



Blazor



Buy me a coffee

Contents

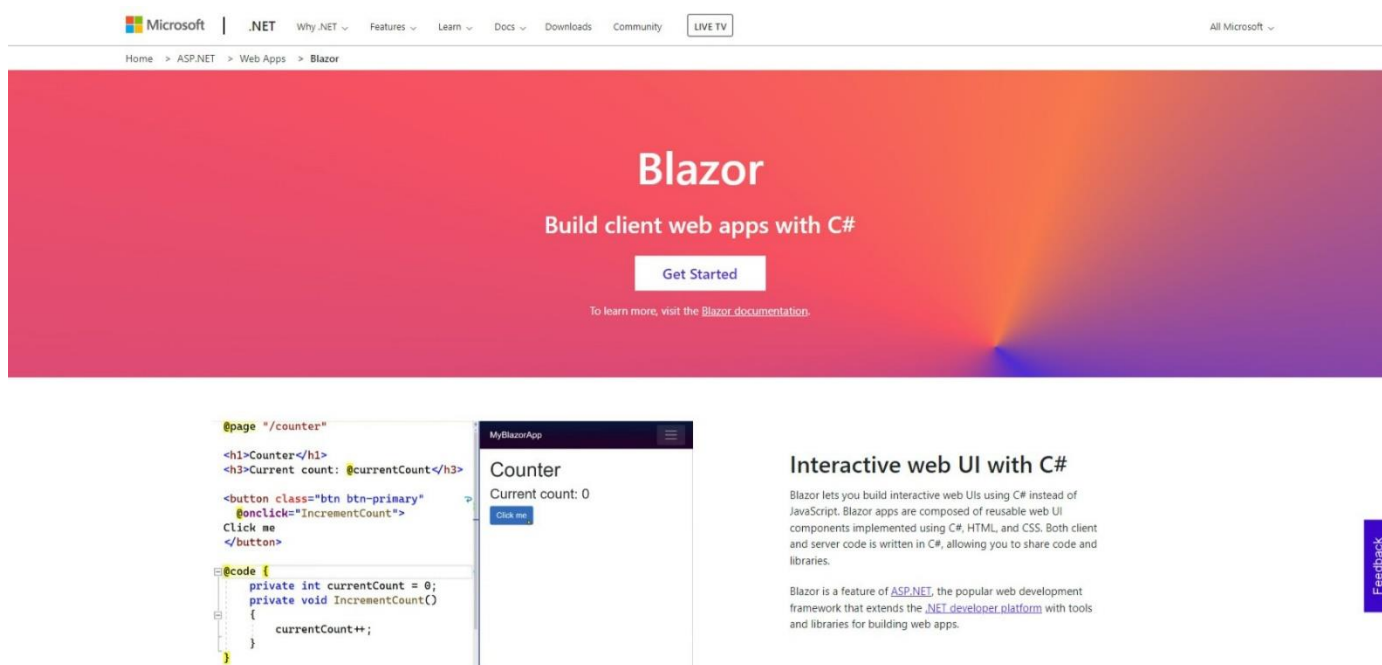
Contents

Introduction.....	2
What is Blazor?	2
What is Visual Studio Code?	3
Setup and Start	4
Blazor.....	4
Visual Studio Code.....	7
Components.....	12
Routing.....	12
Markup	14
Styles and CSS	15
Images	16
Binding and Events.....	17
Binding	17
Events.....	18
Conditions and Collections	23
Forms	26
Dependency Injection.....	28

Introduction

What is Blazor?

Blazor allows you to build interactive client web applications using **C#** instead of **JavaScript** with Applications composed using reusable **Components** using **C#**, **HTML** and **CSS** that supports both **Client** using **Web Assembly** and **Server** using **ASP.NET** created by **Microsoft**.

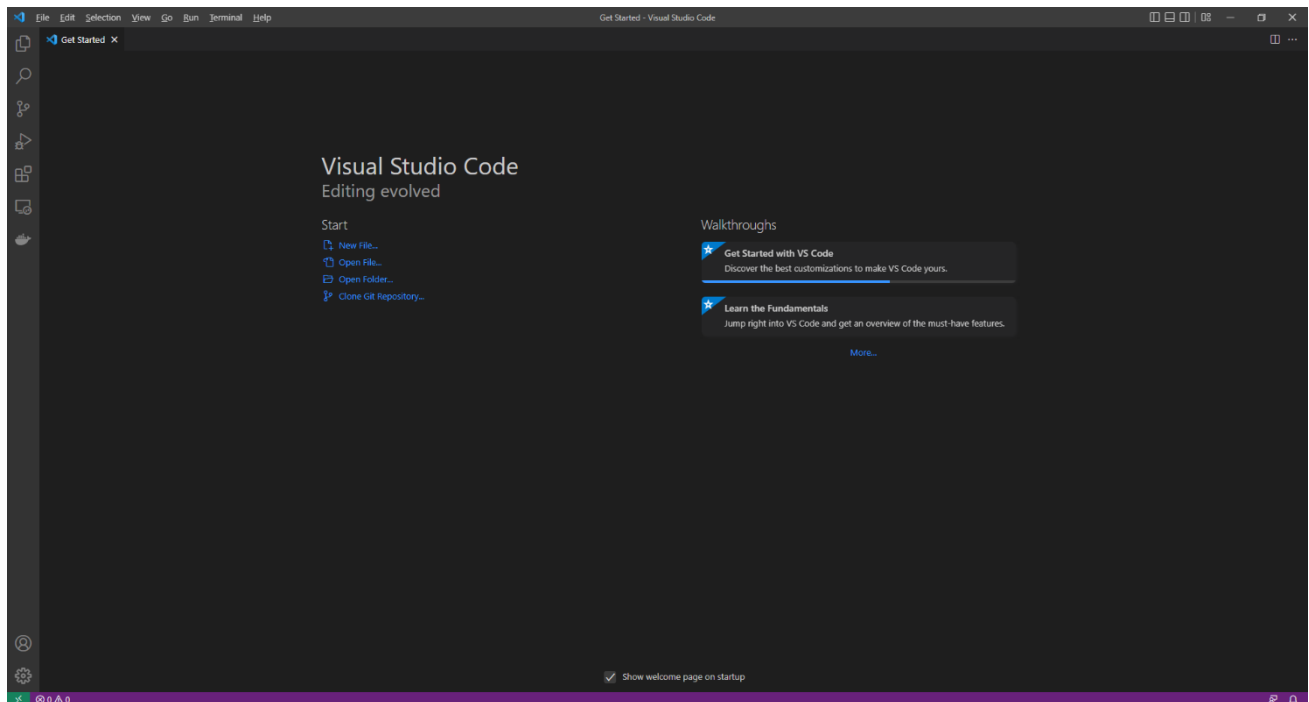


Blazor uses **C#**, which supports many features common in modern programming languages. **C#** is used with **.NET**, which not only supports developing Applications in **Blazor** but you can also use **C#** to develop Applications for web, mobile, desktop, games, IoT and more. For more information about using **.NET** along with documentation, examples and more then visit dot.net.

Blazor allows you to develop Applications that run on the **Server** where events are passed using **SignalR**, you can run your **C#** code directly in the **Browser** using **WebAssembly** and can re-use code between the **Client** and **Server**. **Blazor** also enables **Native** cross-platform Applications using **Blazor Hybrid** with **.NET MAUI**. and you can still work with **JavaScript** in the **Browser** from **Blazor** when needed. You can find out more about **Blazor** including documentation, examples and more at blazor.net.

What is Visual Studio Code?

Visual Studio Code will help create **Blazor** applications even more easily. **Visual Studio Code** is a free **Integrated Development Environment or IDE** created by **Microsoft**.

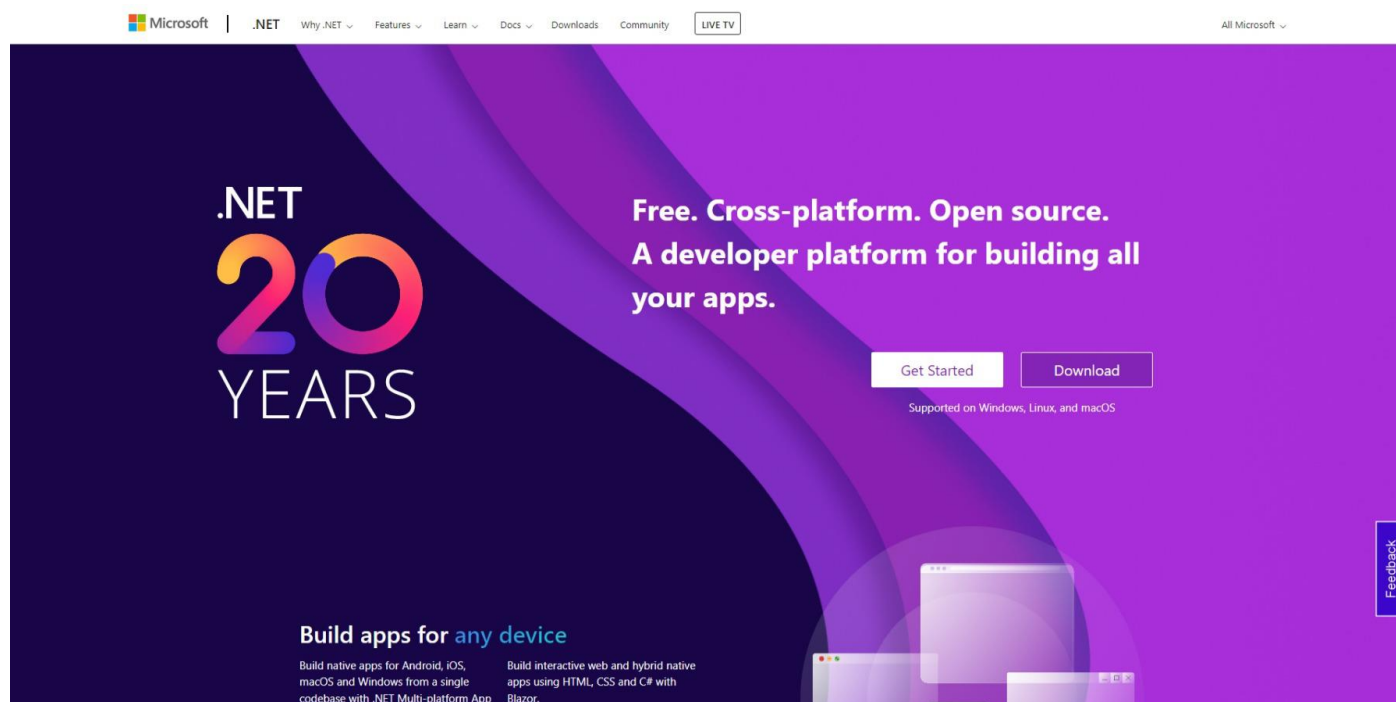


Visual Studio Code supports syntax highlighting which will add colours to certain parts of the text and make it easy to make sure everything is being entered correctly when writing **Blazor** Applications. You can use **Visual Studio Code** to edit any other **C#**, **Razor**, **CSS**, **HTML** and more, making more than just creating **Blazor** applications straightforward. If you want to find out more about **Visual Studio Code** along with documentation, extensions and more you can visit code.visualstudio.com.

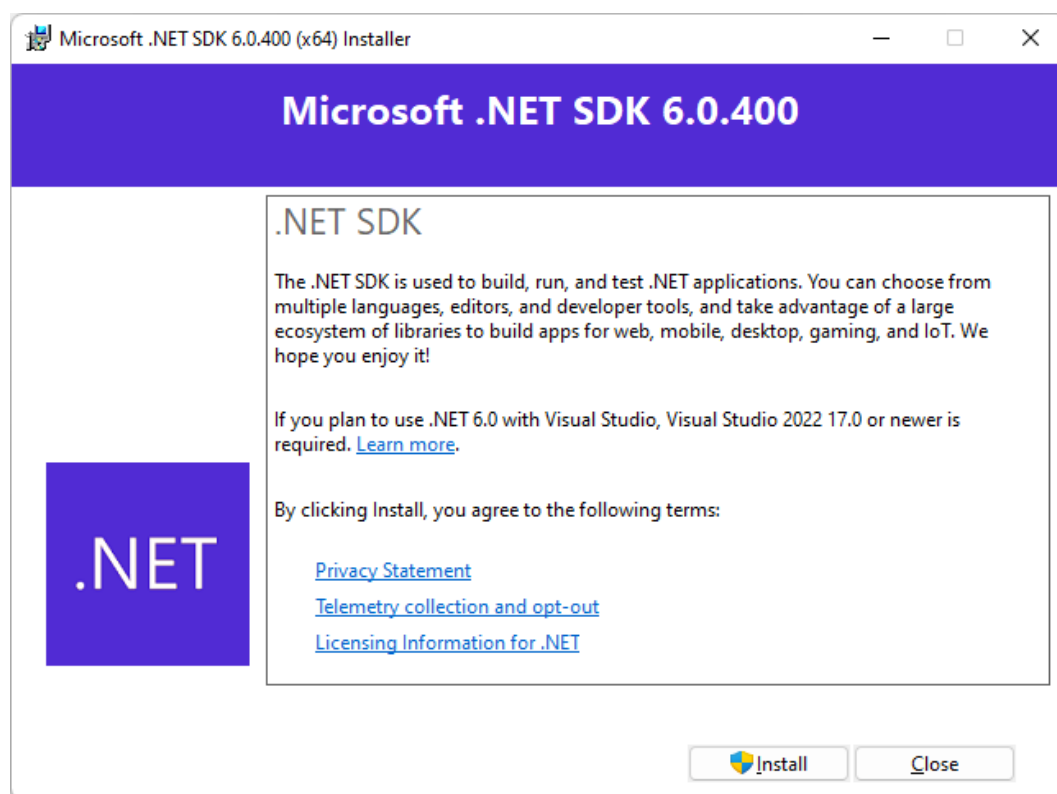
Setup and Start

Blazor

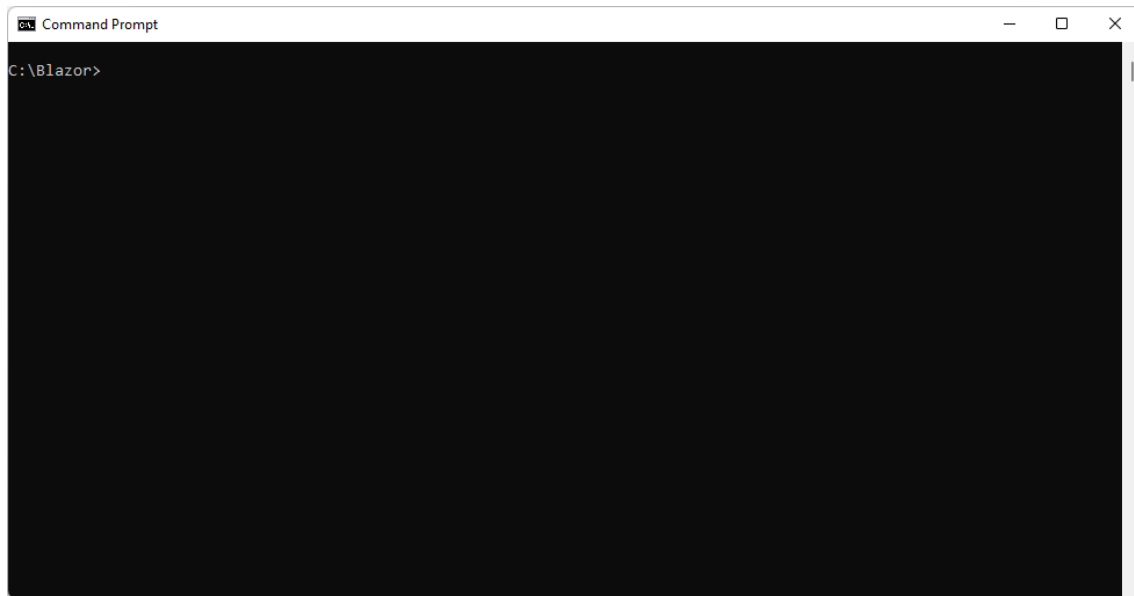
Blazor requires the latest version of the **.NET SDK** which if you have it already you can **Download** the version for your Platform such as **Windows** from dot.net



Once you have **Downloaded** then **Install** the **.NET SDK** by following the steps in the **Installation Wizard**



Once the **.NET SDK** has been **Installed**, or if it was already **Installed**, then if using **Windows** you need to go to **Start** then search for **Command Prompt** and then select it.



Once in the **Command Prompt** you will need to create a new **Folder**, you can use **mkdir** followed by the name of the **Folder** e.g. *Blazor* and then press **Enter**.

```
mkdir Blazor
```

Then you will need to switch to this **Folder**, to do this from the **Command Prompt**, type in the following command and then press **Enter**:

```
cd Blazor
```

Once in this **Folder** you can create a new **Blazor** Application using the **.NET CLI** that was installed as part of the **.NET SDK**. To do this, while still in the **Command Prompt** type in the following command and then press **Enter**:

```
dotnet new blazorwasm -o workshop
```

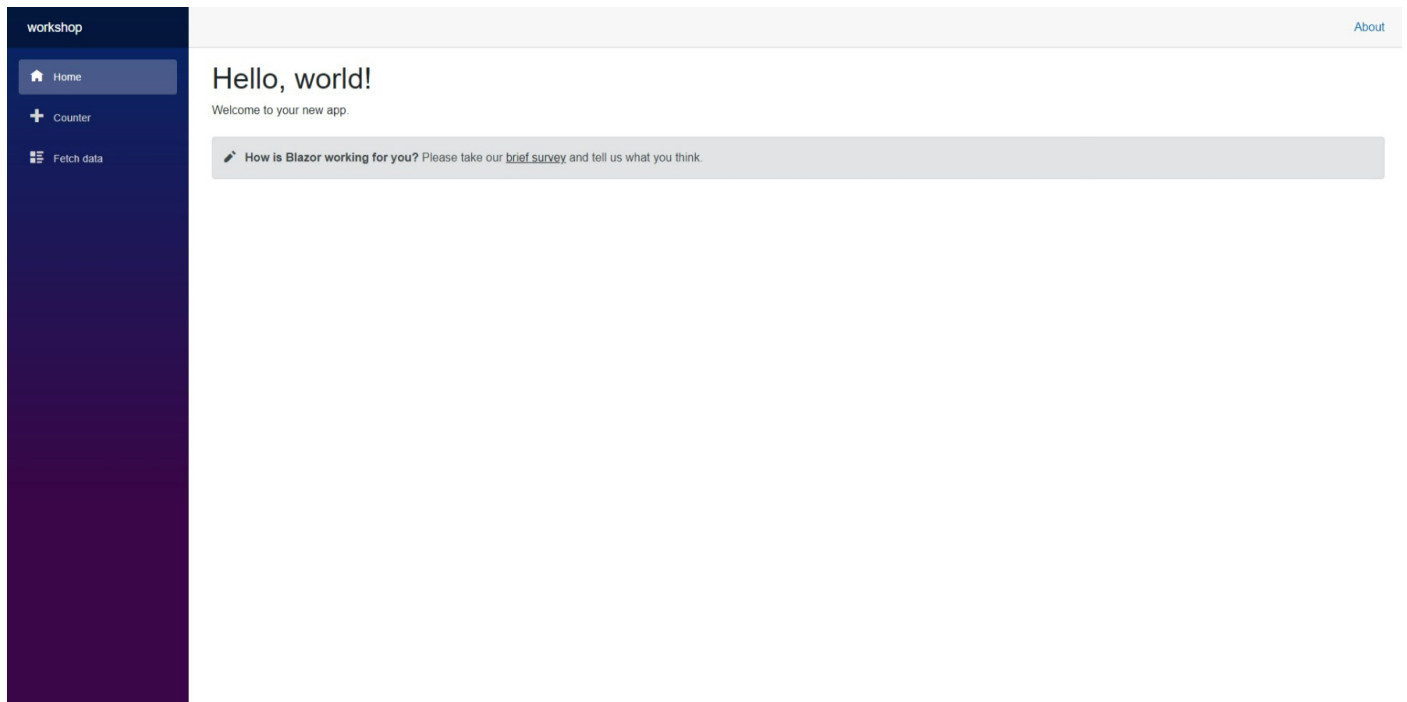
This will create a new **Blazor** for **WebAssembly** or **wasm** Application, once done in the **Command Prompt** you will need to change to the **Folder** for the **Workshop** by typing in the following and then press **Enter**:

```
cd workshop
```

Once done, while still in the **Command Prompt** type in the following command and then press **Enter**.

```
dotnet watch
```

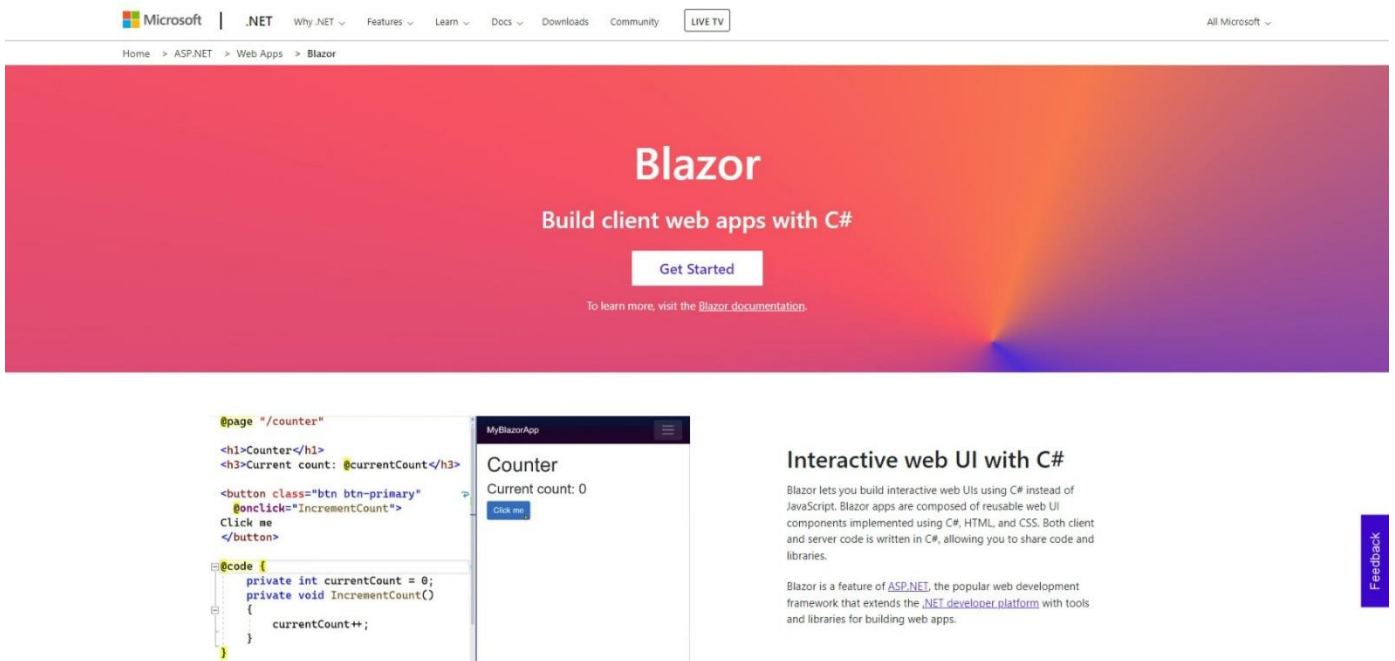
This will **Build** and **Start** the Application and display it in your **Browser**



Keep the **Command Prompt** and **Browser** open during the **Workshop** but can **Close** it if finished. If **Closed** and you need to continue the **Workshop**, just go to **Start** then find **Command Prompt** then go to the **Folder** for the **Workshop** e.g. `C:\Blazor\workshop` and to then **Build** and **Start** the Application by typing the following commands, after each press **Enter**:

```
cd C:\Blazor\workshop
dotnet watch
```

Should you need to, you can get information, documentation and more about **Blazor** at blazor.net



The image shows the Blazor landing page with the heading "Blazor" and the subheading "Build client web apps with C#". A "Get Started" button is visible. Below the heading, there is a code preview for a Blazor application. The code is as follows:

```
@page "/counter"
<h1>Counter</h1>
<h3>Current count: @currentCount</h3>
<button class="btn btn-primary"
    @onclick="IncrementCount">
    Click me
</button>

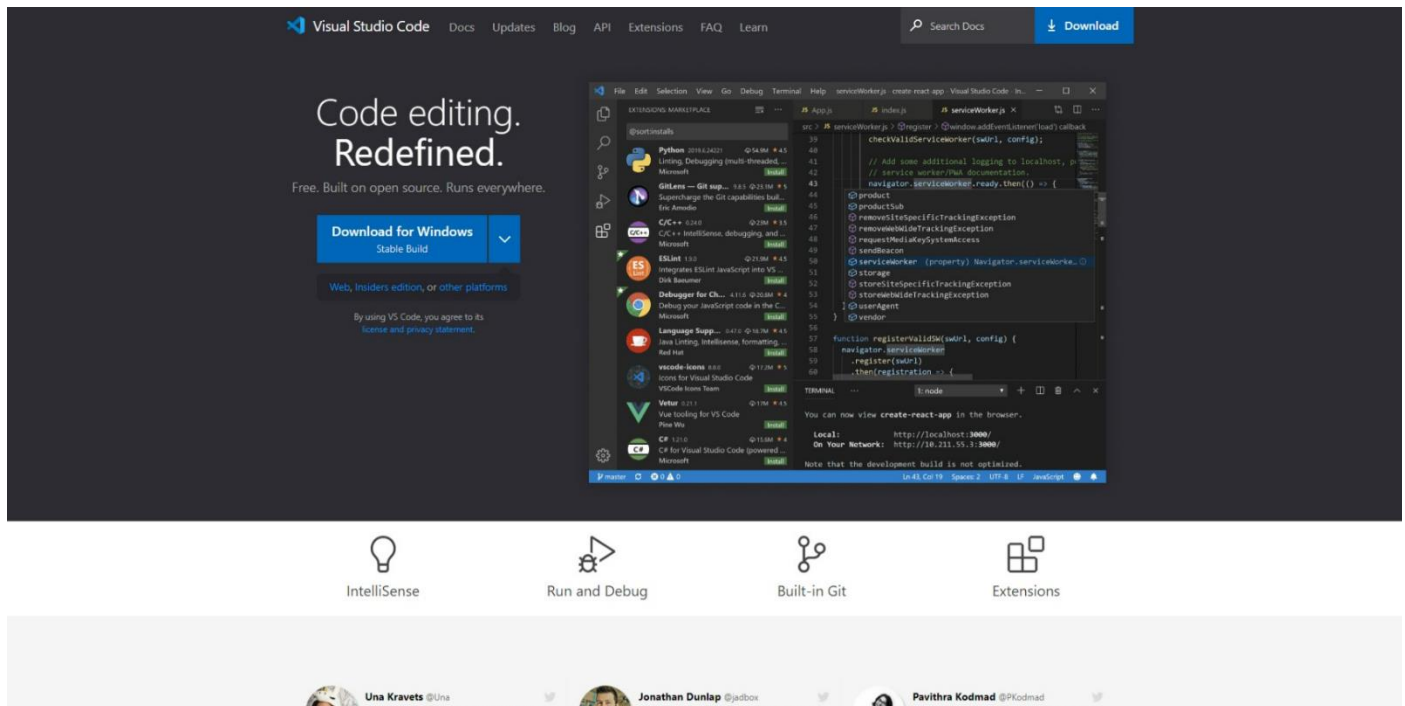
@code {
    private int currentCount = 0;
    private void IncrementCount()
    {
        currentCount++;
    }
}
```

The preview also shows a visual representation of the application with a "Counter" heading and a "Current count: 0" display, along with a "Click me" button. To the right of the code preview, there is a section titled "Interactive web UI with C#" which explains that Blazor lets you build interactive web UIs using C# instead of JavaScript. It also mentions that Blazor is a feature of ASP.NET, the popular web development framework that extends the .NET developer platform with tools and libraries for building web apps.

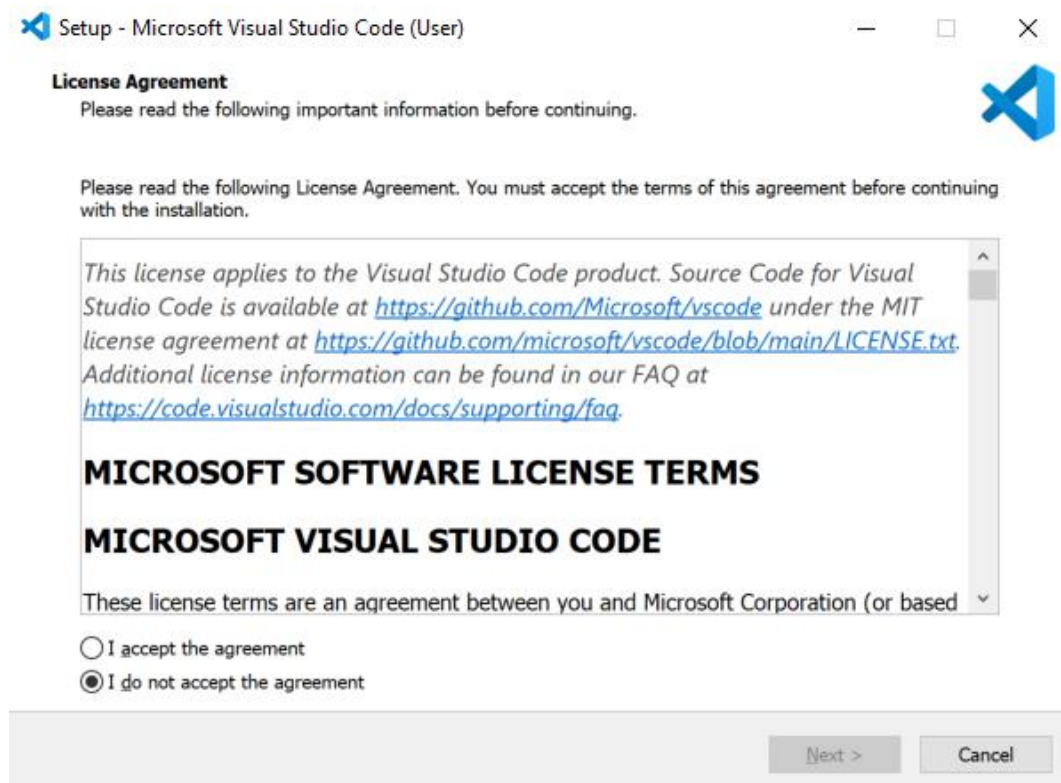
This **Workshop** supports at least **.NET 6** and **C# 10** throughout for **Blazor**.

Visual Studio Code

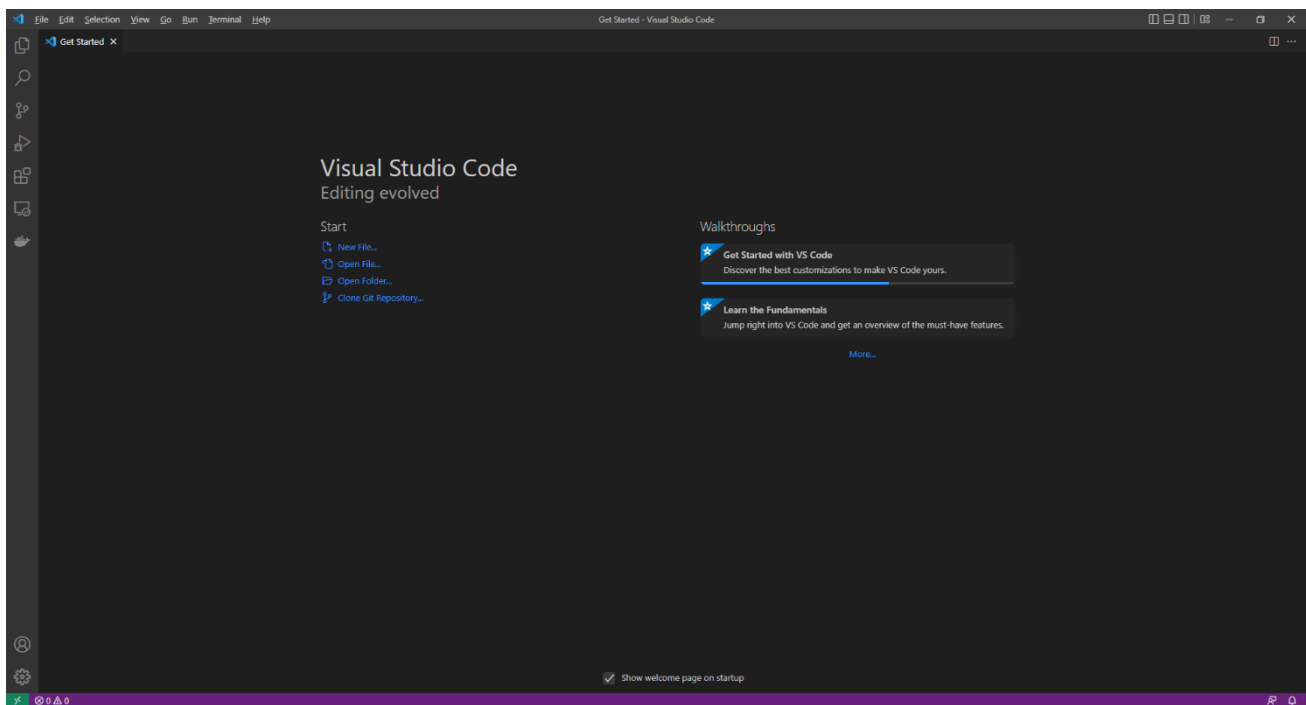
To be able to **Edit** your Application you will need to **Download**, if you don't have it already, **Visual Studio Code** for your Platform such as **Windows** from code.visualstudio.com.



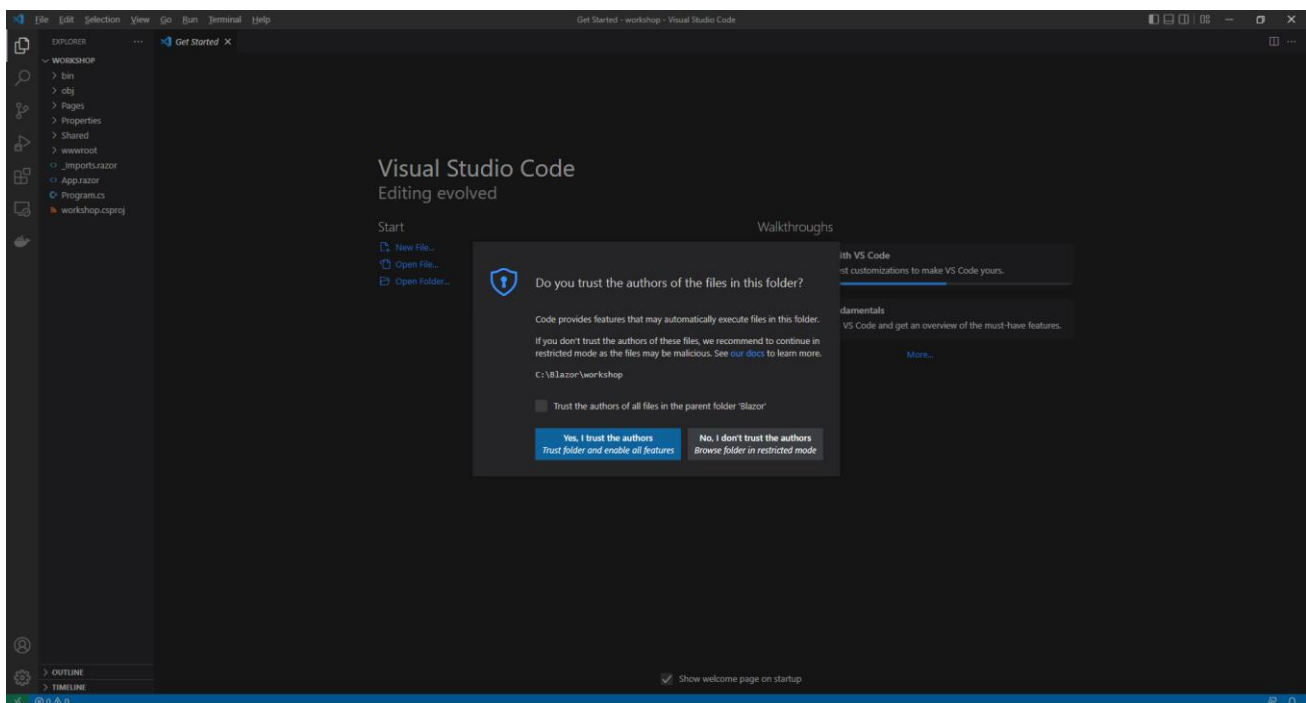
Once **Downloaded**, you can then **Install** it by following the steps in the **Installation Wizard**



Once **Visual Studio Code** has been **Installed**, or if it was already **Installed**, then if using **Windows** you need to go to **Start** then search for **Visual Studio Code** and then select it.

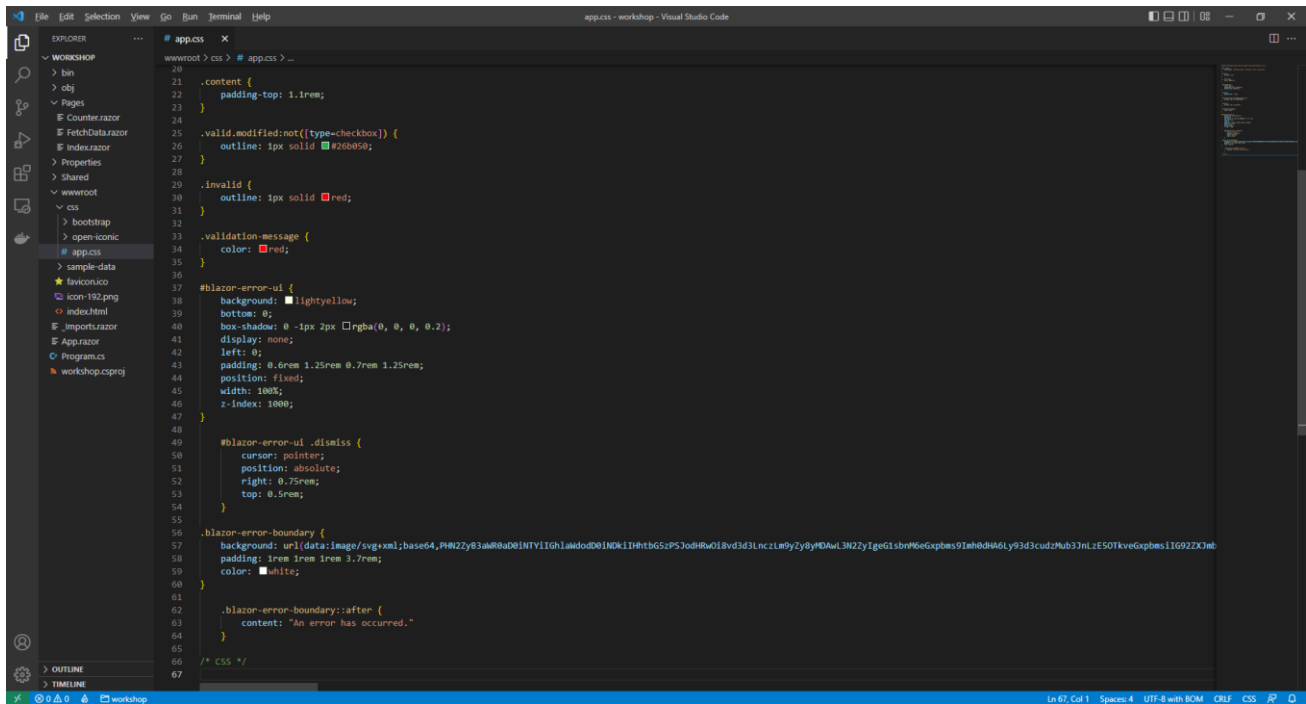


Once **Visual Studio Code** has opened from the **Menu** choose **File** then **Open Folder...** then select the **Folder** for your Application e.g. `C:\Blazor\workshop`. Then once the **Folder** has been opened Select the **Yes, I trust the authors** option in the **Do you trust the authors of the files in this folder?** if this is displayed.

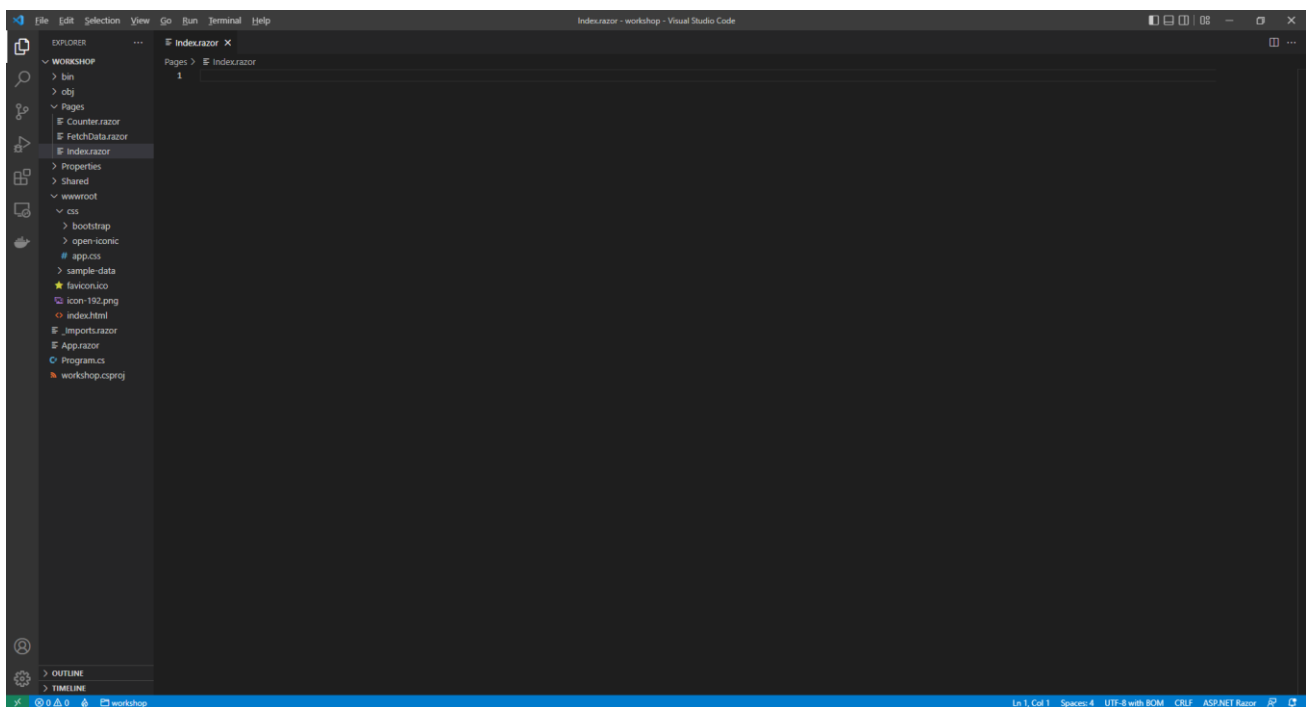


Within **Visual Studio Code** is **Explorer**, expand the **Folder** for **wwwroot** and then **css** to find *app.css*, this defines **CSS** styles for the Application, then at the bottom of *app.css*, type in the following **Comment**:

```
/* CSS */
```



Also in **Visual Studio Code** within the **Explorer** you can Expand the **Folder** for **Pages** to find the main **Component** for the Application which is *Index.razor*, this is where you will be spending most of your time in the **Workshop**, you need to clear the contents of this file so that it is Blank, like as follows:



Then while still in the **Component** of *Index.razor* in **Visual Studio Code** type in the following:

```
@page "/"
@* Injects *@

@code
{
    // Variables

    // Methods

}

@* Application *@
```

Then in **Visual Studio Code** from the **Menu** select **File** then **Save** to save these **Changes** to *Index.razor*. You should always do this when you make any **Changes** to *Index.razor* and other files.

Switch over to the **Command Prompt** that should be still open for **dotnet watch** and there should be the following message, **Do you want to restart your app - Yes (y) / No (n) / Always (a) / Never (v)?** Press **a** to select the **Always** Option and should you **Close** the **Workshop** then you will need to do this again in the **Command Prompt** after getting everything running again and making your first **Changes** after doing so.

Then back in **Visual Studio Code** for the **Component** for the Application of *Index.razor* when you need to use an **@inject** in the **Component** for the Application of *Index.razor* then these should be placed below the **Comment** using the **Razor** Syntax **@*** and ***@** of **@* Injects *@** when needed in the **Workshop**.

To add or declare a **Variable** in the **Component** for the Application of *Index.razor* these should be placed on their own line within the **Block** for **@code** below the **Comment** using the **C#** Syntax **//** of **// Variables** or placed on their own line below any previously declared **Variable**.

To add a **Method** in the **Component** for the Application of *Index.razor* such as a **Function** that has a **return** for a **Value** or one that is **void** and performs some functionality without **Returning** a **Value**, both will contain **{** for the start of the **Body** of the **Method** and **}** for the end of the **Body** of the **Method**, then these should be placed the **Block** for **@code** below the **C#** Syntax **Comment** of **// Methods** or placed below the end of the **Body** of the **Method** of any previously declared **Methods**.

Finally anything else to be added using a combination of **HTML** and **Razor** Syntax in the **Component** for the Application of *Index.razor* then this should be typed in below the **Razor** Syntax **Comment** of **@* Application *@** or below any previously entered **HTML** and **Razor**.

You can use the same **Blazor** Application throughout the **Workshop** and you don't need to remove anything else.

Components

Components make up **Blazor** Applications using **Razor Components** also known as **Blazor Components**. **Components** are self-contained, they define part of a **User Interface** and with **Logic** to define and enable **Dynamic Behaviour**. **Components** are a combination of **C#**, **HTML** and **Razor** Syntax.

Follow **Setup and Start** and you should have the **Command Prompt** for **dotnet watch** open, your **Browser** should also be open and you should also have **Visual Studio Code** open with the **Folder** for the **Workshop** open e.g. *C:\BLazor\workshop*.

Then in **Visual Studio Code** select within **Explorer** in the **Folder** of **Pages** the **Component** for the Application of *Index.razor* and below the **Comment** of *// Variables* type in the following **Variable**:

```
string message = "Hello World";
```

This **Variable** is a **string** which contains some text, in this case **Hello World** when using a **string** it needs to be surrounded by a pair of double quotes " and ". Then to use this **Variable** within a **H1** Tag in **HTML**, you can do so by putting **@** in front of the **Variable** by typing below the **Comment** of **@* Application *** the following:

```
<h1>@message</h1>
```

Then in **Visual Studio Code** from the **Menu** select **File** then **Save** to save these Changes to *Index.razor*. Then switch to the **Browser** that was opened with **dotnet watch** from the **Command Prompt** and you should see the text **Hello World** displayed in a **h1** Tag.

Routing

Routing in **Blazor** allows you to provide a **Route** to a **Component** using the **Directive** of **@page** allowing the **Component** to be accessed using a Relative **URL** or **Website** Address in the **Browser**.

Return to **Visual Studio Code** and from the **Explorer** you need to **Right-Click** on the **Folder** for **Pages** and select the **New File** option then type in the following and then press **Enter**:

```
Message.razor
```

Message.razor will form the basis of a new **Component** so while still in *Message.razor* type in the following:

```
@page "/message/{Value}"
<h2>@Value</h2>
@code
{
    [Parameter]
    public string Value { get; set; } = string.Empty;
}
```

Then in **Visual Studio Code** from the **Menu** select **File** then **Save** to save these Changes to *Message.razor*.

This **Component** for **Message** uses the **Directive** of **@page** with **/message/{Value}**, which will allow the **Component** to be accessed using a Relative **URL** or **Website** Address in the **Browser** along with being able to specify the **Value**. Then there a **h2** Tag being used to display a **Property** of **Value**. This **Property** is C# Syntax allowing the **Value** which is a **string** to be Read, denoted with **get** and to be Written, denoted with **set**. This **Property** is assigned to, using **=** to the **string.Empty** which would be **""** and this **Property** is also used with the **Attribute** of **Parameter** which indicates that this **Property** can be set from another **Component**.

To use this **Component**, from **Visual Studio Code** within **Explorer** from the **Folder** of **Pages** select the **Index.razor**, then in the section with the **Comment** of **@* Application *@** and below **<h1>@message</h1>** type in the following:

```
<Message Value="Hello Again!"/>
```

Then in **Visual Studio Code** from the **Menu** select **File** then **Save** to save these Changes to **Index.razor**. Then switch to the **Browser** that was opened with **dotnet watch** from the **Command Prompt** and you should see the text *Hello Again!* displayed in a **h2** Tag, then while still in the **Browser** in the **Address Bar** type the following Relative **URL** after whatever is there e.g. *https://localhost:7095* (your number may be different):

```
/message/Hello!
```

In the **Browser** the **Address Bar** will be something like *https://localhost:7095/message/Hello!* and you should see the Text *Hello!* displayed in the **Browser** you can also change the *Hello!* passed in to anything you like. When done use the **Home** option from list of options shown below **Workshop** to return to the **Page** showing *Hello World* and *Hello Again!*

Markup

Components can also be created to output specific **Markup** using **C#**, **HTML** and **Razor** Syntax. Return to **Visual Studio Code** and from the **Explorer** you need to **Right-Click** on the **Folder** for **Pages** and select the **New File** option then type in the following and then press **Enter**:

```
Date.razor
```

Date.razor will form the basis of a new **Component** so while still in *Date.razor* type in the following:

```
<h3>@Value.ToString("MMMM dd, yyyy")</h3>
@code
{
    [Parameter]
    public DateOnly Value { get; set; }
}
```

This **Component** for **Date** is being used to display a **Property** of **Value** in a **h3** Tag using **Method** of **ToString("MMMM dd, yyyy")** to **Format** the Output with **MMMM** for the **Month**, **dd** for the **Day** and **yyyy** for the **Year**, these are **Date Format Strings**. This **Property** is **C#** Syntax allowing the **Value** which is a **DateOnly** to be Read, denoted with **get** and to be Written, denoted with **set**. This **Property** is also used with the **Attribute** of **Parameter** which indicates that this **Property** can be set from another **Component**.

To use this **Component**, from **Visual Studio Code** within **Explorer** from the **Folder** of **Pages** select *Index.razor* and below the **Comment** of **// Variables** and after any previously declared **Variable**, type in the following **Variable**:

```
DateOnly dateOfBirth = DateOnly.Parse("23-June-1912");
```

DateOnly is a **Type** in **C#** that only stores a **Date** and can use **Method** the **Parse** to get this from a **string** representation of a **Date** such as **"23-June-1912"**. While still in the **Component** for the Application of *Index.razor* in the section with the **Comment** of **@* Application *@** below **<Message Value="Hello Again!"/>** type in the following:

```
<Date Value="dateOfBirth"/>
```

While still in **Visual Studio Code** from the **Menu** select **File** then **Save** to save these Changes to *Index.razor*. If you switch to the **Browser** is open for the **Workshop**, you will see the **Component** for **Date** being displayed as *June 23, 1912*, which is Alan Turing's birthday, a pioneer in the field of computing!

Styles and CSS

Components can also allow a **Value** to be used to define the **CSS Style** of a Tag of **HTML**. In **Visual Studio Code** within **Explorer** from the **Folder** of **Pages** select the *Index.razor* and below the **Comment** of **// Variables** and after any previously declared **Variables**, type in the following **Variable**:

```
string styling = "background-color: yellow";
```

To use the **string** as **CSS Style**, while still in the **Component** for the Application of *Index.razor* type in below `<Date Value="dateOfBirth"/>` the following:

```
<div><span style="@styling">Highlighted</span></div>
```

Then in **Visual Studio Code** from the **Menu** select **File** then **Save** to save these Changes to *Index.razor*. Then switch over to the **Browser** that was opened with **dotnet watch** you will see the Text of *Highlighted* with a Background Colour of *yellow*.

Then, back in **Visual Studio Code** from **Explorer** in the **Folder** of **wwwroot** and **css** select *app.css* and define some **CSS** for the Application by typing in below the **Comment** of **/* CSS */** the following:

```
.inverted {
    color: white;
    background-color: black;
}

.large {
    font-size: 2.0em;
}
```

Then in **Visual Studio Code** from the **Menu** select **File** then **Save** to save these Changes for *app.css*. Return to the **Component** for the Application within the **Folder** for **Pages** of *Index.razor* and below the **Comment** of **// Variables** and after any previously declared **Variables**, type in the following **Variable**:

```
string[] contrast = { "inverted", "large" };
```

The square-brackets of `[` and `]` denote an **Array** which is a set of **Elements** of a given **Type** in this case they are a **string** with the **Values** of **inverted** and **large**. While still in the **Component** of *Index.razor* and below `<div>Highlighted</div>` type in the following:

```
<div><span class="@string.Join(" ", contrast)">Contrast</span></div>
```

This will set the **class** for the **span** using the **Variable** of **contrast** since **CSS** needs to be defined with Spaces inbetween the **Method** for **string.Join** will be used to combine the Values from the **Array** with a Space, denoted with `" "`. If you switch over to the **Browser** that was opened with **dotnet watch** from the **Command Prompt** it will have the Text of *Contrast* in *white* with a *black* Background.

Images

Components can contain any other **HTML** Elements such as an **Image** using the **img** Tag and can set the **src** from anything, including a **Method**, to do this, in **Visual Studio Code** in the **Component** for the **Application** of *Index.razor* within the **Folder** of **Pages** type below the **Comment** for *// Methods* the following **Method**:

```
Uri GetImage()  
{  
    return new("https://openmoji.org/data/color/svg/1F600.svg");  
}
```

This **Method** of **GetImage()** will return a **Uri**, to use this, while still in the **Component** for the Application of *Index.razor* below `<div>Contrast</div>` in the section with the **Comment** of *@* Application *@*, type in the following:

```
<div>  
      
</div>
```

This will set the **src** the **img** Tag using the **Result** from the **Method** of **GetImage()**. Back in the **Browser**, you should see a *Grinning Face* displayed, image courtesy of openmoji.org.

Binding and Events

Binding

Binding in **Blazor** can be used in **Components** to allow for **Data Binding** Elements using the **Directive** of **@bind** with **Values** such as a **Field** or **Variable**, **Property** or an **Expression**.

After following **Setup and Start** and **Components** you should have the **Command Prompt** for **dotnet watch** open, your **Browser** should also be open and you should also have **Visual Studio Code** open with the **Folder** for the **Workshop** open e.g. *C:\Blazor\workshop*.

In **Visual Studio Code** select the **Component** for the Application, *Index.razor* found within **Explorer** in the **Folder** of **Pages** below the **Comment** of *// Variables* and after any previously declared **Variables**, type in the following **Variable**:

```
string text = string.Empty;
```

Then while still in **Visual Studio Code** for the **Component** for the Application of *Index.razor*, within the section containing the **Comment** of *@* Application ** below the final *</div>* type in the following:

```
<div>
    <input type="text" @bind="text" @bind:event="oninput" />
    <h2>@text</h2>
</div>
```

This will use the **Directive** for **@bind** to **Bind** to the **Field** or **Variable** of **text** that was declared, and using the **bind:event** will update **text** when the **input** is typed into as this will trigger the **Event** for **oninput** of the **input** and underneath the **Value** of **input** will be displayed in a **h2** Tag.

Then **Visual Studio Code** from the **Menu** select **File** then **Save** to save the Changes to *Index.razor*.

Switch to the **Browser** opened with **dotnet watch** from the **Command Prompt** you will see an **input**, which when Typed into will have the same contents displayed underneath in a **h2** Tag.

Events

Events in **Blazor** can be used in **Components** to create more **Dynamic Behaviour** in a **Blazor** Application. In **Visual Studio Code** select the **Component** for the Application, *Index.razor* found within **Explorer** in the **Folder** of **Pages** type below the **Comment** of `@* Injects *@` type in the following:

```
@inject IJSRuntime runtime;
```

This will use **inject** to include **IJSRuntime** which is an **Interface** exposing functionality from **JavaScript** to allow the **Blazor** application to **Invoke** features from **JavaScript**. Then below the **Comment** for `// Methods` and after the **Body** of any previous **Methods** type the following **Methods**:

```
async void Alert(string message)
{
    await runtime.InvokeVoidAsync("alert", message);
}

void ShowMessage()
{
    Alert("Hello World");
}
```

The first **Method** is used to **Invoke** a feature from **JavaScript**, in this case for **alert** which is used to display an **alert** Message in a **Browser**. This **Method** uses **async** and **await** which means it will perform a **Task** that won't happen at the same time as anything else, or **Asynchronously** denoted with **async** and when this is completed it will come back after this **Task** has completed which is denoted with **await**. The second **Method** will **Call** the **Method** for **Alert** with the **string** of **Hello World**.

While still in the **Component** of *Index.razor* in the section with the **Comment** of `@* Application *@` below the final `</div>` type in the following:

```
<div>
    <button class="btn btn-primary" @onclick="ShowMessage">
        Show Message
    </button>
</div>
```

Then you can select the **Browser** opened with **dotnet watch** from the **Command Prompt** you will see a **button** labelled *Show Message* which when **Clicked** will display an **alert** displaying the Message of **Hello World**.

You can also use **Events** to perform other **Dynamic Behaviour** such as changing the **Style** of an Element. Return to **Visual Studio Code** and select the **Component** for the Application, *Index.razor* found within **Explorer** in the **Folder** of **Pages** then below the **Comment** of `// Variables` and after any previously declared **Variables**, type in the following **Variables**:

```
string style = string.Empty;
bool isSelected = false;
```

The first **Variable** is a **string** for the **Style** and the second **Variable** is a **bool** which is a **Value** that can be either **true** or **false**. Then below the **Comment** for `// Methods` and after the **Body** of any previous **Methods** type the following **Method**:

```
void SetStyle()
{
    isSelected = !isSelected;
    style = $"font-weight: {(isSelected ? "bold" : "normal")}";
}
```

This **Method** will first set **isSelected** to **isSelected** with the **!** or the **Operator** for **Not**, this works with a **bool** by changing anything that was **true** to be **false** and anything that was **false** to be **true**. Then **style** is set with a string using **String Interpolation** which is denoted with the use of **\$** at the start. There is an **Expression** contained within the brackets of (and) which uses the **Conditional Operators** of **?** and **..**. Should the **Value** of **isSelected** be **true** then **"bold"** will be **Returned** from the **Conditional** or when **isSelected** false then **"normal"** will be **Returned** from the **Conditional**.

Then, while still in the **Component** of *Index.razor* in the section with the **Comment** of `@* Application *@` and below the final `</div>` type in the following:

```
<div>
    <a href="#" style="@style" @onclick="SetStyle">Toggle Style</a>
</div>
```

Then **Visual Studio Code** from the **Menu** select **File** then **Save** to save the Changes to *Index.razor* and then go to the **Browser** that was opened with **dotnet watch** and **Click** the **Link** with the Text of *Toggle Style* and this Text will switch between being *bold* or *normal* when **Clicked**.

It is possible to combine **Binding** with **Events**. Return to **Visual Studio Code** then select the **Component** for the Application, *Index.razor* found within **Explorer** in the **Folder** of **Pages** and below the **Comment** of *// Variables* and after any previously declared **Variables**, type in the following **Variable**:

```
string value = string.Empty;
```

While still in the **Component** of *Index.razor* in the section with the **Comment** of *@* Application ** below the final `</div>` type in the following:

```
<div>
    <input type="text" @bind="value" />
    <button class="btn btn-primary" type="button"
        @onclick="@{e => Alert(value)}">
        Show
    </button>
</div>
```

This will use `@bind` to **Bind** the **input** to the **Variable** of **value**, then the **button** will use the **Event** of `@onclick` to **Call** the **Method** of **Alert** with **value**.

In **Visual Studio Code** from the **Menu** select **File** then **Save** to save the Changes to *Index.razor* and then you can select the **Browser** opened with **dotnet watch** and you should see an **input** with a **button** labelled *Show*, anything you type in the **input** will be displayed in an **alert** when the **button** of *Show* is **Clicked**.

You can define **Events** in a **Component**. In **Visual Studio Code** and then from **Explorer** you need to **Right-Click** on the **Folder** for **Pages** and select **New File** then type in the following and then press **Enter**.

```
Sizer.razor
```

This will form the basis of a new **Component**, within the **Component** of *Sizer.razor* type in the following:

```
<div>
    <button class="btn btn-primary"
        @onclick="Decrease" title="Decrease">-</button>
    <button class="btn btn-primary"
        @onclick="Increase" title="Increase">+</button>
    <span style="font-size:@(Size)px">Font Size: @Size<text>px</text></span>
</div>
@code
{
    [Parameter]
    public int Size { get; set; }

    [Parameter]
    public EventCallback<int> SizeChanged { get; set; }

    private async void Resize(int delta)
    {
        Size = Math.Min(40, Math.Max(8, + this.Size + delta));
        await SizeChanged.InvokeAsync(Size);
    }

    private void Decrease() =>
        Resize(-1);

    private void Increase() =>
        Resize(+1);
}
```

Starting with the **Block** for **@code** there is a **Parameter** for the **Component** of **Size** which is an **int** which represents a whole number such as 20, there is also another **Parameter** for the **Component** for an **EventCallback**, this uses the **Generic** Syntax denoted by the angled-brackets or chevrons of < and > and in this case this is the **Type** of **int** which will be used for the latest value of **Size**. Both of these use the **Keyword** in **C#** for **public**, this means they are accessible outside of the **Component**.

Then there are **Methods**, marked with **private** which indicates they should only be used or be accessible from within the **Component**, starting with **Resize** which is marked **async** which means it will perform a **Task** that won't happen at the same time as anything else or **Asynchronously**. The **Method** performs a **Calculation** using the **delta** value passed in as a **Parameter** for the **Method**, once the **Calculation** is done, then the **Method** for **InvokeAsync** is invoked which will be used to get the latest **Value** to use. The last two **Methods** invoke **Resize** and either pass in **-1** or **+1** accordingly.

Then above the **Block** for **@code** there is a **button** when **Clicked** or **@onclick** will either *Decrease* with - or *Increase* with + and then there is a **span** which will display the **Value** of **Size** and use this with the **Style** of **font-size** with it too.

To use this **Component**, from **Visual Studio Code** within **Explorer** from the **Folder** of **Pages** select *Index.razor*, then below the **Comment** of `// Variables` and after any previously declared **Variables**, type in the following **Variable**:

```
int size = 20;
```

This **Variable** is an **int** and has been set to **20** for the initial **Value** of *size*. While still in *Index.razor*, in the section with the **Comment** of `@* Application *@` below the final `</div>`, type in the following:

```
<div>
    <Sizer Size="@size" SizeChanged="(int value) => size = value" />
    <div style="font-size:@(size)px">Resizable Text</div>
</div>
```

This will include the **Component** of **Sizer** and then will set the **Value** for **Size** with the **Variable** of *size*, along with creating an **Expression** to use with **SizeChanged** which will update the **Variable** of *size* and there is also a **div** where the **Style** for the **font-size** will be used too.

Then **Visual Studio Code** from the **Menu** select **File** then **Save** to save the Changes to *Index.razor*.

Then when you switch over to the **Browser** you can use the *Sizer* to change the *font-size* of itself and the **div** contained the Text of *Resizable Text* too.

Conditions and Collections

Conditions can be used in **Blazor** to control what is being displayed to add even more **Dynamic Behaviour** to an Application.

After following **Setup and Start, Components** and **Binding and Events**. Then you should have the **Command Prompt** for **dotnet watch** open, your **Browser** should also be open and you should also have **Visual Studio Code** open with the **Folder** for the **Workshop** open e.g. *C:\Blazor\workshop*.

In **Visual Studio Code** select the **Component** for the Application, *Index.razor* found within **Explorer** in the **Folder** of **Pages** below the **Comment** of *// Variables* and after any previously declared **Variables**, type in the following **Variable** for a **bool**:

```
bool isShown = false;
```

Then in **Visual Studio Code** for *Index.razor*, below the **Comment** for *// Methods* and after the **Body** of any previous **Methods** type the following **Method**:

```
void Toggle()
{
    this.isShown = !this.isShown;
}
```

The **Method** of **Toggle()** will set **isShown** to **isShown** with the **!** or the **Operator** for **Not**, this works with a **bool** by changing anything that was **true** to be **false** and anything that was **false** to be **true**.

Then while still in *Index.razor*, in the section with the **Comment** of *@* Application *@* below the final *</div>*, type in the following:

```
<div>
    <button class="btn btn-primary"
        @onclick="Toggle">Click Here</button>
    @if (isShown)
    {
        <h2>Hello World!</h2>
    }
</div>
```

This contains a **button** that when **Clicked** or **@onclick** will invoke the **Method** for **Toggle** which updates **isShown**. Below this is a **Conditional Statement** of **if** which the contents of which, containing a **h2** Tag with the Text of *Hello World!* but this will only be displayed when **isShown** is **true**.

If you switch to the **Browser** that was opened with **dotnet watch** from the **Command Prompt**, you will see the **button** with the Text of *Click Here*, which when **Clicked** will either display underneath the Text of *Hello World!* in a **h2** Tag or hide it when **Clicked** if it is already there.

Components can also display the contents of a **Collection**. Return to **Visual Studio Code** and select the **Component** for the Application, *Index.razor* found within **Explorer** in the **Folder** of **Pages** then below the **Comment** of `// Variables` and after any previously declared **Variables**, type in the following **Variable**:

```
List<string> items = new()
{
    "Hello",
    "World"
};
```

The **Collection** used here is a **List** which uses the **Generic** Syntax denoted by the angled-brackets or chevrons of `<` and `>` and in this case this **List** has Elements with a **Type** of **string** with the **Values** being the **string** of "Hello" and "World".

While still in the **Component** for the Application of *Index.razor*, in the section with the **Comment** of `@* Application *` and below the final `</div>`, type in the following:

```
<div>
  <ul>
    @foreach (string item in items)
    {
      <li>@item</li>
    }
  </ul>
</div>
```

Within this is a **ul** for an **Unordered List** or **Bulleted List** then uses **foreach** to go through each **Value** in the **Collection** as a **string** and output them using a **li** or **List Item**.

In **Visual Studio Code** from the **Menu** select **File** then **Save** to save the Changes to *Index.razor*, then switch over to the **Browser** that was opened, there will be a **Bulleted List** showing the **List Items** of *Hello* and *World*.

You can combine **Conditions** and **Collections** using a **switch** to display values from a **Variable**. Return to **Visual Studio Code** and select the **Component** for the Application, *Index.razor* found within **Explorer** in the **Folder** of **Pages** then below the **Comment** of `// Variables` and after any previously declared **Variables**, type in the following **Variable**:

```
Dictionary<string, string> values = new()
{
    { "None", "" },
    { "Danger", "red" },
    { "Warning", "yellow" },
    { "Proceed", "green" }
};
```

The **Collection** used here is a **Dictionary** which also uses the **Generic** Syntax denoted by the angled-brackets or chevrons of `<` and `>` and in this case this **Dictionary** has Elements with the **Key** and the **Value** the **Type** of **string** and has some **Values** set.

While still in the **Component** for the Application of *Index.razor*, in the section with the **Comment** of **@* Application** and below the final `</div>`, type in the following:

```
<div>
  <ul>
    @foreach (var value in values)
    {
      <li>
        @switch (value.Key)
        {
          case "Danger":
            <span style="background-color: red">
              Danger
            </span>
            break;
          case "Warning":
            <span style="background-color: yellow">
              Warning
            </span>
            break;
          case "Proceed":
            <span style="background-color: green">
              Proceed
            </span>
            break;
          default:
            <span>
              None
            </span>
            break;
        }
      </li>
    }
  </ul>
</div>
```

Within this is a **ul** for an **Unordered List** or **Bulleted List** then uses **foreach** to go through each **Value** in the **Collection** as a **var** which means the **Type** can be **Inferred** rather than **Explicitly** stated. Then there is a **switch** which will use the **Key** from the **Dictionary** of **Values** and output using a **li** or **List Item** for a given **case** of the **Key** with **default** being used for any other **Value**.

In **Visual Studio Code** from the **Menu** select **File** then **Save** to save the Changes to *Index.razor*. Then if you switch over to the **Browser** there will be another **Bulleted List** showing the List Items of *None*, then *Danger* with a *red* Background, *Warning* with a *yellow* Background and *Proceed* with a *green* Background.

Forms

Forms in **Blazor** take advantage of built-in **Components** for **Input** and you can **Bind** to a **Model** using **Data Annotations** that allow you to **Validate** any **Input** from **Forms**.

Follow **Setup and Start**, **Components**, **Binding and Events** and **Conditions and Collections**. You should then have the **Command Prompt** for dotnet watch open, your **Browser** should also be open and you should also have **Visual Studio Code** open with the **Folder** for the **Workshop** open e.g. `C:\Blazor\workshop`.

In **Visual Studio Code** from the **Menu** select **File** and then **New Text File** then select **File** again and this time choose the **Save** option, select the **Folder** for the **Workshop** e.g. `C:\Blazor\workshop`, then set the **Filename** to the following:

```
Model.cs
```

Then either press **Enter** or select **Save** once done, in **Visual Studio Code** from the **Explorer** within the **Workshop** select `Model.cs` and then type in the following:

```
using System.ComponentModel.DataAnnotations;

public class Model
{
    [Required]
    public string? Name { get; set; }
}
```

Then in **Visual Studio Code** from the **Menu** select **File** then **Save** to save the Changes for `Model.cs`, which is a **class** which in **C#** is a pattern of a particular object. In this case it will be for **Name** which is a **Property** with the **Type** of a **string?** the **?** denotes that this **Property** can be **null** which is a special **Value** that means it has no value. The **Property** also has an **Attribute** that is part of the **Data Annotations** which is brought in with **using** of `System.ComponentModel.DataAnnotations` to denote that this **Property** is **Required** so it must have a **Value** when used with the **Form**.

Then in **Visual Studio Code** in the **Explorer** within the **Folder** of **Pages** in the **Component** for the Application of `Index.razor` below the **Comment** of `// Variables` and after any previously declared **Variables** type in the following **Variable**:

```
Model model = new();
```

This will create an **Instance** of the **Class** for the `Model` so it can be used in the **Form**.

While still in **Visual Studio Code** for *Index.razor*, below the **Comment** for `// Methods` and after the **Body** of any previous **Methods** type the following **Method**:

```
void HandleValidSubmit()
{
    Alert(model.Name ?? string.Empty);
}
```

This **Method** will use the **Method** for **Alert** to display a message containing the **Value** of **Name** from the **Instance** of **Class** for **Model** of **model**. This also features the `??` or **Null-coalescing Operator** which when the value is **null** it will use **Value** for **string.Empty** or `""` instead.

Then while still in **Visual Studio Code** in the **Component** for the Application of *Index.razor*, in the section with the **Comment** of `@* Application *@` and below the final `</div>`, type in the following:

```
<div>
    <EditForm Model="@model" OnValidSubmit="@HandleValidSubmit">
        <DataAnnotationsValidator />
        <label>
            Name
            <InputText id="name" @bind-Value="model.Name" />
        </label>
        <ValidationSummary />
        <button class="btn btn-primary" type="submit">Submit</button>
    </EditForm>
</div>
```

This uses the built-in **Component** of **EditForm** with the **Model** set to the **Instance** of **model** for the **Class** it uses the **Method** of **HandleValidSubmit** when the **Form** is **Valid**, this is based on the **DataAnnotationsValidator** which will check if the **Property** of **Name** is not **null** and has a **Value**. The **Component** of **InputText** and is **Bound** to the **Property** from the **Model** of **Name**. If there is no **Valid** input then any issues will be displayed using the **ValidationSummary** then finally there is the **button** with the **type** of **submit** to process the **Form** with the Text of *Submit*.

In **Visual Studio Code** from the **Menu** select **File** then **Save** to save the Changes to *Index.razor*. Then, when you switch to the **Browser** that was opened with **dotnet watch** from the **Command Prompt**, there will be an **input** next to the **button** with the Text of *Submit* if you type in anything then **Click** on *Submit* then it will display an **alert** with what was typed in, if nothing is entered then it will state that *The Name field is required.* along with showing the **input** in red.

Dependency Injection

Dependency Injection in **Blazor** allows **Services** to be **Injected** into an Application, these **Services** can be either **Framework-registered** or **Custom**. **Dependency Injection** is a software design pattern allowing functionality to be provided to a given part of an **Application** to achieve **Inversion of Control** where **Implementation** is written to depend on or **Implement** higher-level abstractions to allow for more modular and maintainable code.

Follow **Setup and Start**, **Components**, **Binding and Events**, **Conditions and Collections** and **Forms**. You should then have the **Command Prompt** for `dotnet watch` open, your **Browser** should also be open and you should also have **Visual Studio Code** open with the **Folder** for the **Workshop** open e.g. `C:\Blazor\workshop`.

In **Visual Studio Code** from the **Menu** select **File** and then **New Text File** then select **File** again and this time choose the **Save** option, select the **Folder** for the **Workshop** e.g. `C:\Blazor\workshop`, then set the **Filename** to the following:

```
DemoService.cs
```

Then either press **Enter** or select **Save** once done, in **Visual Studio Code** from the **Explorer** within the **Workshop** select `DemoService.cs` and then type in the following **class**:

```
public class DemoService
{
    public string GetMessage()
    {
        return "Hello Demo!";
    }
}
```

This **class** represents a simple **Service** that has a single **Method** of `GetMessage()` which will **return** the **string** of "Hello Demo!".

Then in **Visual Studio Code** from the **Explorer** for **Workshop** you should find `program.cs`. In this, below the line of `builder.Services.AddScoped(sp => new HttpClient { BaseAddress = new Uri(builder.HostEnvironment.BaseAddress) });` type in the following:

```
builder.Services.AddScoped(sp => new DemoService());
```

This line will **Register** the **Service** with the Application. Then in **Visual Studio Code** from the **Menu** select **File** then **Save** to save the Changes to `program.cs`.

Then while still in **Visual Studio Code** select the **Component** for the Application, *Index.razor* found within **Explorer** in the **Folder** of **Pages** and type below the **Comment** of `@* Injects *` and after any previous `@inject` type in the following:

```
@inject DemoService service;
```

Then while still in the **Component** for the Application of *Index.razor*, within the section with the **Comment** of `@* Application *` and below the final `</div>`, type in the following:

```
<h2>@service.GetMessage()</h2>
```

This will get the **Value** for a **h2** Tag using the **Method** of `GetMessage` from the **Service** that was **Injected** using **Dependency Injection**.

Then switch to the **Browser** that was opened with `dotnet watch` from the **Command Prompt** and you should see the text *Hello Demo!* displayed in a **h2** Tag and that concludes this **Workshop** about **Blazor** from tutorialr.com!