200 XP

Exercise - Add new pizza form

10 minutes

In this unit, you'll finish the Pizza List page by adding a form to create new pizzas. You'll also add page handlers to handle the form submission and deletion of pizzas.

Add a form to create new pizzas

Let's start by adding properties to the `PizzaListModel` class to represent the user's input. You'll add the appropriate page handler, too.

1. Open *Pages\PizzaList.cshtml.cs* and add the following property to the `PizzaListModel` class:

   C#

   ```
   [BindProperty]
   public Pizza NewPizza { get; set; } = default!;
   ```

   In the preceding code:

   - A property named `NewPizza` is added to the `PizzaListModel` class.
     - `NewPizza` is a `Pizza` object.
   - The `BindProperty` attribute is applied to the `NewPizza` property.
     - The `BindProperty` attribute is used to bind the `NewPizza` property to the Razor page. When an HTTP POST request is made, the `NewPizza` property will be populated with the user's input.
   - The `NewPizza` property is initialized to `default!`.
     - The `default!` keyword is used to initialize the `NewPizza` property to `null`. This prevents the compiler from generating a warning about the `NewPizza` property being uninitialized.

2. Now add the page handler for HTTP POST. In the same file, add the following method to the `PizzaListModel` class:

   C#

   ```
   public IActionResult OnPost()
   {
       if (!ModelState.IsValid || NewPizza == null)
       {
           return Page();
       }

       _service.AddPizza(NewPizza);

       return RedirectToAction("Get");
   }
   ```

   In the preceding code:

   - The `ModelState.IsValid` property is used to determine if the user's input is valid.
     - The validation rules are inferred from attributes (such as `Required` and `Range`) on the `Pizza` class in *Models\Pizza.cs*.
     - If the user's input is invalid, the `Page` method is called to re-render the page.
   - The `NewPizza` property is used to add a new pizza to the `_service` object.
   - The `RedirectToAction` method is used to redirect the user to the `Get` page handler, which will re-render the page with the updated list of pizzas.

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

4. Return to the terminal running `dotnet watch` and press **Ctrl+R** to reload the app.

Now that there's a page handler to handle the form submission, let's add the form to the Razor Page.

1. Open *Pages\PizzaList.cshtml* and replace the `<!-- New Pizza form will go here -->` with the following code:

   razor

   ```
   <form method="post">
   <div asp-validation-summary="ModelOnly" class="text-danger"></div>
   ```

```
<div class="form-group">
    <label asp-for="NewPizza.Name" class="control-label"></label>
    <input asp-for="NewPizza.Name" class="form-control" />
    <span asp-validation-for="NewPizza.Name" class="text-danger"></span>
</div>
<div class="form-group">
    <label asp-for="NewPizza.Size" class="control-label"></label>
    <select asp-for="NewPizza.Size" class="form-control" id="PizzaSize">
        <option value="">-- Select Size --</option>
        <option value="Small">Small</option>
        <option value="Medium">Medium</option>
        <option value="Large">Large</option>
    </select>
    <span asp-validation-for="NewPizza.Size" class="text-danger"></span>
</div>
<div class="form-group form-check">
    <label class="form-check-label">
        <input class="form-check-input" asp-for="NewPizza.IsGlutenFree" /> @Html.DisplayNameFor(model => model.NewPizza.IsGlutenFree)
    </label>
</div>
<div class="form-group">
    <label asp-for="NewPizza.Price" class="control-label"></label>
    <input asp-for="NewPizza.Price" class="form-control" />
    <span asp-validation-for="NewPizza.Price" class="text-danger"></span>
</div>
<div class="form-group">
    <input type="submit" value="Create" class="btn btn-primary" />
</div>
</form>
```

In the preceding code:

- The `asp-validation-summary` attribute is used to display validation errors for the entire model.
- Each form field (`<input>` and `<select>` elements) and each `<label>` is bound to the corresponding `NewPizza` property using the `asp-for` attribute.
- The `asp-validation-for` attribute is used to display any validation errors for each form field.
- The `@Html.DisplayNameFor` method is used to display the display name for the `IsGlutenFree` property. This is a Razor helper method that's used to display the display name for a property. Doing the label this way ensures that the checkbox is selected when the user clicks the label.
- A submit button labeled `Create` is added to the form to post the form data to the server. At runtime, when the user selects this **Create** button, the browser sends the form as an HTTP POST request to the server.

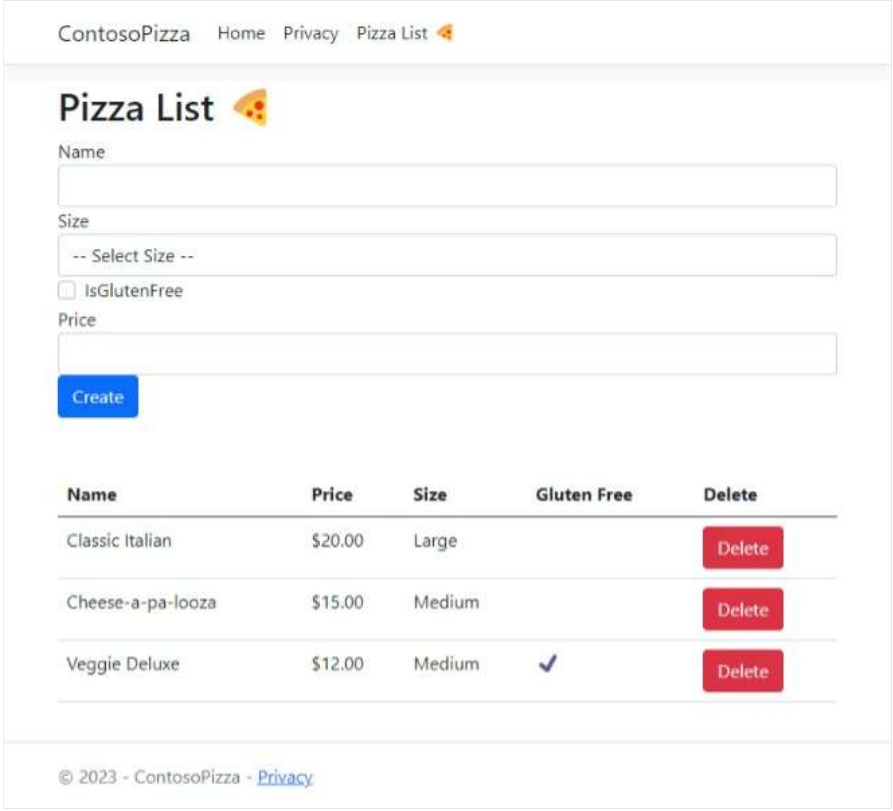
2. At the bottom of the page, add the following code:

razor

```
@section Scripts {
<partial name="_ValidationScriptsPartial" />
}
```

This injects the client-side validation scripts into the page. The client-side validation scripts are used to validate the user's input before the form is submitted to the server.

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



4. Enter a new pizza and select the **Create** button. The page should refresh and display the new pizza in the list.

Add a page handler to delete pizzas

There's one last piece to add to the Pizza List page: a page handler to delete pizzas. The buttons to delete pizzas are already on the page, but they don't do anything yet.

1. Back in *Pages\PizzaList.cshtml.cs*, add the following method to the `PizzaListModel` class:

```
C#

public IActionResult OnPostDelete(int id)
{
    _service.DeletePizza(id);

    return RedirectToAction("Get");
}
```

In the preceding code:

- The `OnPostDelete` method is called when the user clicks the **Delete** button for a pizza.
  - The page knows to use this method because the `asp-page-handler` attribute on the **Delete** button in *Pages\PizzaList.cshtml* is set to `Delete`.
- The `id` parameter is used to identify the pizza to delete.
  - The `id` parameter is bound to the `id` route value in the URL. This is accomplished with the `asp-route-id` attribute on the **Delete** button in *Pages\PizzaList.cshtml*.
- The `DeletePizza` method is called on the `_service` object to delete the pizza.
- The `RedirectToAction` method is used to redirect the user to the `Get` page handler, which will re-render the page with the updated list of pizzas.

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

3. Test the **Delete** button for a pizza. The page should refresh and the selected pizza should be removed from the list.

Congratulations! You've successfully created a Razor Page that displays a list of pizzas, allows the user to add new pizzas, and also allows the user to delete pizzas.

Check your knowledge

1. What method would you use to handle form submission in a `PageModel`? *

- ○ Use an `OnPost` (or `OnPostAsync`) method.

  ✔ Correct! Either an **`OnPost`** or **`OnPostAsync`** method should be used to process form submissions.

- ◉ Use an `OnGet` method.

  ✕ Incorrect. An **`OnGet`** method should be used to load information, but not to modify information.

- ○ Use a `DataAnnotation` to handle form submission.

Next unit: Summary

Continue >