Walkthrough 20 - Blazor WebAssembly Introduction

Setup - BlazorWasmIntro

This lab will explore Blazor WebAssembly.

1. Start Visual Studio.
2. Click Create a new project.
3. Set language to **C#** and project type to **Web**.
4. Select the Blazor WebAssembly App template, click Next.
5. Set Project name to **BlazorWasmIntro**.
6. Set Location to a **folder of your choosing**.
7. Ensure Place solution and project in the same directory is not selected, click Next.
8. Set version to **.NET 5.0**, unselect Configure for HTTPS, unselect ASP.NET Core hosted, unselect Progressive Web Application, click Create.
9. Run the site and test each of the 3 pages. The functionality is identical to BlazorServerIntro.
10. Close the browser.

index.html

1. Open wwwroot / index.html.
2. In the body tag, notice the div tag for app.

App.razor

1. Open App.razor.
2. This file is the same as in BlazorServerIntro.
3. Notice the RouteView component and the DefaultLayout="@typeof(MainLayout)" attribute.

MainLayout.razor

1. Open Shared / MainLayout.razor.
2. This file is the same as in BlazorServerIntro.
3. Notice the @Body property. This is where other Blazor components will render.
4. In the sidebar div, notice the NavMenu tag.

NavMenu.razor

1. Open Shared / NavMenu.razor.
2. This file is the same as in BlazorServerIntro.
3. Notice the NavLink components.

Index.razor

1. Open Pages / Index.razor file.
2. This file is the same as in BlazorServerIntro.

Counter.razor

1. Open Pages / Counter.razor file.
2. This file is the same as in BlazorServerIntro.

FetchData.razor

1. Open Pages / FetchData.razor file.
2. This file is different than in BlazorServerIntro.
3. Instead of injecting a C# class, the HttpClient class is injected.
4. Recall, we used the HttpClient class to call a remote web service earlier in the semester.
5. Scroll to the @code section.
6. Notice that the WeatherForecast class is defined here.
7. Also notice that the GetFromJsonAsync method of HttpClient is called to retrieve the JSON from a static file in the wwwroot / sample-data folder.

Todo.razor

1. Right-click the Page folder and select Add / Razor Component..., name it Todo.razor.
2. Add a page directive and change the heading from h3 to h1.
3.
```
@page "/todo"
```

```
<h3h1>Todo</h3h1>

@code {

}
```

NavMenu.razor

1. Open Shared / NavMenu.razor.
2. Add a menu item for Todo below Fetch data.
3. Blazor uses the **Open Iconic** (https://useiconic.com/open) icon set.
4.

```
...
        <li class="nav-item px-3">
            <NavLink class="nav-link" href="fetchdata">
                <span> class="oi oi-list-rich" aria-hidden="true"></span> Fetch data
            </NavLink>
        </li>
        <li class="nav-item px-3">
            <NavLink class="nav-link" href="todo">
                <span> class="oi oi-list" aria-hidden="true"></span> Todo
            </NavLink>
        </li>
    </ul>
</div>
...
```

5. Save the file.
6. Run the site, click the Todo button in the nav.

TodoItem.cs

1. Create a **Models** folder.
2. Add a class named **TodoItem** to the Models folder.
3. Add two properties to the class.
4.

```
public class TodoItem
{
    public string Title { get; set; }
    public bool IsDone { get; set; }
}
```

5. Save the file.

_Imports.razor

1. Open _Imports.razor.
2. Add the Models namespace.
3.

```
@using System.Net.Http
@using System.Net.Http.Json
@using Microsoft.AspNetCore.Components.Forms
@using Microsoft.AspNetCore.Components.Routing
@using Microsoft.AspNetCore.Components.Web
@using Microsoft.AspNetCore.Components.Web.Virtualization
@using Microsoft.AspNetCore.Components.WebAssembly.Http
@using Microsoft.JSInterop
@using BlazorWasmIntro
@using BlazorWasmIntro.Shared
@using BlazorWasmIntro.Models
```

4. Save the file.

Todo.razor

1. Add a field for todo items.
2.

```
@page "/todo"

<h1>Todo</h1>

@code {
    private List<TodoItem> todos = new();
}
```

3. Add an unordered list and a foreach loop to render each todo item as a list item.
4.

```
@page "/todo"

<h1>Todo</h1>

<ul>
    @foreach (var todo in todos)
    {
        <li>@todo.Title</li>
    }
</ul>

@code {
    private List<TodoItem> todos = new();
}
```

5. Add a text input and a button to facilitate the creation of new items.
6.
```
@page "/todo"

<h1>Todo</h1>

<ul>
    @foreach (var todo in todos)
    {
        <li>@todo.Title</li>
    }
</ul>

<input placeholder="Something todo" />
<button class="btn btn-primary">Add todo</button>

@code {
    private List<TodoItem> todos = new();
}
```
7. Save the file. Refresh the browser.
8. Add a method to handle the button clicks.
9.
```
...
@code {
    private List<TodoItem> todos = new();

    private void AddTodo()
    {

    }
}
```
10. Register the method with the button.
11.
```
...
<button class="btn btn-primary" @onclick="AddTodo">Add todo</button>
...
```
12. Add a string for the new todo item.
13.
```
...
@code {
    private List<TodoItem> todos = new();
    private string newTodo;

    private void AddTodo()
    {

    }
}
```
14. Bind the variable to the text input.
15.
```
...
<input placeholder="Something todo" @bind="newTodo" />
...
```
16. Implement AddTodo.
17.
```
private void AddTodo()
{
    if (!string.IsNullOrWhiteSpace(newTodo))
    {
        todos.Add(new TodoItem { Title = newTodo });
        newTodo = "";
    }
}
```
18. Save the file. Refresh the browser, add some todo items.
19. Update the list item to have a checkbox and make each item editable.
20.
```
...
<ul>
    @foreach (var todo in todos)
    {
        <li>
            @todo.Title
            <input type="checkbox" @bind="todo.IsDone" />
            <input @bind="todo.Title" />
        </li>
    }
</ul>
...
```
21. Save the file. Refresh the browser, add some todo items.
22. Update the header to show a count of the number of items that aren't done.
23.
```
@page "/todo"

<h1>Todo (@todos.Count(todo => !todo.IsDone))</h1>
...
```
24. Save the file. Refresh the browser, add some todo items.
25. From the File menu, close the solution.

Setup - BlazorWasmWithBackend

1. Click Create a new project.
2. From the Recent project templates area, click the Blazor WebAssembly App template, click Next.
3. Set Project name to **BlazorWasmWithBackend**.
4. Set Location to a **folder of your choosing**.
5. Ensure Place solution and project in the same directory is not selected, click Next.
6. Set version to **.NET 5.0**, unselect Configure for HTTPS, select ASP.NET Core hosted, unselect Progressive Web Application, click Create.
7. Notice that 2 projects have been created
8. Run the site and test each of the 3 pages. The functionality is identical to BlazorServerIntro and BlazorWasmIntro.
9. Close the browser.

BlazorWasmWithBackend.Shared

1. Note the WeatherForecast class is here.

WeatherForecastController.cs

1. Open BlazorWasmWithBackend.Server / Controllers / WeatherForecastController.cs.
2. Note that it is a Web API Controller.
3. Modify the Get method to accept a parameter to determine the number of forecasts that will be returned.
4.
```
[HttpGet]
public IEnumerable<WeatherForecast> Get(int num = 5)
{
    var rng = new Random();
    return Enumerable.Range(1, num).Select(index => new WeatherForecast
    {
        Date = DateTime.Now.AddDays(index),
        TemperatureC = rng.Next(-20, 55),
        Summary = Summaries[rng.Next(Summaries.Length)]
    })
    .ToArray();
}
```
5. Save the file.

FetchData.razor

1. Open BlazorWasmWithBackend.Client / Pages / FetchData.razor.
2. Add a variable for number of forecasts to retrieve.
3.
```
...
@code {
    private WeatherForecast[] forecasts;
    private string numberOfForecasts;

    protected override async Task OnInitializedAsync()
    {
        forecasts = await Http.GetFromJsonAsync<WeatherForecast[]>("WeatherForecast");
    }
}
```
4. Add a method to get the forecasts.
5.
```
...
@code {
    private WeatherForecast[] forecasts;
    private string numberOfForecasts;

    protected override async Task OnInitializedAsync()
    {
        forecasts = await Http.GetFromJsonAsync<WeatherForecast[]>("WeatherForecast");
    }

    private async Task GetForecasts()
    {
        int.TryParse(numberOfForecasts, out int num);
        if (num <= 0)
        {
            num = 5;
        }
        numberOfForecasts = num.ToString();
        forecasts = await Http.GetFromJsonAsync<WeatherForecast[]>($"WeatherForecast/?num={num}");
    }
}
```
6. Add a text input and a button to retrieve the forecasts.
7.
```
@page "/fetchdata"
@using BlazorWasmWithBackend.Shared
@inject HttpClient Http

<h1>Weather forecast</h1>

<p>This component demonstrates fetching data from the server.</p>
<p>
    Number of forecasts to retrieve
    <input @bind="numberOfForecasts" />
    <button class="btn btn-primary" @onclick="GetForecasts">Get Forecasts</button>
```

```
</p>

@if (forecasts == null)
...
```

8. Save the file. Run the site. Try retrieving a different number of forecasts, including invalid input.
9. From the File menu, close the solution.

Setup - BlazorWasmPWA

1. Click Create a new project.
2. From the Recent project templates area, click the Blazor WebAssembly App template, click Next.
3. Set Project name to **BlazorWasmPWA**.
4. Set Location to a **folder of your choosing**.
5. Ensure Place solution and project in the same directory is not selected, click Next.
6. Set version to **.NET 5.0**, unselect Configure for HTTPS, unselect ASP.NET Core hosted, select Progressive Web Application, click Create.
7. Notice that 2 projects have been created

index.html

1. Open wwwroot / index.html.
2. In the head section, notice the manifest and icon references.
3. Scroll to the bottom of the file and notice script tag for the service-worker.

manifest.json

1. Open wwwroot / manifest.json.
2. Inspect it.

service-worker.js

1. Open wwwroot / service-worker.js
2. Notice how simple it is. This version of the file is used during development.
3. In Solution Explorer, expand service-worker.js and select service-worker.published.js. Briefly inspect it. This version of the file is used in production.

Running the PWA

1. Run the site and test each of the 3 pages. The functionality is identical to BlazorServerIntro, BlazorWasmIntro and BlazorWasmWithBackend.
2. Depending on the browser, there will be a button to install the app. In Chrome and Edge, the button is at the right-end of the address bar.
3. Click the button and click Install. The app will appear in its own window. When installing, you will be given options on how to install the app. Select pin to Start Menu and any other options you wish.
4. Close the browser and close the app.
5. While in development, as long as your Visual Studio web server is running, represented by an icon in the taskbar notification area, the PWA will work.
6. Open the app again.
7. Close the app.
8. Access the Windows menu, right-click BlazorWasmPWA and select uninstall.