100 XP

Exercise - Add a new Razor Page

10 minutes

In the previous unit, you obtained the source code for the Contoso Pizza project, and then you made some simple changes to the home page. In this unit, you'll add a new Razor page to the project.

Create the Pizza List page

To create a new Razor page, you'll use the .NET CLI.

1. Since the terminal is blocked by the `dotnet watch` command, open another terminal by right-clicking on the *ContosoPizza* folder in the **Explorer** and select **Open in Integrated Terminal**.

2. In the new terminal window, enter the following command:

   .NET CLI

   ```
   dotnet new page --name PizzaList --namespace ContosoPizza.Pages --output Pages
   ```

   The preceding command:

   - Creates these two files in the `ContosoPizza.Pages` namespace:
     - *PizzaList.cshtml* - the Razor page
     - *PizzaList.cshtml.cs* - the accompanying `PageModel` class
   - Stores both files in the project's *Pages* subdirectory.

3. In *Pages/PizzaList.cshtml*, add the following code inside the `@{ }` code block:

   razor

   ```
   ViewData["Title"] = "Pizza List 🍕";
   ```

   This sets the `<title>` element for the page.

4. At the end of the file, add the following code:

   razor

   ```
   <h1>Pizza List 🍕</h1>

   <!-- New Pizza form will go here -->

   <!-- List of pizzas will go here -->
   ```

   This adds a heading to the page, as well as two HTML comment placeholders for functionality you'll add later.

5. Save the file. If you're using GitHub Codespaces, the file saves automatically.

6. Return to the terminal running `dotnet watch` and press **Ctrl+R** to reload the app and detect the new files.

Add the Pizza List page to the navigation menu

This would be a good time to test the page, but the page can't be reached in the browser because it isn't yet linked in the navigation menu. You'll link it now.

1. Open *Pages/Shared/_Layout.cshtml*.

2. In the `<ul>` element with the `navbar-nav` class (starts on line 21), note the `<li>` elements that contain the links to the *Home* and *Privacy* pages. Add the following code to the end of the list, after the `<li>` element containing the *Privacy* link:

   razor

```
<li class="nav-item">
    <a class="nav-link text-dark" asp-area="" asp-page="/PizzaList">Pizza List 🍕</a>
</li>
```

This adds a link to the *PizzaList* page to the navigation menu.

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.

4. Select the *Pizza List* 🍕 link in the navigation menu. The Pizza List page appears.

Register the PizzaService class with the dependency injection container

The Pizza List page depends on the `PizzaService` object to retrieve the list of pizzas. You'll use dependency injection to provide the `PizzaService` object to the page. First, register the `PizzaService` class with the container.

1. Open *Program.cs*.

2. In the section that adds services to the container, add the following code:

```
C#

builder.Services.AddScoped<PizzaService>();
```

This code registers the `PizzaService` class with the dependency injection container. The `AddScoped` method indicates that a new `PizzaService` object should be created for each HTTP request. Now the `PizzaService` can be injected into any Razor page.

3. Save the file. If you're using GitHub Codespaces, the file saves automatically.

Display a list of pizzas

Let's modify the `PageModel` class for the Pizza List page to retrieve the list of pizzas from the `PizzaService` object and store it in a property.

1. Open *Pages/PizzaList.cshtml.cs*.

2. Add the following `using` statements to the top of the file:

```
C#

using ContosoPizza.Models;
using ContosoPizza.Services;
```

These statements import the `Pizza` and `PizzaService` types you'll use in the page.

3. Inside the `ContosoPizza.Pages` namespace block, replace the entire `PizzaListModel` class with the following code:

```
C#

public class PizzaListModel : PageModel
{
    private readonly PizzaService _service;
    public IList<Pizza> PizzaList { get;set; } = default!;

    public PizzaListModel(PizzaService service)
    {
        _service = service;
    }

    public void OnGet()
    {
        PizzaList = _service.GetPizzas();
    }
}
```

In the preceding code:

- A private readonly `PizzaService` named `_service` is created. This variable will hold a reference to a `PizzaService` object.

  - The `readonly` keyword indicates that the value of the `_service` variable can't be changed after it's set in the constructor.
- A `PizzaList` property is defined to hold the list of pizzas.
  - The `IList<Pizza>` type indicates that the `PizzaList` property will hold a list of `Pizza` objects.
  - `PizzaList` is initialized to `default!` to indicate to the compiler that it will be initialized later, so null safety checks aren't required.
- The constructor accepts a `PizzaService` object.
  - The `PizzaService` object is provided by dependency injection.
- An `OnGet` method is defined to retrieve the list of pizzas from the `PizzaService` object and store it in the `PizzaList` property.

> **Tip**
>
> If you need help understanding null safety, see Null safety in C#.

4. Save the file. If you're using GitHub Codespaces, the file saves automatically.

5. Return to the terminal running `dotnet watch` and press **Ctrl+R** to reload the app with the registered service and the new constructor for `PizzaListModel`.

Display the list of pizzas

Now that the page has access to the list of pizzas, you'll use that list to display the pizzas on the page.

1. Open *Pages/PizzaList.cshtml*.

2. Replace the `<!-- List of pizzas will go here -->` comment with the following code:

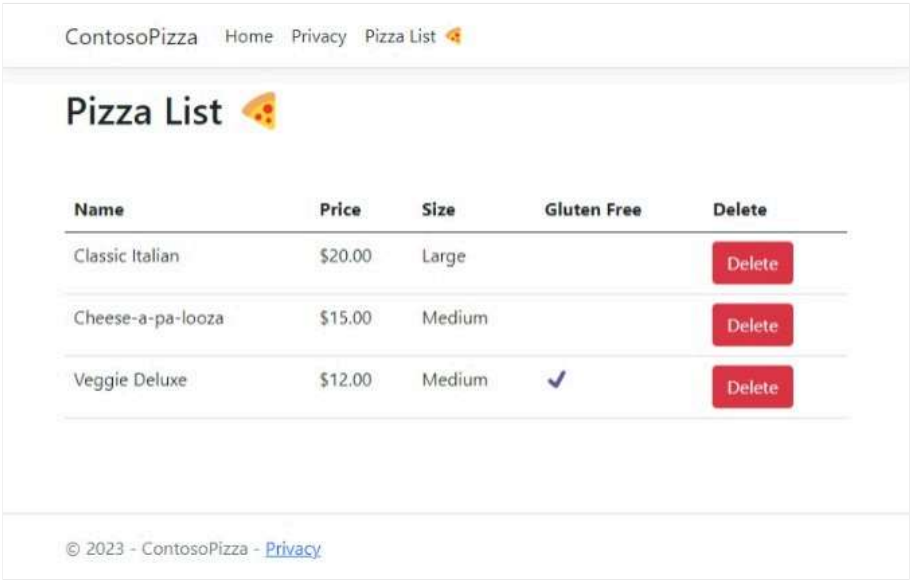
```razor
<table class="table mt-5">
    <thead>
        <tr>
            <th scope="col">Name</th>
            <th scope="col">Price</th>
            <th scope="col">Size</th>
            <th scope="col">Gluten Free</th>
            <th scope="col">Delete</th>
        </tr>
    </thead>
    <tbody>
    @foreach (var pizza in Model.PizzaList)
    {
        <tr>
            <td>@pizza.Name</td>
            <td>@($"{pizza.Price:C}")</td>
            <td>@pizza.Size</td>
            <td>@(pizza.IsGlutenFree ? "✔" : string.Empty)</td>
            <td>
                <form method="post" asp-page-handler="Delete" asp-route-id="@pizza.Id">
                    <button class="btn btn-danger">Delete</button>
                </form>
            </td>
        </tr>
    }
    </tbody>
</table>
```

In the preceding code:

- A `<table>` element is created to display the list of pizzas.
- A `<thead>` element is created to hold the table header.
- The `@foreach` statement inside the `<tbody>` iterates over the list of pizzas.
  - The `Model` property refers to the `PizzaListModel` object that was created in the code-behind file.
  - The `PizzaList` property refers to the `PizzaList` property that was defined in the code-behind file.
- Each iteration of the `@foreach` statement creates a `<tr>` element to hold the pizza data:
  - Razor syntax is used to display the pizza data in the `<td>` elements. This syntax is used to display the properties of the `Pizza` object that's stored in the `pizza` variable.
  - `Price` is formatted using C# string interpolation.
  - A ternary expression is used to display the value of the `IsGlutenFree` property as "✔" or a blank cell.
  - A form is created to delete the pizza.

- The `asp-page-handler` attribute indicates that the form should be submitted to the `Delete` handler in the code-behind file. You'll create that handler in a later unit.
- The `asp-route-id` attribute indicates that the `Id` property of the `Pizza` object should be passed to the `Delete` handler.

3. Save the file. In the browser, the Pizza List page refreshes with the list of pizzas. If you're using GitHub Codespaces, the file saves automatically, but you'll need to refresh the browser tab manually.



Good work! You've created a Razor page that displays a list of pizzas. In the next unit, you'll learn about tag helpers and page handlers.

Next unit: Understand tag helpers and page handlers

Continue >