

Exercise - Customize the project

12 minutes

Your team has split up the work for the pizza inventory management app. Your teammates have created the ASP.NET Core web app for you, and they've already built a service to read and write pizza data to a database. You've been assigned to work on the Pizza List page, which will display a list of pizzas and allow you to add new pizzas to the database. We'll start by taking a tour of the project to understand how it's organized.

Note

This module uses the .NET CLI (Command Line Interface) and Visual Studio Code for local development. After completing this module, you can apply the concepts using Visual Studio (Windows) or continued development using Visual Studio Code (Windows, Linux, and macOS).

Obtain the project files

If you're using GitHub Codespaces, just [open the repository in the browser](#), select **Code**, and then create a new codespace on the `main` branch.

If you aren't using GitHub Codespaces, obtain the project files and open them in Visual Studio Code with the following steps:

1. Open a command shell and clone the project from GitHub using the command line:

```
Bash

git clone https://github.com/MicrosoftDocs/mslearn-create-razor-pages-aspnet-core
```

2. Navigate to the `mslearn-create-razor-pages-aspnet-core` directory and open the project in Visual Studio Code:

```
Bash

cd mslearn-create-razor-pages-aspnet-core
code .
```

Tip

If you've got a compatible container runtime installed, you can use the [Dev Containers](#) extension to open the repository in a container with the tools preinstalled.

Review your teammates' work

Let's take a moment to get familiar with the existing code in the ContosoPizza folder. The project is an ASP.NET Core web app created using the `dotnet new webapp` command. The changes your teammates made are described below.

Tip

Don't spend too much time reviewing these changes. Your teammates have already done the work to create the database and the service to read and write pizzas to the database, but they didn't make any UI changes. You'll be building a UI that consumes their service in the next unit.


- A *Models* folder was added to the project.
 - The model folder contains a `Pizza` class that represents a pizza.
- A *Data* folder was added to the project.
 - The data folder contains a `PizzaContext` class that represents the database context. It inherits from the `DbContext` class in Entity Framework Core. Entity Framework Core is an object-relational mapper (ORM) that makes it easier to work with databases.
- A *Services* folder was added to the project.
 - The services folder contains a `PizzaService` class that exposes methods to list and add pizzas.
 - The `PizzaService` class uses the `PizzaContext` class to read and write pizzas to the database.
 - The class is registered for dependency injection in *Program.cs* (line 10).

Entity Framework Core generated a few things, too:

- A *Migrations* folder was generated.
 - The migrations folder contains code to create the database schema.
- The SQLite database file *ContosoPizza.db* was generated.
 - If you have the [SQLite extension](#) installed (or you're using GitHub Codespaces), you can view the database by right-clicking the file and selecting **Open Database**. The database schema is shown in the **SQLite Explorer** tab of the Explorer pane.

Review the Razor Pages project structure

Everything else in the project is unchanged from when the project was created. The following table describes the project structure that was generated by the `dotnet new webapp` command.

 Expand table

Name	Description
<i>Pages/</i>	Contains Razor Pages and supporting files. Each Razor page has a <i>.cshtml</i> file and a <i>.cshtml.cs</i> <code>PageModel</code> class file.
<i>wwwroot/</i>	Contains static asset files like HTML, JavaScript, and CSS.
<i>ContosoPizza.csproj</i>	Contains project configuration metadata, such as dependencies.
<i>Program.cs</i>	Serves as the app's entry point and configures app behavior, like routing.


Other noteworthy observations:

- **Razor page files and their paired `PageModel` class file**

Razor pages are stored in the *Pages* directory. As noted above, each Razor page has a *.cshtml* file and a *.cshtml.cs* `PageModel` class file. The `PageModel` class allows separation of a Razor page's logic and presentation, defines page handlers for requests, and encapsulates data properties and logic scoped to its Razor page.


- **The *Pages* directory structure and routing requests**

Razor Pages uses the *Pages* directory structure as the convention for routing requests. The following table shows how URLs map to filenames:

 Expand table

URL	Maps to Razor page
<code>www.domain.com</code>	<i>Pages/Index.cshtml</i>
<code>www.domain.com/Index</code>	<i>Pages/Index.cshtml</i>
<code>www.domain.com/Privacy</code>	<i>Pages/Privacy.cshtml</i>
<code>www.domain.com/Error</code>	<i>Pages/Error.cshtml</i>

Subfolders in the *Pages* directory are used to organize Razor pages. For example, if there were a *Pages/Products* directory, the URLs would reflect that structure:

 Expand table

URL	Maps to Razor page
<code>www.domain.com/Products</code>	<i>Pages/Products/Index.cshtml</i>
<code>www.domain.com/Products/Index</code>	<i>Pages/Products/Index.cshtml</i>

URL	Maps to Razor page
<code>www.domain.com/Products/Create</code>	<code>Pages/Products/Create.cshtml</code>

- **Layout and other shared files**

There are several files that are shared across multiple pages. These files determine common layout elements and page imports. The following table describes the purpose of each file.

[Expand table](#)

File	Description
<code>_ViewImports.cshtml</code>	Imports namespaces and classes that are used across multiple pages.
<code>_ViewStart.cshtml</code>	Specifies the default layout for all Razor pages.
<code>Pages/Shared/_Layout.cshtml</code>	This is the layout specified by the <code>_ViewStart.cshtml</code> file. Implements common layout elements across multiple pages.
<code>Pages/Shared/_ValidationScriptsPartial.cshtml</code>	Provides validation functionality to all pages.

Run the project for the first time

Let's run the project so we can see it in action.


1. Right-click on the `ContosoPizza` folder in the **Explorer** and select **Open in Integrated Terminal**. This opens a terminal window in the context of the project folder.
2. In the terminal window, enter the following command:

```
.NET CLI
dotnet watch
```

This command:

- Builds the project.
- Starts the app.
- Watches for file changes and restarts the app when it detects a change.

3. The IDE prompts you to open the app in a browser. Select **Open in Browser**.

 **Tip**

You can also open the app by finding the URL in the terminal window. Hold **Ctrl** and click the URL to open it in a browser.

4. Compare the rendered home page to `Pages/Index.cshtml` in the IDE:
 - Observe the combination of HTML, Razor Syntax, and C# code in the file.
 - Razor Syntax is denoted by `@` characters.
 - C# code is enclosed in `@{ }` blocks. Take note of the directives at the top of the file:
 - The `@page` directive specifies that this file is a Razor page.
 - The `@model` directive specifies the model type for the page (in this case, `IndexModel`, which is defined in `Pages/Index.cshtml.cs`).
 - Review the C# code block.
 - The code sets the value of the `Title` item within the `ViewData` dictionary to "Home page".
 - The `ViewData` dictionary is used to pass data between the Razor page and the `IndexModel` class.
 - At runtime, the `Title` value is used to set the page's `<title>` element.

Leave the app running in the terminal window. We'll use it in the upcoming units. Leave the browser tab with the running Contoso Pizza app, too. You'll make changes to the app and the browser will automatically refresh to display the changes.

Customize the landing page

Let's make a few changes to the landing page to make it more relevant to the pizza app.

1. In *Pages/Index.cshtml*, replace the code in the C# code block with the following code:

```
C#

ViewData["Title"] = "The Home for Pizza Lovers";
TimeSpan timeInBusiness = DateTime.Now - new DateTime(2018, 8, 14);
```

The preceding code:

- Sets the value of the `Title` item within the `ViewData` dictionary to "The Home for Pizza Lovers".
- Calculates the amount of time that has passed since the business opened.

2. Modify the HTML as follows:

- Replace the `<h1>` element with the following code:

```
CSHTML

<h1 class="display-4">Welcome to Contoso Pizza</h1>
```

- Replace the `<p>` element with the following code:

```
CSHTML

<p class="lead">The best pizza in town for @Convert.ToInt32(timeInBusiness.TotalDays) days!</p>
```

The preceding code:

- Changes the heading to "Welcome to Contoso Pizza".
- Displays the number of days that have passed since the business opened.
 - The `@` character is used to switch from HTML to Razor Syntax.
 - The `Convert.ToInt32` method is used to convert the `TotalDays` property of the `timeInBusiness` variable to an integer.
 - The `Convert` class is part of the `System` namespace, which is imported automatically by the `<ImplicitUsings>` element in the *ContosoPizza.csproj* file.

3. Save the file. The browser tab with the app automatically refreshes to display the changes. If you're using GitHub Codespaces, the file saves automatically, but you'll need to refresh the browser tab manually.

📌 Important

Keep a close eye on the terminal window with `dotnet watch` every time you save your file. Sometimes your code might contain what it calls a *rude edit*. This means that the code you changed can't be recompiled without restarting the application. If prompted to restart the app, press `y` (yes) or `a` (always). You can always stop the application by pressing **Ctrl+C** twice in the terminal window, and then restart it by running `dotnet watch` again.

You've made your first changes to a Razor page! In the next unit, you'll add a new page to the app to display a list of pizzas.

Next unit: Exercise - Add a new Razor Page

Continue >