are my everything.

Table of Contents

About the Author

Dirk Strauss is a software developer from South Africa who has been writing code since 2003. He has extensive experience in SYSPRO, with C# and web development being his main focus. He studied at the Nelson Mandela University, where he wrote software on a part-time basis to gain a better understanding of the technology. He remains passionate about writing code and imparting what he learns to others.

About the Technical Reviewer

Carsten Thomsen is a back-end developer primarily but working with smaller front-end bits as well. He has authored and reviewed a number of books and created numerous Microsoft Learning courses, all to do with software development. He works as a freelancer/contractor in various countries in Europe; Azure, Visual Studio, Azure DevOps, and GitHub are some of the tools he works with. Being an exceptional troubleshooter, asking the right questions, including the less logical ones, in a most logical to least logical fashion, he also enjoys working with architecture, research, analysis, development, testing, and bug fixing. Carsten is a very good communicator with great mentoring and team-lead skills, and great skills in researching and presenting new material.

Acknowledgments

I would like to thank my wife and kids for their support during the writing of this book. I love you always!

Introduction

.NET Core has given .NET developers a lot to think about. Some developers have embraced the technology, while others have taken a wait-and-see approach. Whatever approach you are taking, .NET Core is without a doubt here to stay.

Developing web applications is also not one of the easiest things to do. I've always wanted to write a book on developing web applications, but to do it in a way that is very structured and takes the reader on a journey of discovery.

Creating ASP.NET Core Web Applications is my attempt at that book. I always try to take the point of view that the book I'm writing is a reference book for my bookshelf. With this in mind, I, therefore, tried to cover a wide set of topics.

As with all projects, Chapter 1 starts with creating your project and using the .NET CLI. We have a look at adding Razor pages and also how to configure the application using the appsettings.json file. I then create a dummy data service, which is used to get the application up and running with test data. This test data is designed in such a way that it can easily be swapped out at a later stage (and I show you how to do this).

Chapter 2 takes a look at the process of creating models, model binding, tag helpers, working with a query string, and page routes. To illustrate these concepts, Chapter 2 shows you how to implement a search form. This allows us to search for data, view the details, and add in logic to handle bad requests.

Chapter 3 illustrates the concepts of editing the data, displaying validation errors, and modifying the data access service to suit our needs. I also discuss the differences between singleton, scoped, and transient lifetime registration for services.

EF Core and SQL Server become the focus in the next chapter. Chapter 4 shows you how to install Entity Framework Core, define your connection strings, what database migrations are, and how to use them. We will also be implementing a new data access service and changing the data access service registration from the test data to the SQL data.

Moving to the front end next, we have a look at working with Razor pages in Chapter 5. Here, we will look at what sections are and how they benefit you as a developer. We take a closer look at _ViewImports and _ViewStart files. I also show you how to create your own tag helper, how to work with partial views, and, finally, how to work with ViewComponents.

Staying front end, we have a look at adding client-side logic in Chapter 6. I show you how to separate production scripts from development scripts, use SCSS to generate CSS, how SCSS works, and the different features you can use to create CSS with SCSS, as well as work with Chrome Developer Tools. This is, in my opinion, crucial for any web developer to know.

With Chapter 7, we will take a look at what middleware is. This is a very important chapter and one that will require some explaining. We have a look at some of the built-in middleware components, but also how to create a custom middleware component if the built-in middleware components don't suit your needs. After creating a custom middleware component, we will have a look at logging in ASP.NET Core. Logging is a big subject, but this book tries to cover the basics.

Finally, Chapter 8 will take you through getting your web application ready for deployment and finally publishing your web application and hooking it up to a SQL Server database. I hope that you will enjoy this book as much as I enjoyed writing it.

CHAPTER 1

Creating and Setting Up Your Project

Welcome to Creating ASP.NET Core Web Applications! This book will guide you through creating a typical ASP.NET Core Web Application, from start to finish. All the code illustrated in this book is available on GitHub and will be an invaluable resource to you as you navigate the code samples in the book.

This chapter will take you through the steps required to start your web application development. We will also have a look at adding and editing Razor pages, working with Entities, creating and registering a data service, and using that data service to display test data on the web page.

Creating Your Web Application Project

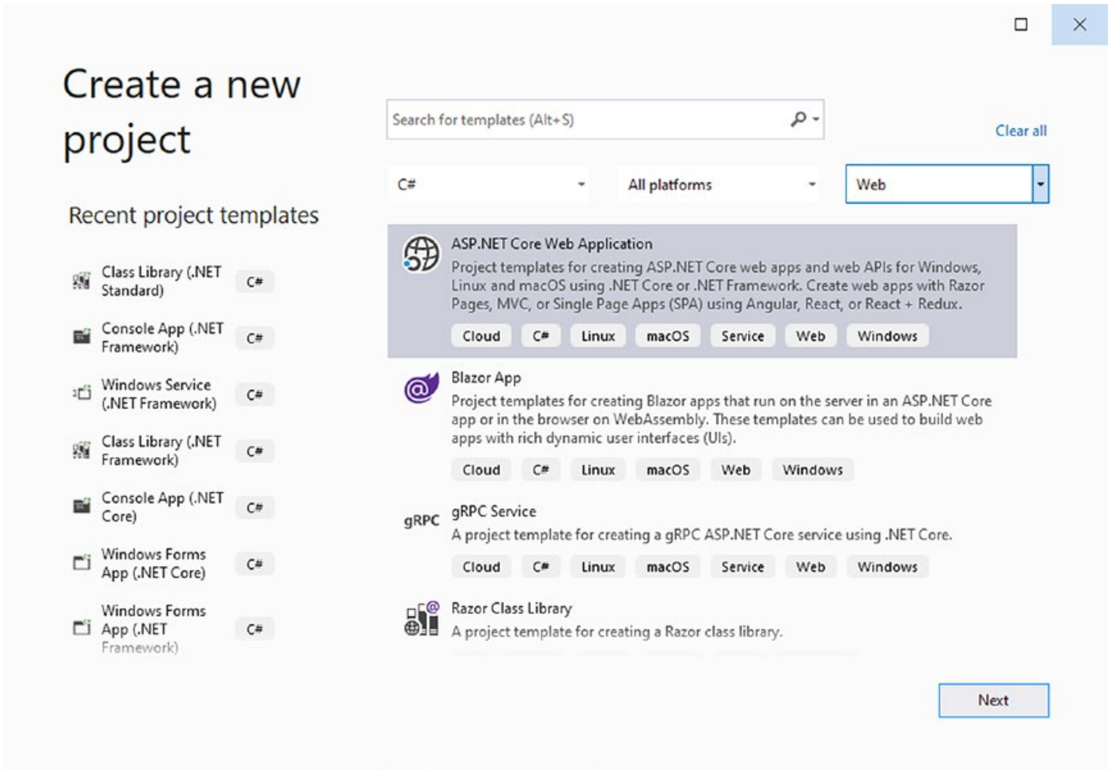
In this book, I will be using Visual Studio 2019 to illustrate the concepts surrounding ASP.NET Core Web Applications. For those folks that do not use Visual Studio, the same result as detailed in the following can be achieved for creating an application by using the .NET CLI.

I will assume that you have already installed .NET Core onto your machine. The web application we will be creating will use .NET Core 3.1. If you have not installed .NET Core, you can do so by visiting this link: `https://dotnet.microsoft.com/download`.

Because we are working with .NET Core which is cross-platform, I will also show you how to create an application using the Command Prompt later in this section.

© Dirk Strauss 2021
D. Strauss, *Creating ASP.NET Core Web Applications*, https://doi.org/10.1007/978-1-4842-6828-5_1

For now, let us start by creating a new project in Visual Studio. From the file menu, click New Project. This will display the Create a new project screen as seen in Figure 1-1.


Create a new project

Search for templates (Alt+S)

Clear all

C# | All platforms | Web

Recent project templates

- Class Library (.NET Standard) C#
- Console App (.NET Framework) C#
- Windows Service (.NET Framework) C#
- Class Library (.NET Framework) C#
- Console App (.NET Core) C#
- Windows Forms App (.NET Core) C#
- Windows Forms App (.NET Framework) C#

ASP.NET Core Web Application
Project templates for creating ASP.NET Core web apps and web APIs for Windows, Linux and macOS using .NET Core or .NET Framework. Create web apps with Razor Pages, MVC, or Single Page Apps (SPA) using Angular, React, or React + Redux.
Cloud C# Linux macOS Service Web Windows

Blazor App
Project templates for creating Blazor apps that run on the server in an ASP.NET Core app or in the browser on WebAssembly. These templates can be used to build web apps with rich dynamic user interfaces (UIs).
Cloud C# Linux macOS Web Windows

gRPC Service
A project template for creating a gRPC ASP.NET Core service using .NET Core.
Cloud C# Linux macOS Service Web Windows

Razor Class Library
A project template for creating a Razor class library.

Next

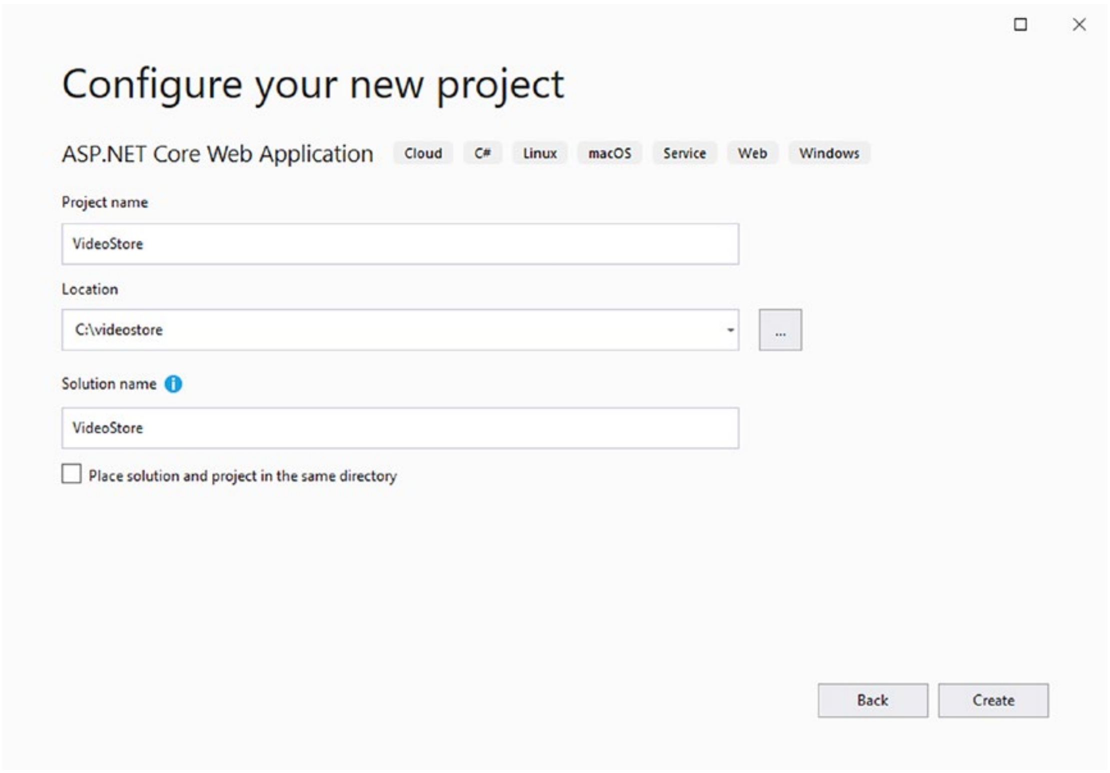
Figure 1-1. The Create New Project Screen

The Create a new project screen that allows you to select the correct project template lists all the available templates included in Visual Studio. In our case, we will be using the ASP.NET Core Web Application template.

If you are used to working in previous versions of Visual Studio, you will notice that this screen has been vastly improved. You can search for templates by typing a template name into the search text box or by holding down Alt+S.

You can also filter project templates from the drop-downs on the form. You will notice that you can filter by language, platform, and project type.

Clicking the Next button will take you to the Configure your new project screen as seen in Figure 1-2.



Configure your new project

ASP.NET Core Web Application

Project name: VideoStore

Location: C:\videostore

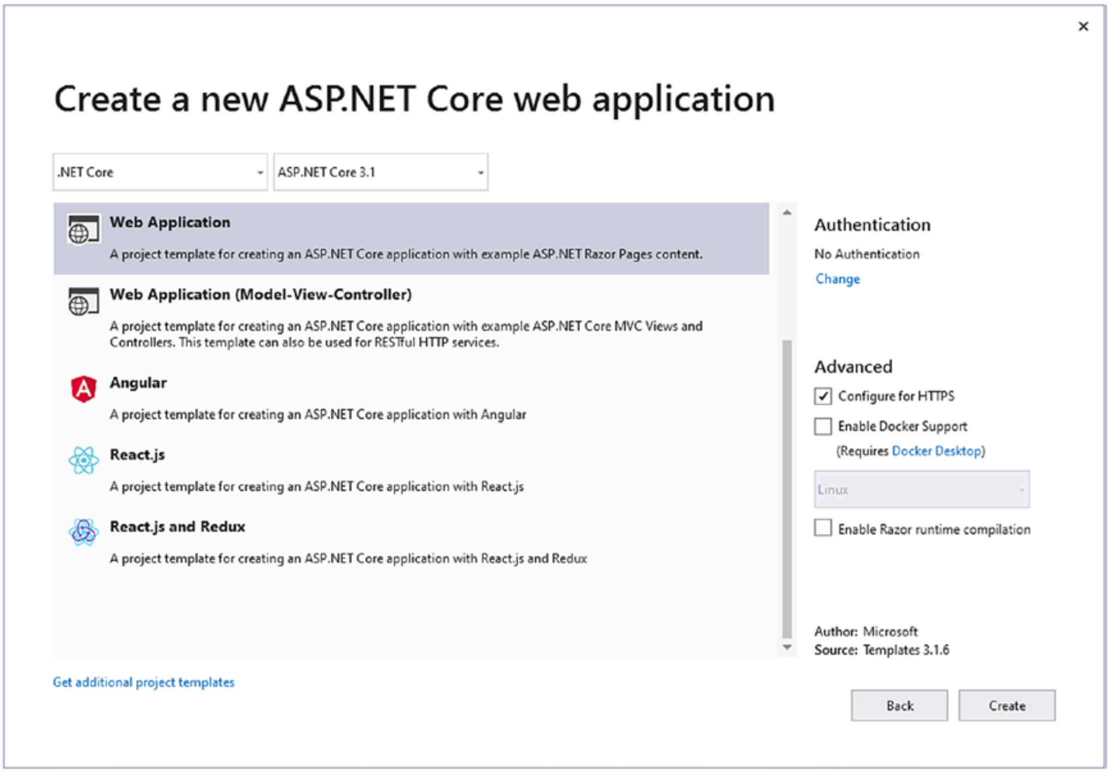
Solution name: VideoStore

☐ Place solution and project in the same directory

Figure 1-2. *Configure Your New Project*

Give the project a suitable name. For this book, we will simply call the project VideoStore and specify a location to create the project in. When you have done this, click the Create button.

You will now be taken to a second screen as seen in Figure 1-3 where you can select the specific type of template that you want to use.



Create a new ASP.NET Core web application

.NET Core | ASP.NET Core 3.1

Web Application
A project template for creating an ASP.NET Core application with example ASP.NET Razor Pages content.

Web Application (Model-View-Controller)
A project template for creating an ASP.NET Core application with example ASP.NET Core MVC Views and Controllers. This template can also be used for RESTful HTTP services.

Angular
A project template for creating an ASP.NET Core application with Angular

React.js
A project template for creating an ASP.NET Core application with React.js

React.js and Redux
A project template for creating an ASP.NET Core application with React.js and Redux

Get additional project templates

Authentication
No Authentication
Change

Advanced
☑ Configure for HTTPS
☐ Enable Docker Support
(Requires Docker Desktop)
Linux
☐ Enable Razor runtime compilation

Author: Microsoft
Source: Templates 3.1.6

Figure 1-3. *Selecting a Specific Template Type*

It is here that we can specify the version of .NET Core that we want to use. In this example, we are selecting .NET Core 3.1. We can then tell Visual Studio that we want to create a basic web application. Just leave the rest of the settings at their default values and click the Create button.

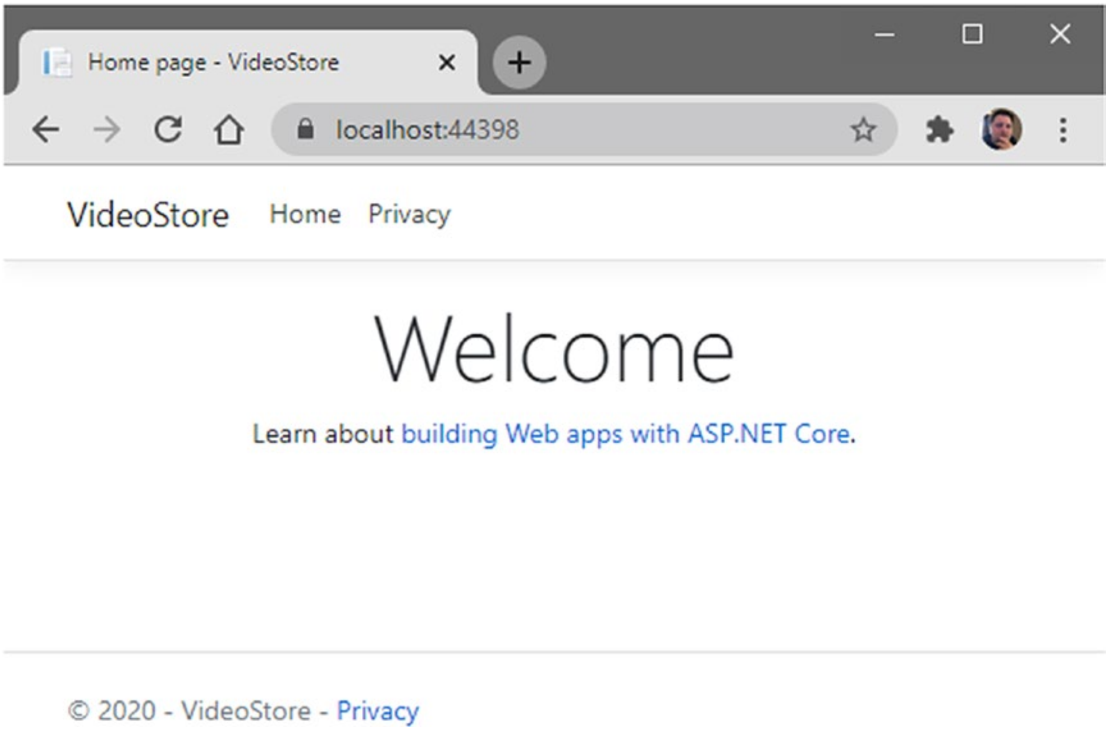

Home page - VideoStore
localhost:44398
VideoStore
Home
Privacy
Welcome
Learn about building Web apps with ASP.NET Core.
© 2020 - VideoStore - Privacy

Figure 1-4. *Running the Web Application*

After the project has been created in Visual Studio, you can hit Ctrl+F5 to run the web application. This will run your project without the debugger attached and display the web application in your browser as seen in Figure 1-4.

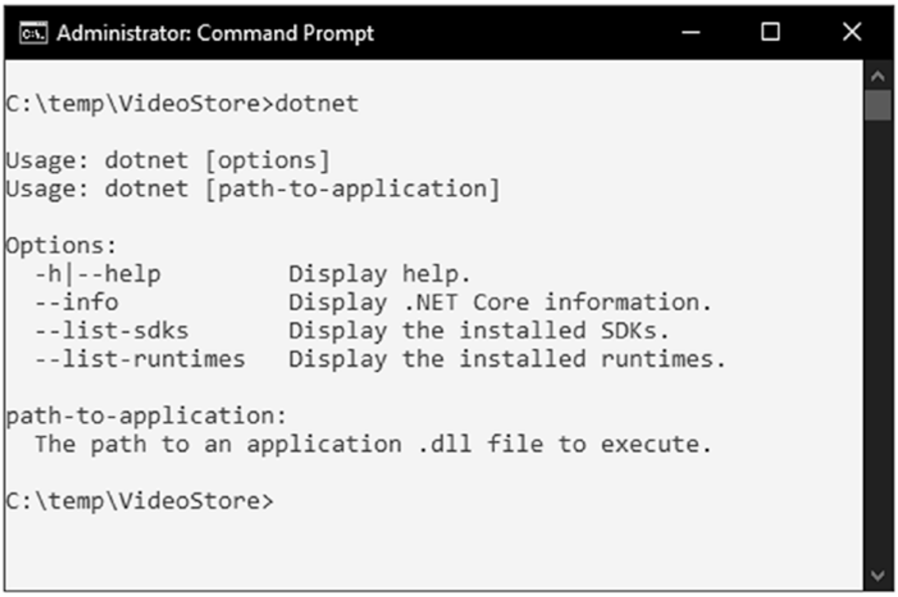
You will notice that the web application is running on port 44398 in this example, but your port will most likely be different. By default, this web application includes some basic features such as a Home page as well as a Privacy page.

It is from here that we will start to flesh out our web application and add more features and functionality to it.

Using the .NET CLI

Earlier in this chapter, I mentioned that we can also create the project from the Command Prompt. Therefore, for those of you that do not use Visual Studio, the .NET CLI offers a cross-platform way for creating .NET Core projects.

Once you have installed .NET Core on your Mac, Linux, or Windows machine, you should be able to simply open your Terminal, Shell, or Command Prompt and type the `dotnet` command as seen in Figure 1-5.



```
C:\temp\VideoStore>dotnet

Usage: dotnet [options]
Usage: dotnet [path-to-application]

Options:
  -h|--help         Display help.
  --info            Display .NET Core information.
  --list-sdks       Display the installed SDKs.
  --list-runtimes   Display the installed runtimes.

path-to-application:
  The path to an application .dll file to execute.

C:\temp\VideoStore>
```

Figure 1-5. *Running the dotnet Command*

To see more of the commands available with `dotnet`, you can type `dotnet -h` in the Command Prompt. If you typed in `dotnet new`, you would see all the available project templates listed in your Command Prompt window.

These templates, along with the short name associated with that specific template, are listed in the following table.

Templates	**Short Name**	**Language**	**Tags**
Console Application	Console	C#, F#, VB	Common/ Console
Class library	classlib	C#, F#, VB	Common/ Library
WPF Application	wpf	C#	Common/WPF
WPF Class library	wpflib	C#	Common/WPF

(*continued*)

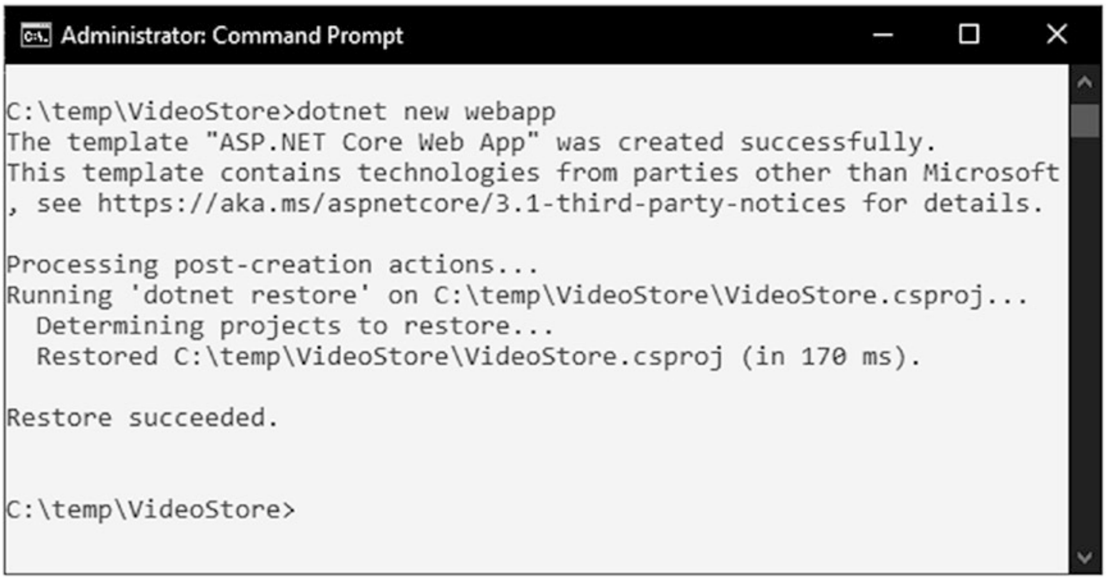
Templates	Short Name	Language	Tags
WPF Custom Control Library	wpfcustomcontrollib	C#	Common/WPF
WPF User Control Library	wpfusercontrollib	C#	Common/WPF
Windows Forms (WinForms) Application	winforms	C#	Common/ WinForms
Windows Forms (WinForms) Class library	winformslib	C#	Common/ WinForms
Worker Service	worker	C#	Common/ Worker/Web
Unit Test Project	mstest	C#, F#, VB	Test/MSTest
NUnit 3 Test Project	nunit	C#, F#, VB	Test/NUnit
NUnit 3 Test Item	nunit-test	C#, F#, VB	Test/NUnit
xUnit Test Project	xunit	C#, F#, VB	Test/xUnit
Razor Component	razorcomponent	C#	Web/ASP.NET
Razor Page	page	C#	Web/ASP.NET
MVC ViewImports	viewimports	C#	Web/ASP.NET
MVC ViewStart	viewstart	C#	Web/ASP.NET
Blazor Server App	blazorserver	C#	Web/Blazor
Blazor WebAssembly App	blazorwasm	C#	Web/Blazor/ WebAssembly
ASP.NET Core Empty	web	C#, F#	Web/Empty
ASP.NET Core Web App (Model-View-Controller)	mvc	C#, F#	Web/MVC
ASP.NET Core Web App	webapp	C#	Web/MVC/ Razor Pages
ASP.NET Core with Angular	angular	C#	Web/MVC/SPA
ASP.NET Core with React.js	React	C#	Web/MVC/SPA
ASP.NET Core with React.js and Redux	Reactredux	C#	Web/MVC/SPA

(*continued*)

Templates	Short Name	Language	Tags
Razor Class Library	Razorclasslib	C#	Web/Razor/ Library/ Razor Class Library
ASP.NET Core Web API	Webapi	C#, F#	Web/WebAPI
ASP.NET Core gRPC Service	Grpc	C#	Web/gRPC
dotnet gitignore file	gitignore		Config
global.json file	globaljson		Config
NuGet Config	nugetconfig		Config
Dotnet local tool manifest file	tool-manifest		Config
Web Config	webconfig		Config
Solution File	sln		Solution

You will notice that to create an ASP.NET Web Application, we need to specify the short name `webapp` with the `new` command.

As seen in Figure 1-6, typing in the command `dotnet new webapp` will create the ASP.NET Web Application inside the current directory.


Administrator: Command Prompt

```
C:\temp\VideoStore>dotnet new webapp
The template "ASP.NET Core Web App" was created successfully.
This template contains technologies from parties other than Microsoft
, see https://aka.ms/aspnetcore/3.1-third-party-notices for details.

Processing post-creation actions...
Running 'dotnet restore' on C:\temp\VideoStore\VideoStore.csproj...
  Determining projects to restore...
  Restored C:\temp\VideoStore\VideoStore.csproj (in 170 ms).

Restore succeeded.


C:\temp\VideoStore>
```

Figure 1-6. *Creating the Web App via the .NET CLI*

If you had to compare the project created via the .NET CLI with the one created in Visual Studio, you will see that these are identical.

The .NET CLI offers a fantastic, quick, and cross-platform way of creating applications.

Adding and Editing Razor Pages

With your web application running, you will notice that if you click the Privacy link in the navigation menu, it will go to the following URL: `https://localhost:44398/Privacy`. The web application is mapping the request created by clicking the Privacy link with the Razor pages in your VideoStore project. Looking at Figure 1-7, you will see the Razor pages in a folder called… you guessed it, Pages.

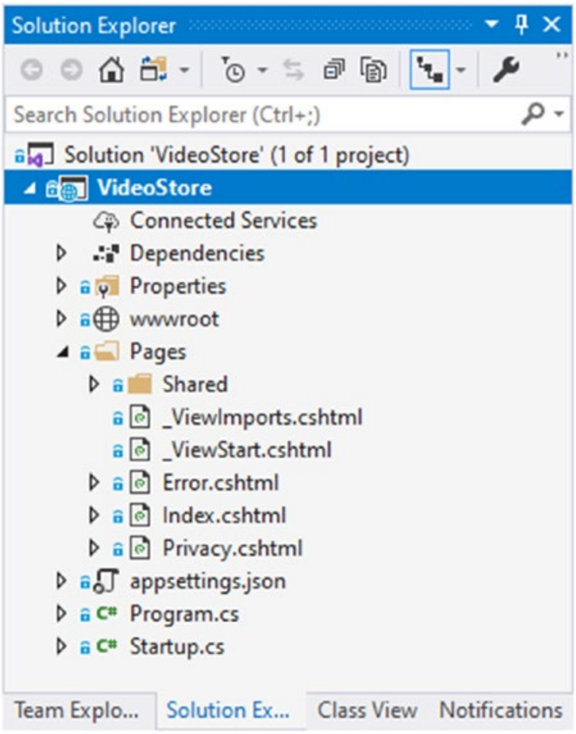
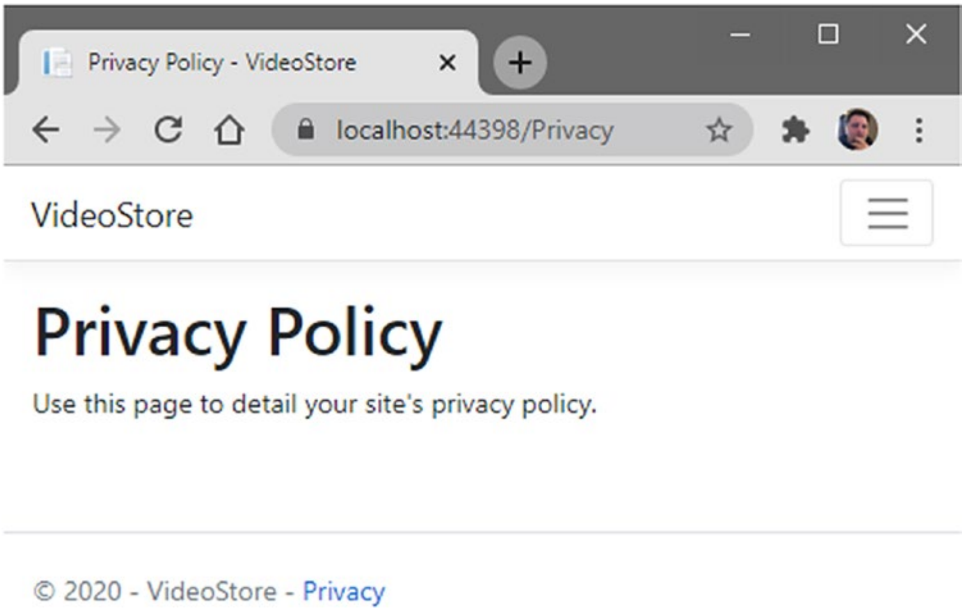


Figure 1-7. *The Razor Pages in the Solution Explorer*

This means that when I view the Privacy page in the web application, ASP.NET Core is busy rendering the Privacy.cshtml page. You will also notice that the cshtml extension is not required in the URL as seen in Figure 1-8.



Privacy Policy

Use this page to detail your site's privacy policy.

© 2020 - VideoStore - Privacy

Figure 1-8. *The Privacy Policy Page*

You will also notice that when you make your browser window smaller, the menu collapses into the hamburger icon. This is made possible by Bootstrap, which is included in the project by default.

If you now click your Privacy.cshtml page in the Solution Explorer, you will see the code as listed in Code Listing 1-1.

Listing 1-1. The Privacy Razor Page

```
@page
@model PrivacyModel
@{
    ViewData["Title"] = "Privacy Policy";
}
<h1>@ViewData["Title"]</h1>

<p>Use this page to detail your site's privacy policy.</p>
```

With your web application running without the debugger attached, if you click the hamburger menu icon, you will see that we have just two pages listed which are Home and Privacy.

Looking at Figure 1-9 and comparing that to the code in Code Listing 1-1, you might be wondering where the code is for the navigation.

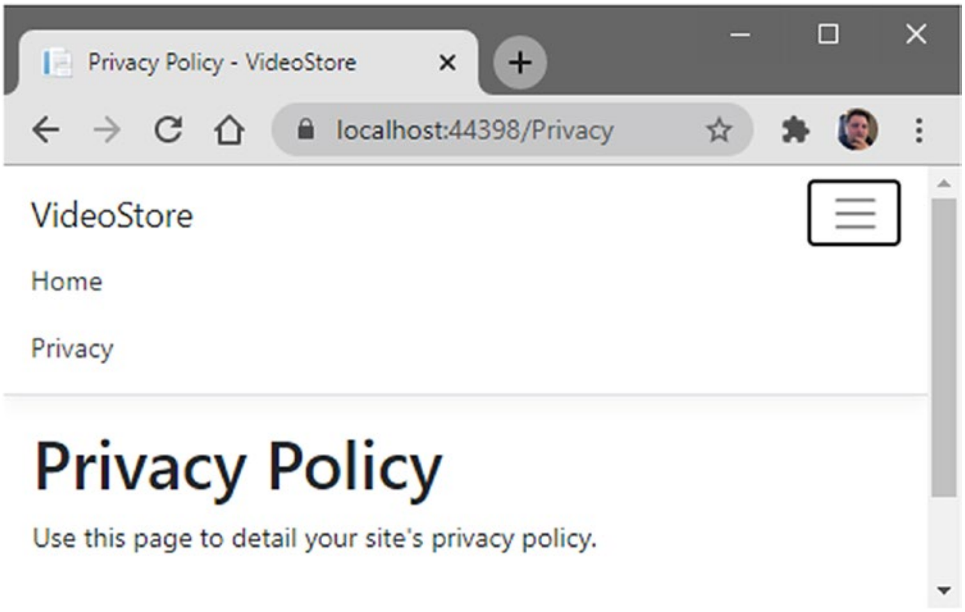

Privacy Policy - VideoStore
localhost:44398/Privacy
VideoStore
Home
Privacy
Privacy Policy
Use this page to detail your site's privacy policy.

Figure 1-9. *The Navigation Menu*

The answer lies in a special Razor page called a Layout page. Swing back to your Solution Explorer, and expand the Shared folder under the Pages folder. There you will see a page called `_Layout.cshtml` as seen in Figure 1-10.

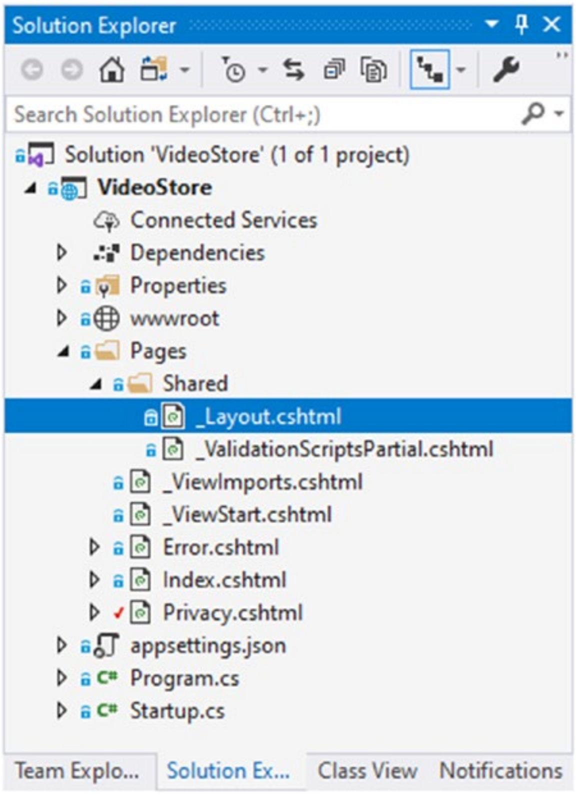


Figure 1-10. *The Shared Layout Page*

It is this Layout page that renders everything within the web application's <head> tags, things such as links to all the required stylesheets, as well as <body> tags that include a <header> section containing the navigation menu. The code for the navigation menu is listed in Code Listing 1-2.

Listing 1-2. The Navigation Menu Code

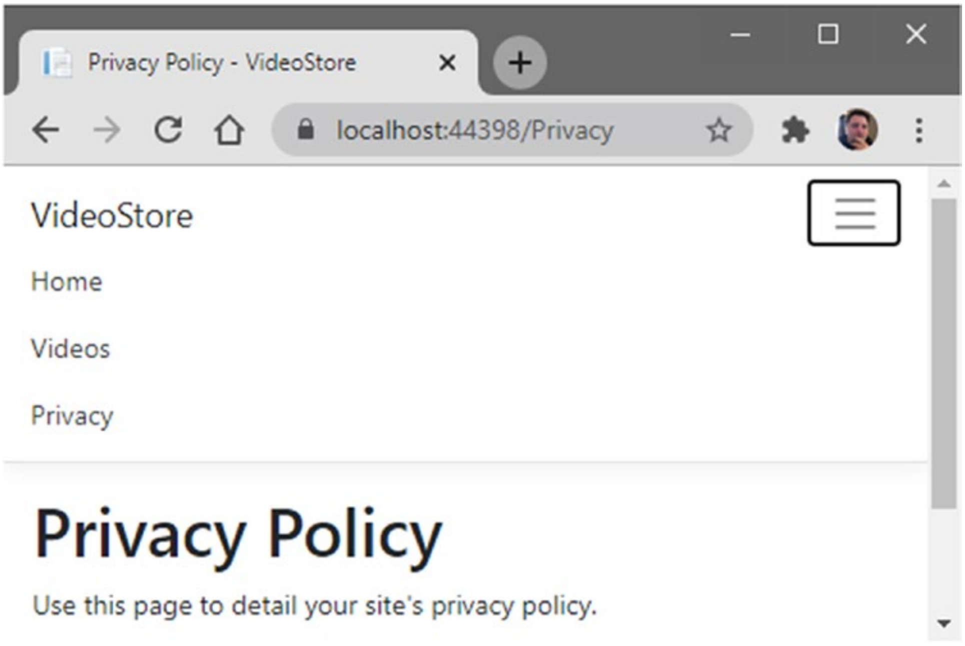
```
<div class="navbar-collapse collapse d-sm-inline-flex flex-sm-row-reverse">
  <ul class="navbar-nav flex-grow-1">
    <li class="nav-item"><a class="nav-link text-dark" asp-area="" asp-
    page="/Index">Home</a></li>
    <li class="nav-item"><a class="nav-link text-dark" asp-area="" asp-
    page="/Privacy">Privacy</a></li>
  </ul>
</div>
```

Go ahead and add another menu item called Videos, by adding a new list item to the unordered list as seen in Code Listing 1-3.

Listing 1-3. Modified Navigation Menu Code

```
<div class="navbar-collapse collapse d-sm-inline-flex flex-sm-row-reverse">
  <ul class="navbar-nav flex-grow-1">
    <li class="nav-item"><a class="nav-link text-dark" asp-area="" asp-
    page="/Index">Home</a></li>
    <li class="nav-item"><a class="nav-link text-dark" asp-area="" asp-
    page="/ Videos/List">Videos</a></li>
    <li class="nav-item"><a class="nav-link text-dark" asp-area="" asp-
    page="/Privacy">Privacy</a></li>
  </ul>
</div>
```

You will notice that the `asp-page` tag helper specifies `Videos/List` which tells my web application that inside a folder called Videos is a page that will display a list of videos. Running your web application again, you will see that the Videos menu item has been added to the navigation menu (Figure 1-11).



Privacy Policy

Use this page to detail your site's privacy policy.

Figure 1-11. *Navigation Menu Modified*

If you click the Videos menu item, the link will not navigate anywhere. This is because we have not yet added the required Razor page. As shown in Figure 1-12, add a new folder under the Pages folder in your Solution Explorer.

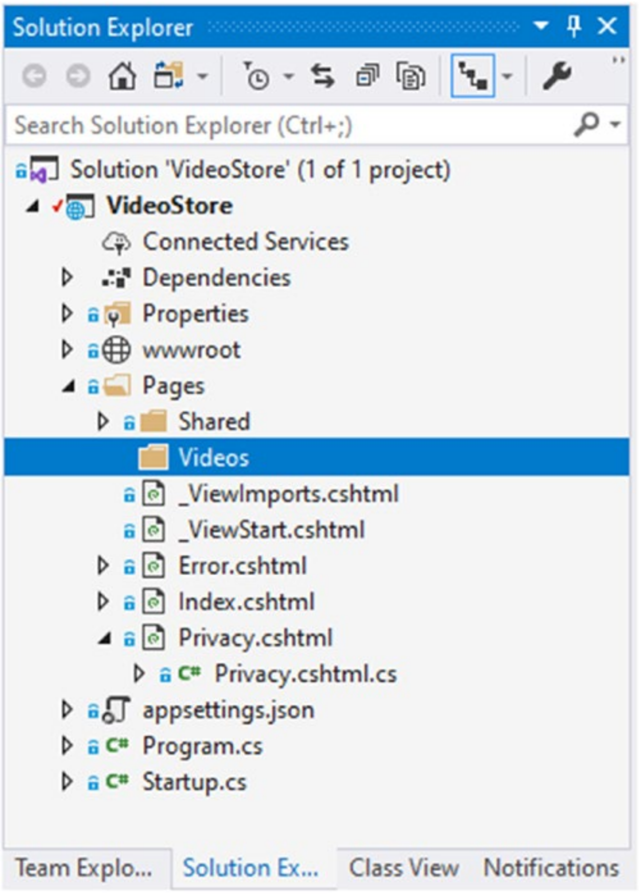


Figure 1-12. *Adding the Videos Folder*

Next, right-click the Videos folder, and add a new Razor page called `List` to the folder. This can be done from the context menu or from the `Add New Item` screen as shown in Figure 1-13.

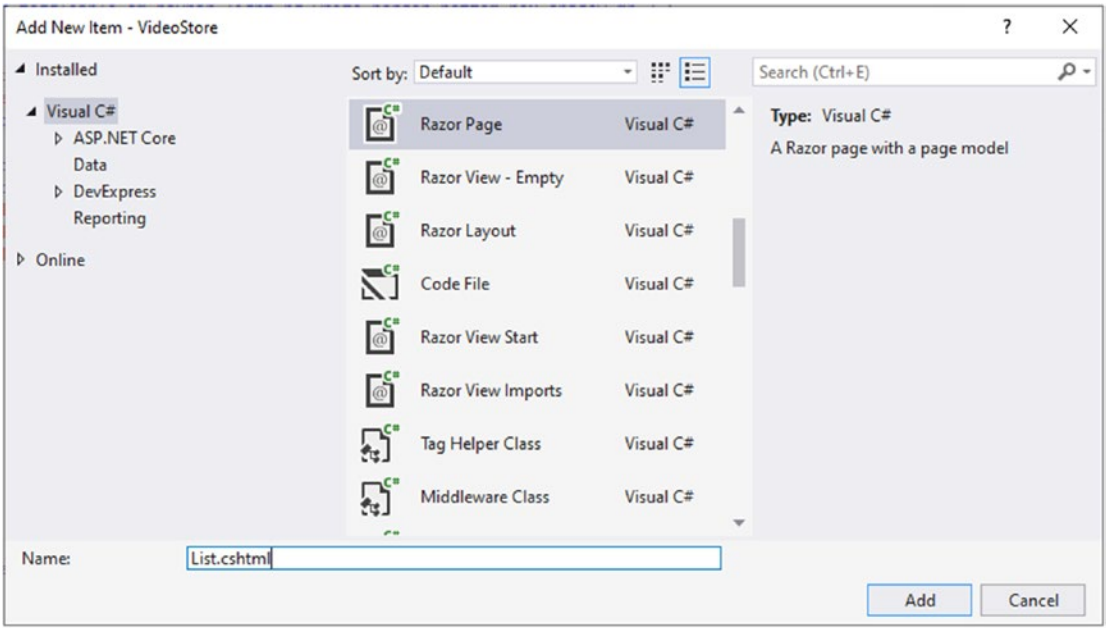


Figure 1-13. *Adding a New Razor Page*

Once the `List.cshtml` page has been added, you will notice that Visual Studio has added a second page (Figure 1-14) called `List.cshtml.cs`. The `List.cshtml` file is essentially my Razor page containing the `@page` directive (Code Listing 1-4).

Listing 1-4. Razor Page Code

```
@page
@model VideoStore.Pages.Videos.ListModel
@{
}
```

Furthermore, the Razor page also specifies a model with the `@model` directive. It is telling .NET Core that the model that contains video information is contained in an object of type `ListModel`.

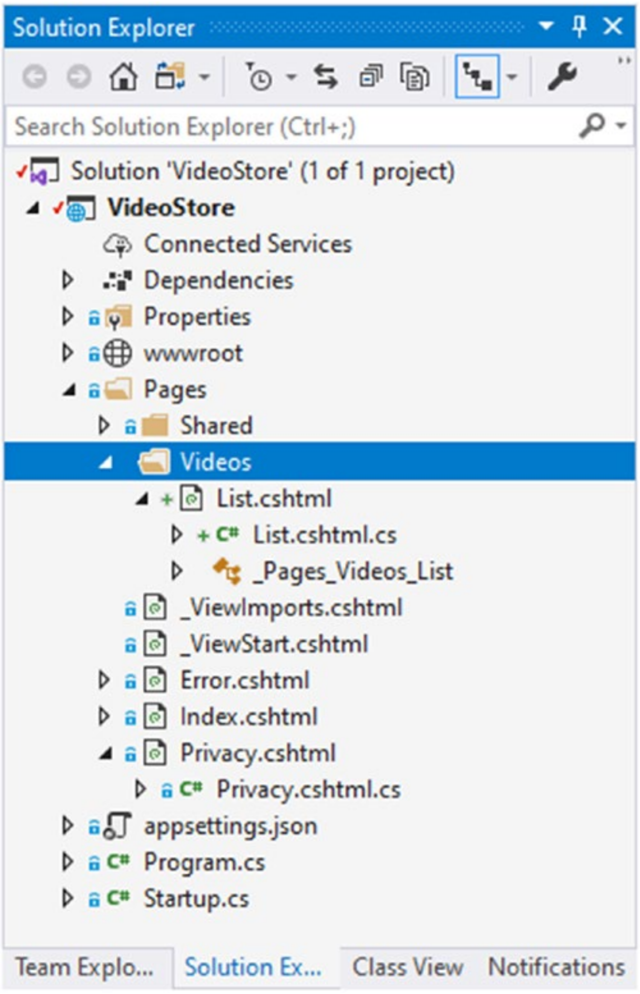


Figure 1-14. *List Page Added*

This `ListModel` is the class contained in the `List.cshtml.cs` file nested under the `List.cshtml` page. Looking at the code listing in Listing 1-5, it is interesting to notice that the `ListModel` class inherits from an abstract class called `PageModel`.