

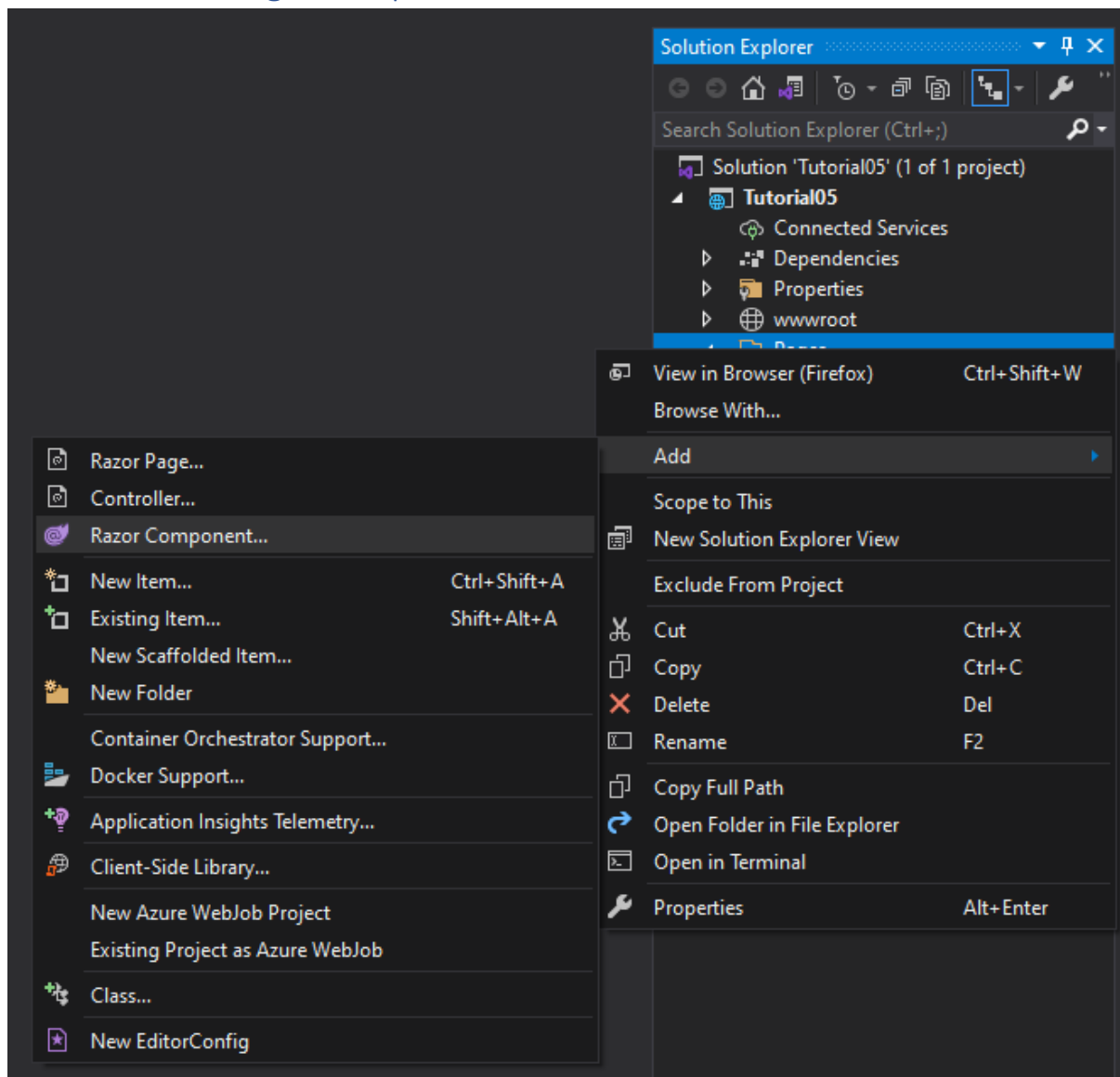
Tutorial 5

Outcomes

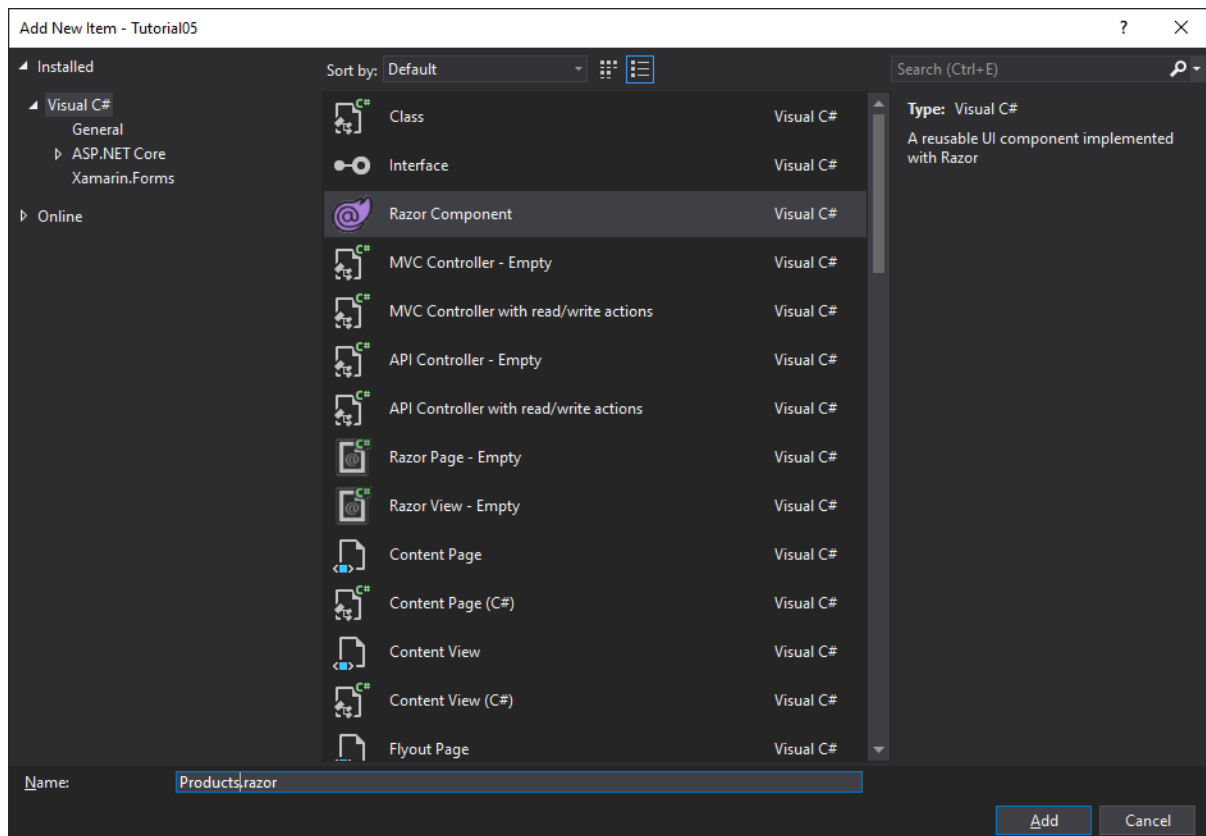
- Create a new Page/Component
- One-way binding
- Two-way binding
- Dynamic UI
 - Conditional rendering
 - Generate UI controls using a loop
- Style

First, we will start by creating a new Blazor Web Assembly project.

Create a new Page/Component



Go right click on the Pages folder -> Add -> Blazor Component.



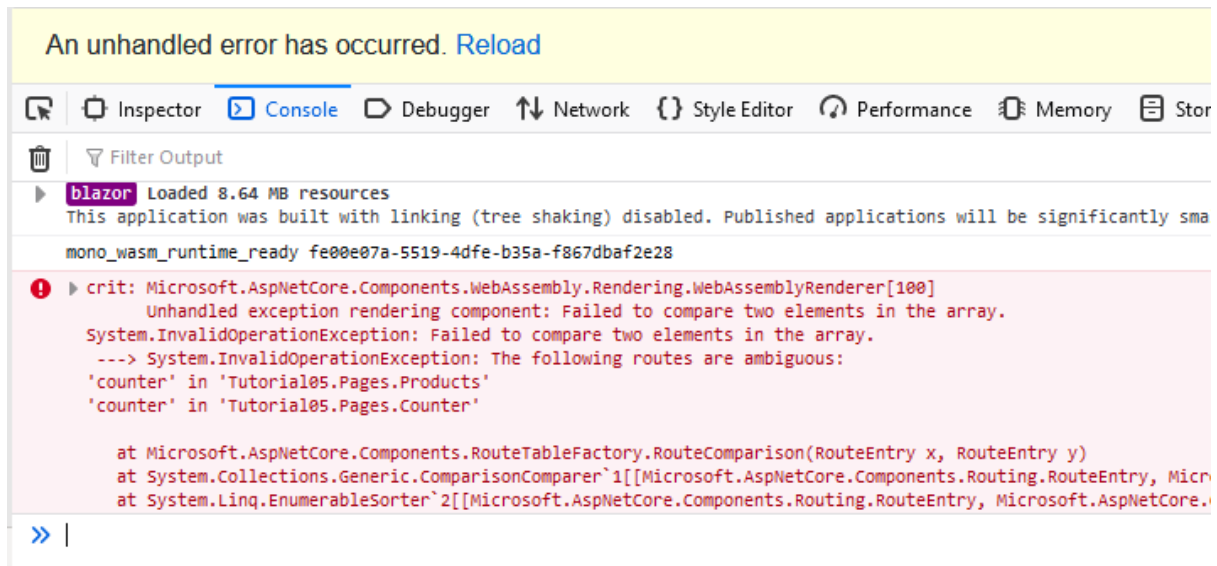
Call this component Products.razor and click on the Add button.

What we created is a component and the question is how do we turn it into a page? In Blazor there is no difference between pages and components.

A page is a component we can access by navigating to a specific URL. Let us add @page "/products" to Products.razor.

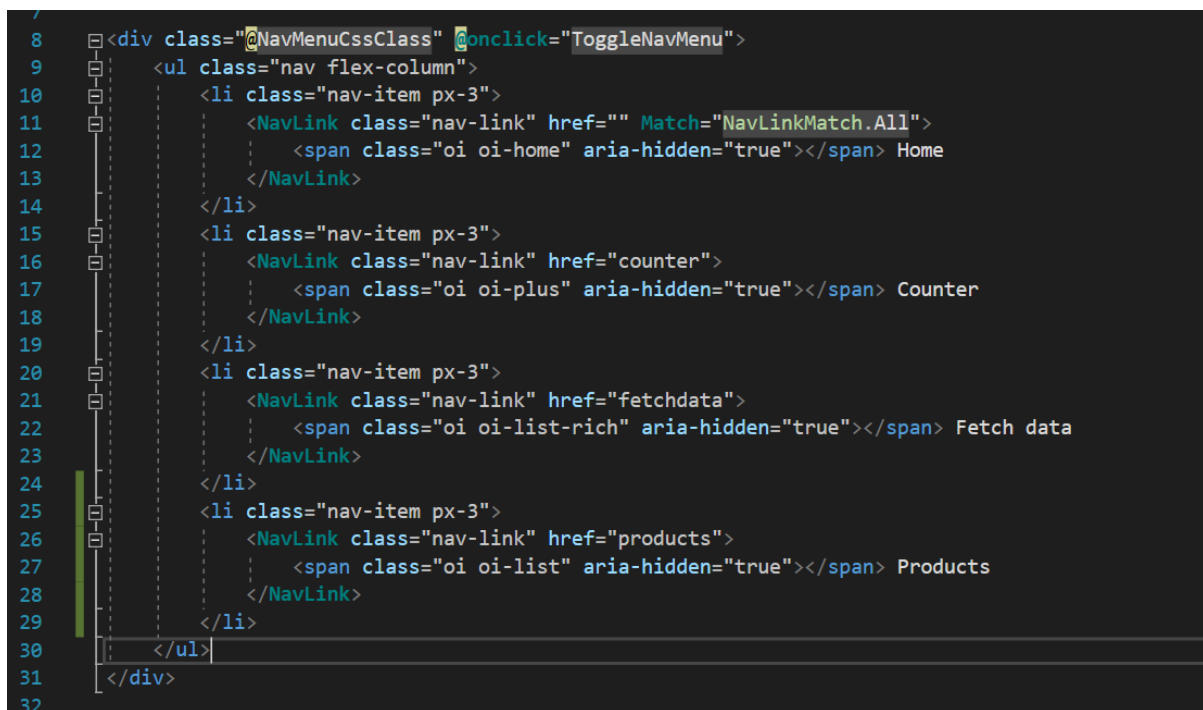
```
1  @page "/products"
2
3  <h1>Products</h1>
4
5  @code {
6
7  }
8
```

If we use the URL that already exists, like /counter for example, we will get a runtime error.



Add an item to the navigation menu

Next let's add a new item to the navigation. Go to NavMenu.razor and add a new item.



When we run the project we can see a new menu item showing up and if we click on it we get to the /products page.

To show you there is no difference between a page and a component let us nest the Counter component in Products.

```
Products.razor X Counter.razor
@page "/products"

<h1>Products</h1>

<Counter></Counter>

@code {
}
```

After we run the project we can see the counter component showing up on the product page. We can nest multiple Counters and we will see that each of them will keep the state of currentCount.

One-way binding

When it comes to one-way binding we already have an example of that in the counter component. We read a value from the currentCount variable and display it on a label whenever we click on the button.

```
Counter.razor X
1  @page "/counter"
2
3  <h1>Counter</h1>
4
5  <p>Current count: @currentCount</p>
6
7  <button class="btn btn-primary" @onclick="IncrementCount">Click me</button>
8
9  @code {
10     private int currentCount = 0;
11
12     private void IncrementCount()
13     {
14         currentCount++;
15     }
16 }
17
```

Once the button is clicked, currentCount is incremented and <p> tag is automatically updated.

Two-way binding

An example of two-way binding is reading from and setting a value into a variable.

Let us create a variable name and bind this variable to an input field using `@bind-value`. We can also control when the change event delegate should trigger with `@bind-value:event="oninput"`.

```
Products.razor Counter.razor Index.razor
@page "/products"

<h1>Products</h1>

<Counter></Counter>
<Counter></Counter>

<hr />

<h2>Two-way binding</h2>

<p>
    Enter name: <input type="text" @bind-value="name" @bind-value:event="oninput" />
</p>
<p>
    Entered name: @name
</p>

@code {
    private string name = "Milos";
}
```

Dynamic UI

As you can see we can easily modify the DOM of our page with C# code. Now, we can do some more complex things such as conditional rendering and loops.

Conditional rendering

Let us say we only want to show an option to add a product to the list of products if the user is logged in. We can create a checkbox that we will use to simulate if the user is logged in.

```
<p>
    Is user logged in? <input type="checkbox" @bind="isLoggedIn">
</p>
<p>
    Logged in: @isLoggedIn
</p>
```

Next, we need an input field and a button, but we want to show these only in case the user is logged in, `@if` and wrap the HTML code. Also, we will need a variable that we will bind to the input field to get the name of the product we want to add.

```

@if(isLoggedIn)
{
    <p>
        <input type="text" @bind-value="@productToAdd" @bind-value:event="oninput" />
    </p>
    <p>
        Product name: @productToAdd
    </p>
    <p>
        <button>Add item</button>
    </p>
}

```

Now, we can create a method that we will trigger and a list which we will use to temporarily store the products. We also need add @onclick to our button so that can trigger the method.

```

Products.razor  Counter.razor  Index.razor
<p>
    Product name: @productToAdd
</p>
<p>
    <button class="btn btn-primary" @onclick="AddItem">Add item</button>
</p>
}
</div>

@code {
    private string name = "Milos";
    private bool isLoggedIn;
    private string productToAdd;
    private List<string> products = new List<string>() { "Apples, Oranges, Grapes" };

    private void AddItem()
    {
        products.Add(productToAdd);
        productToAdd = "";
    }
}

```

Loop

In our list we added some default products but how do we display them, including the new products we add? This is where loops come into play. We can use and HTML tags to construct a list of items. Then we can loop through products list and for each product we can generate one .

```

<ul>
    @foreach (var product in products)
    {
        <li>@product</li>
    }
</ul>

```

The code from the image above will generate the following HTML.

```
<ul>

<li>Apples</li>

<li>Oranges</li>

<li>Grapes</li>

</ul>
```

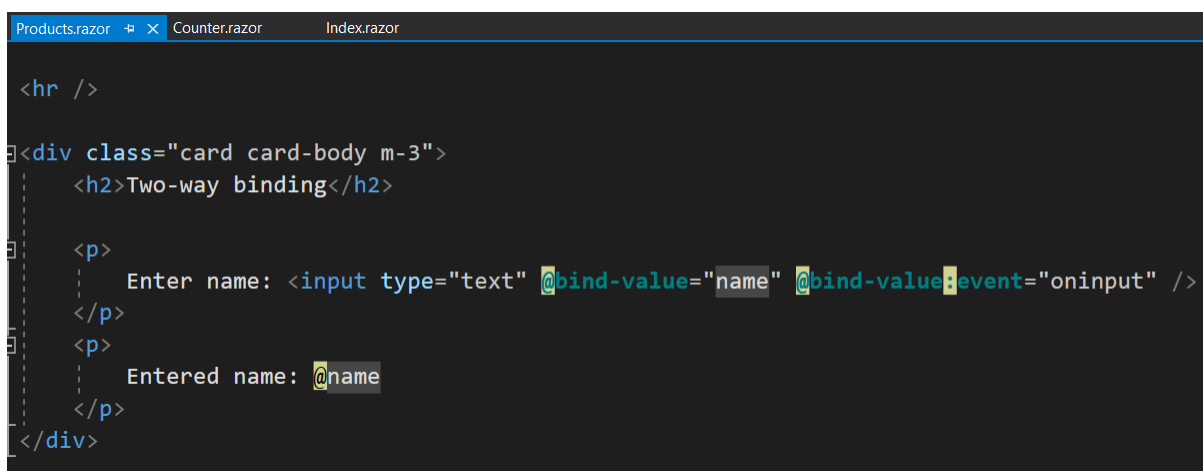
Style

To improve visual appearance, we can add some CSS classes. Blazor integrates Bootstrap which is one of the most popular CSS libraries.

Let us wrap the two-way binding content as well as the user logged in content in a div and apply some style using Bootstrap CSS classes.

```
<div class="card card-body m-3"></div>
```

We can also make our button look nicer with `<button class="btn btn-primary">Add item</button>`.

A screenshot of a code editor with three tabs: 'Products.razor', 'Counter.razor', and 'Index.razor'. The 'Counter.razor' tab is active. The code in the editor is as follows:

```
<hr />

<div class="card card-body m-3">
  <h2>Two-way binding</h2>

  <p>
    Enter name: <input type="text" @bind-value="name" @bind-value:event="oninput" />
  </p>
  <p>
    Entered name: @name
  </p>
</div>
```

Bonus tasks

Counter component replacement

Counter component is used as a page and it has a title. Create a new component called `<NewCounter>` with the same content as Counter but without the title. Make the NewCounter component accept the start value for the counter. Replace `<Counter>` with `<NewCounter>` in the Product page and pass the start value.

Product list item

Move the code from the loop in which we show products to a separate component called ProductItem. Pass the name of the product to the component and append the current date to it.

Useful Links

- <https://docs.microsoft.com/en-us/aspnet/core/blazor/components/data-binding?view=aspnetcore-5.0>