workshop

Blazorfy

Tech on the Tyne

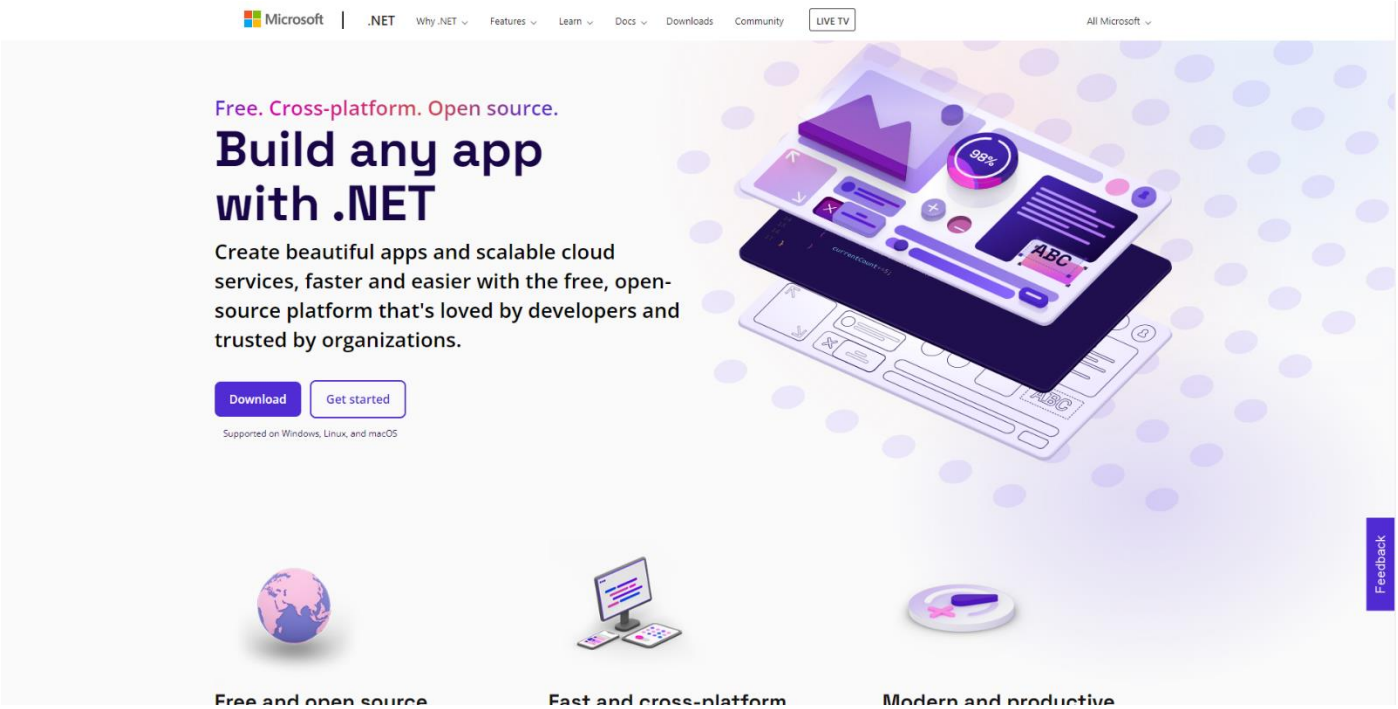
Contents

Contents

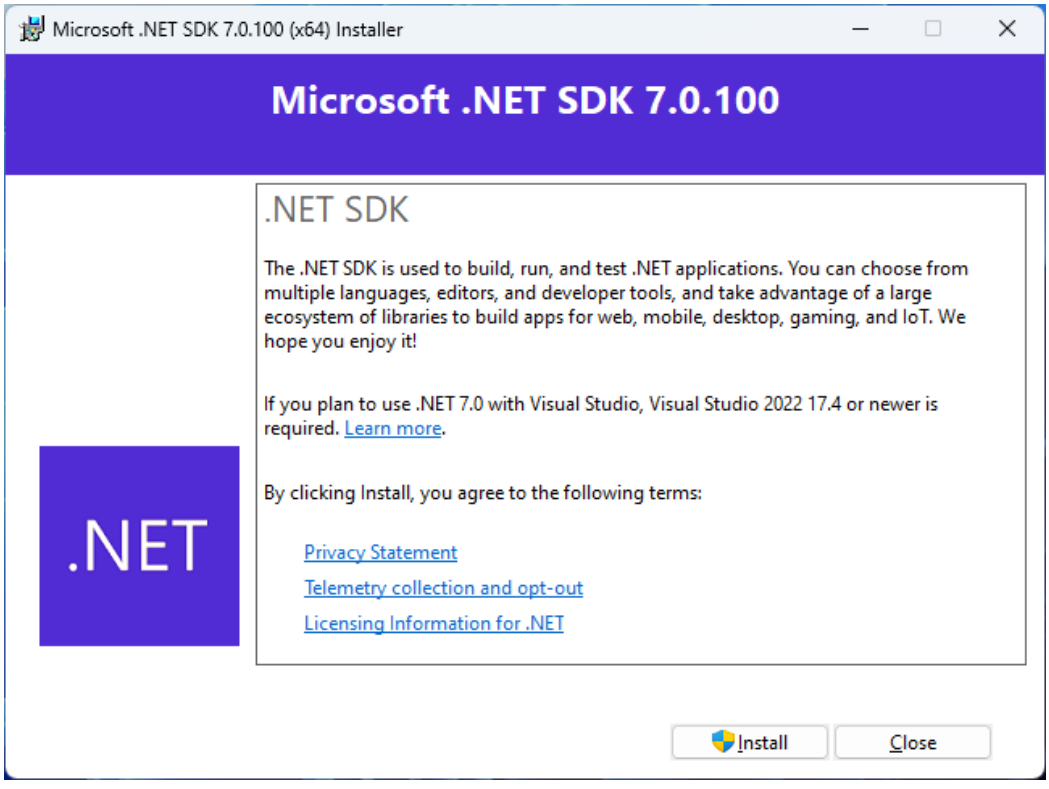
Setup

.NET

.NET includes **Blazor** so you will need to **Download** and **Install** the latest version of the **.NET SDK**, which if you don't have it already you can **Download** it for **Windows** or **Mac** using a new **Browser** tab at [dot.net](dot.net)

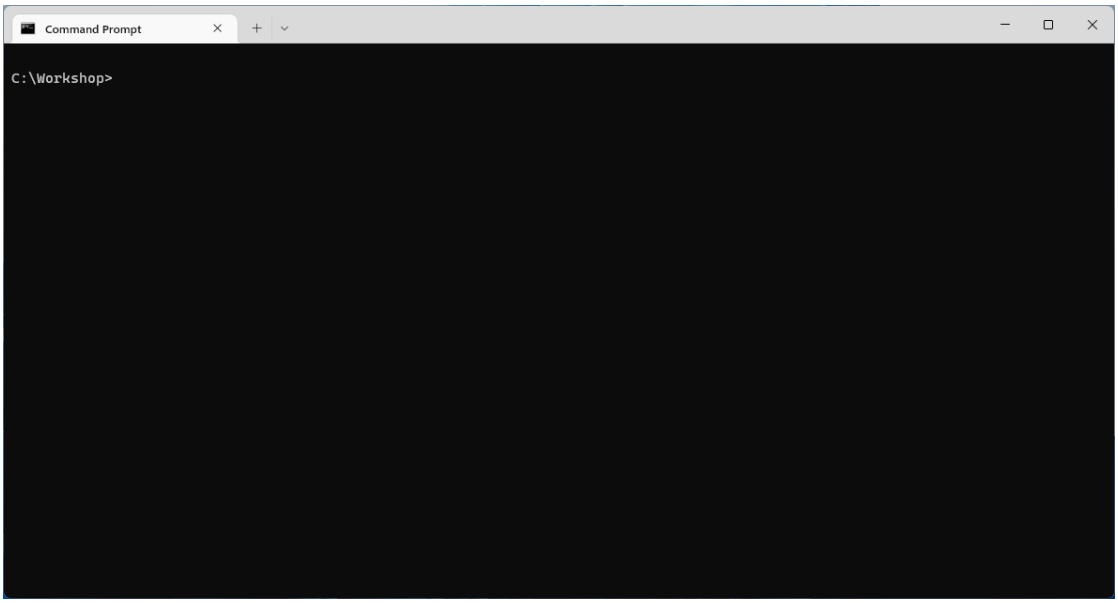


Once **Downloaded** you can open to **Install** the **.NET SDK** by following the steps in the **Installation Wizard**



Project

If the **.NET SDK** has been **Installed**, then if using a **Mac** you then need to go to **Finder** then search for **Terminal** and then select it or if using **Windows** you need to go to **Start** then search for **Command Prompt** and then select it so it launches as follows:



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

```
mkdir Workshop
```

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

```
cd Workshop
```

Once in this **Folder** you can create a new **Project** using the **.NET CLI** that was **Installed** as part of the **.NET SDK**. While still in the **Command Prompt** or **Terminal** type in the following and then press **Enter**:

```
dotnet new blazorwasm -o Blazorfy
```

This will create a new **Project** for **Blazor** using **WebAssembly** or **wasm**. Once the **Project** has been created in the **Command Prompt** or **Terminal** you will need to change to the **Folder** for the **Workshop** by typing in the following and then press **Enter**:

```
cd Blazorfy
```

Please make a note of the **Folder** where you have created the **Project** e.g. *C:\Workshop\Blazor* for later in the **Workshop**.

Packages

While still in the **Command Prompt** or **Terminal** you will add some **Packages** that will be used in **Blazorfy** to add the first **Package** of *Blazored.LocalStorage*, type the following and then press **Enter**:

```
dotnet add package Blazored.LocalStorage
```

This will add the **Package** for *Blazored.LocalStorage* created by *Chris Sainty* which provides access to local storage for **Blazor** applications, this will be used to save and load values in the **Browser**.

Then while still in the **Command Prompt** or **Terminal** you can add the second **Package** of *Spotify.NetStandard* type the following and then press **Enter**:

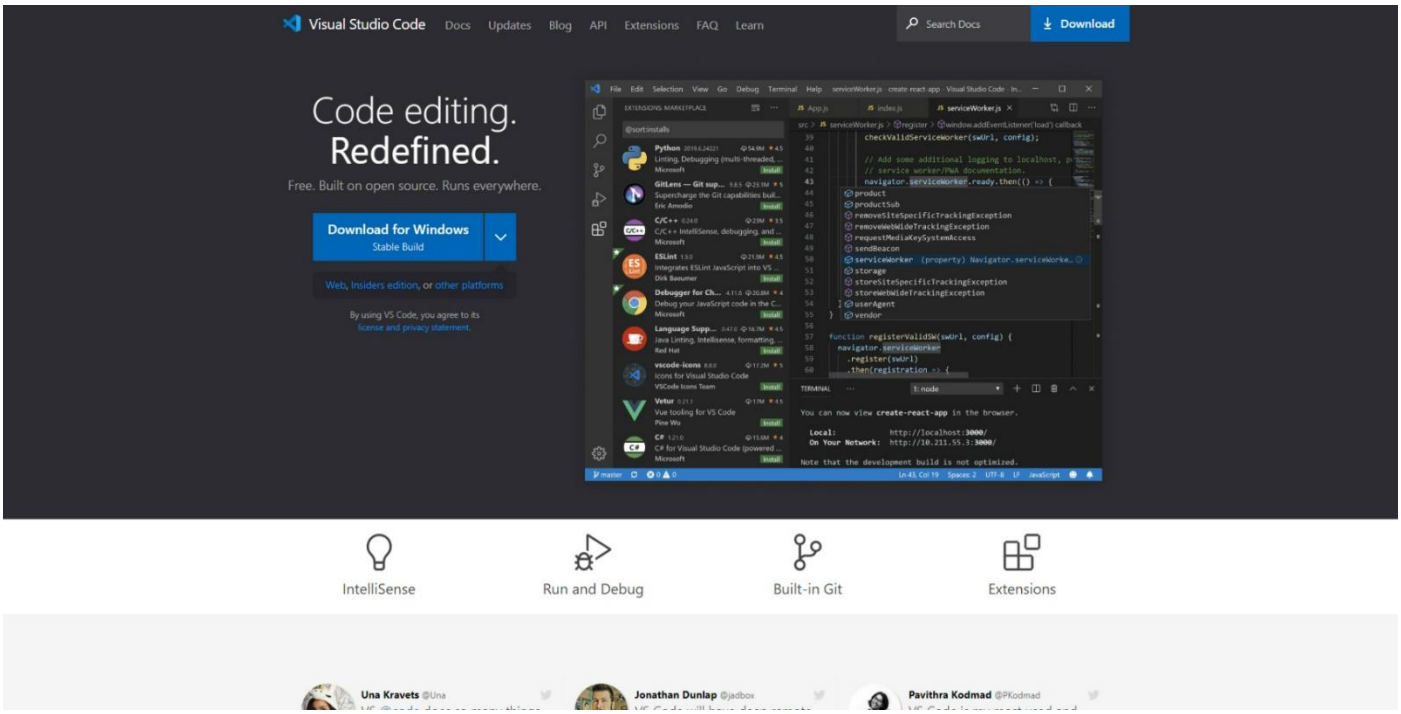
```
dotnet add package Spotify.NetStandard
```

This will add the **Package** for *Spotify.NetStandard* created by *Peter Bull* which provides access to the **Spotify Web API** and will be used to obtain information from **Spotify**.

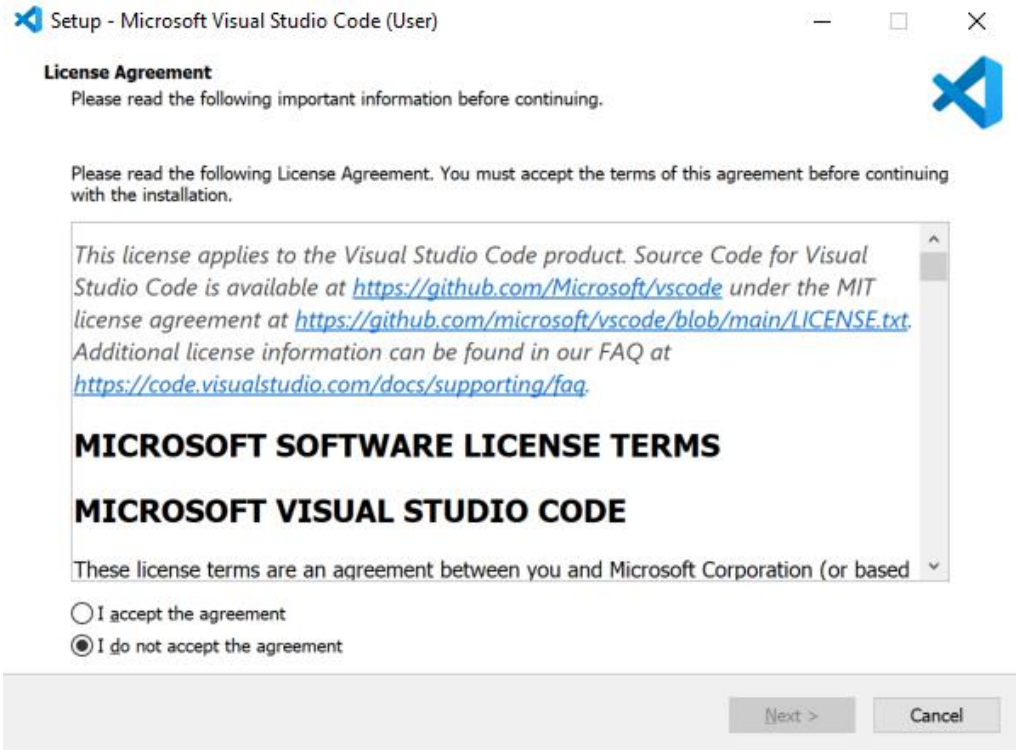
You can then close this **Command Prompt** or **Terminal** as it will no longer be needed in the **Workshop**.

Visual Studio Code

Visual Studio Code is a free **Integrated Development Environment or IDE** created by **Microsoft** and will be used in the **Workshop** and will make writing the application easier. You can **Download** it, if you don't have it already, for **Windows** or **Mac** from a new **Browser** tab code.visualstudio.com



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

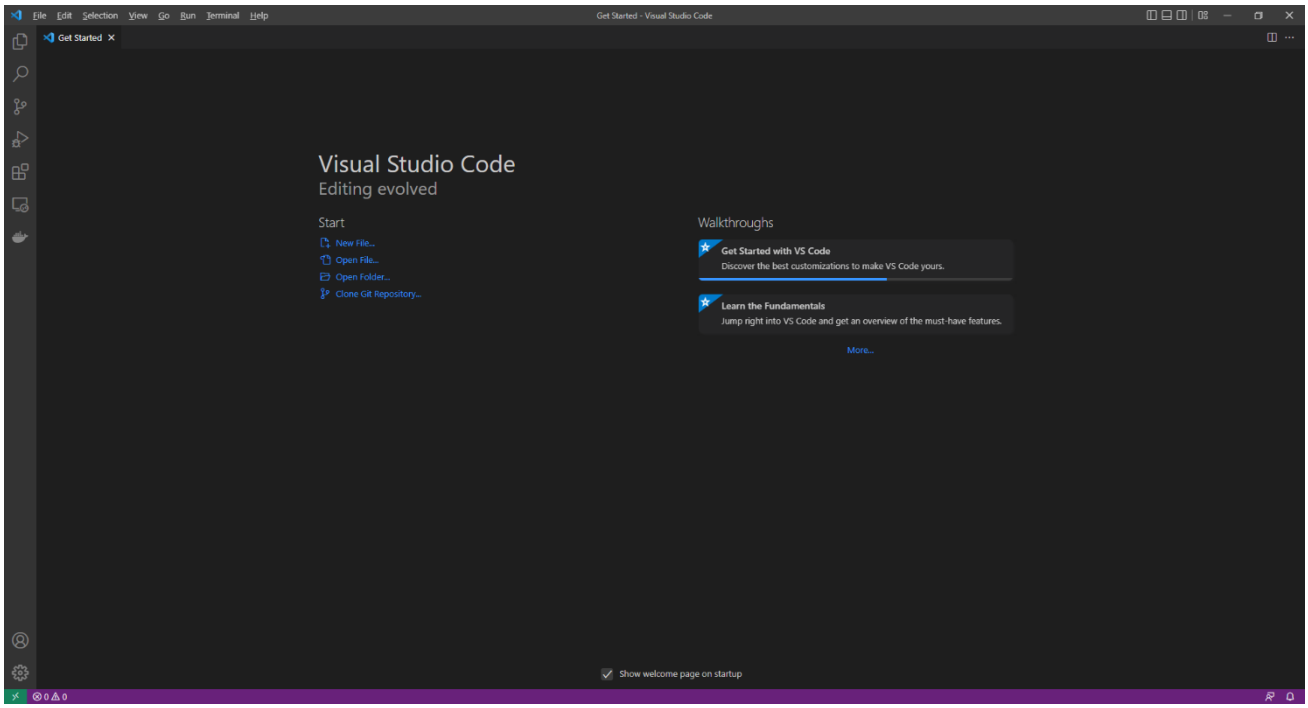


Once you've installed **.NET**, used **`dotnet new blazorwasm -o Blazorfy`**, added the **Packages** and installed **Visual Studio Code** then you're ready for the **Workshop**.

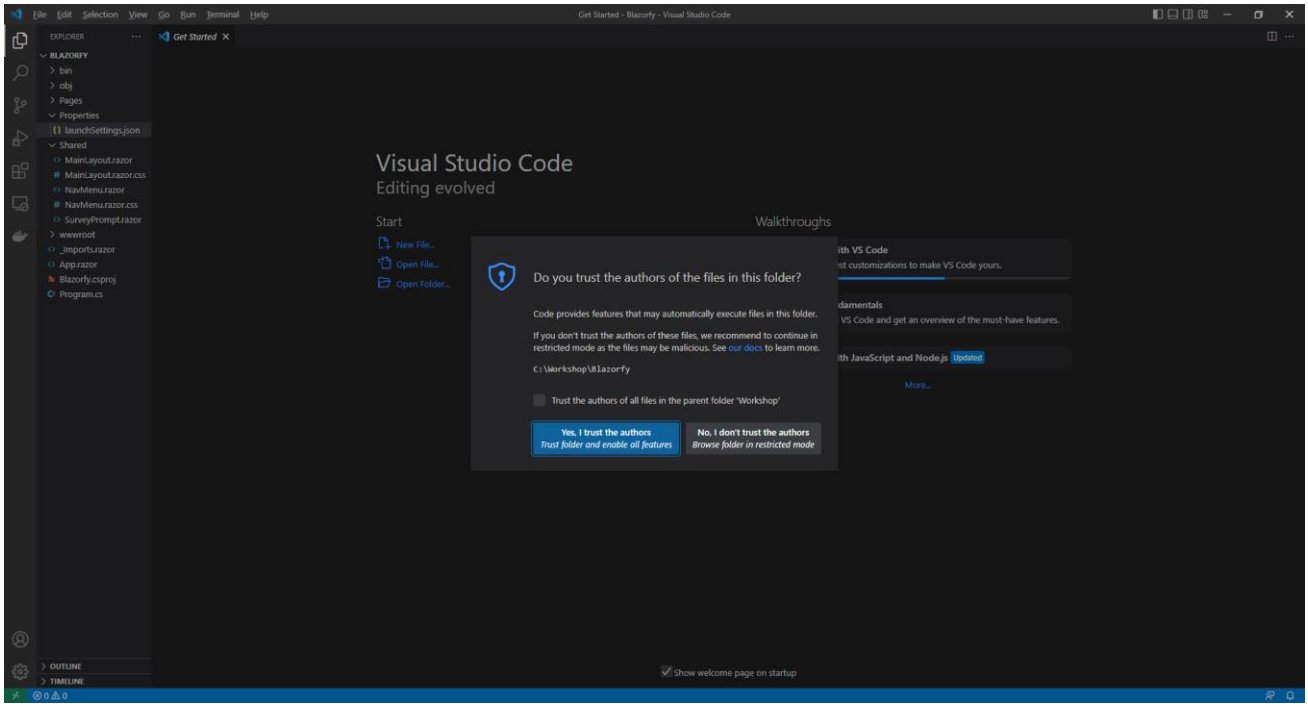
Start

Workspace

Once **Visual Studio Code** has been **Installed**, or was already **Installed** but if it is not already running then if using **Windows** you need to go to **Start** then search for **Visual Studio Code** and then select it or on **Mac** locate it using **Finder** and you should see it loaded with a screen similar to this for **Visual Studio Code**.

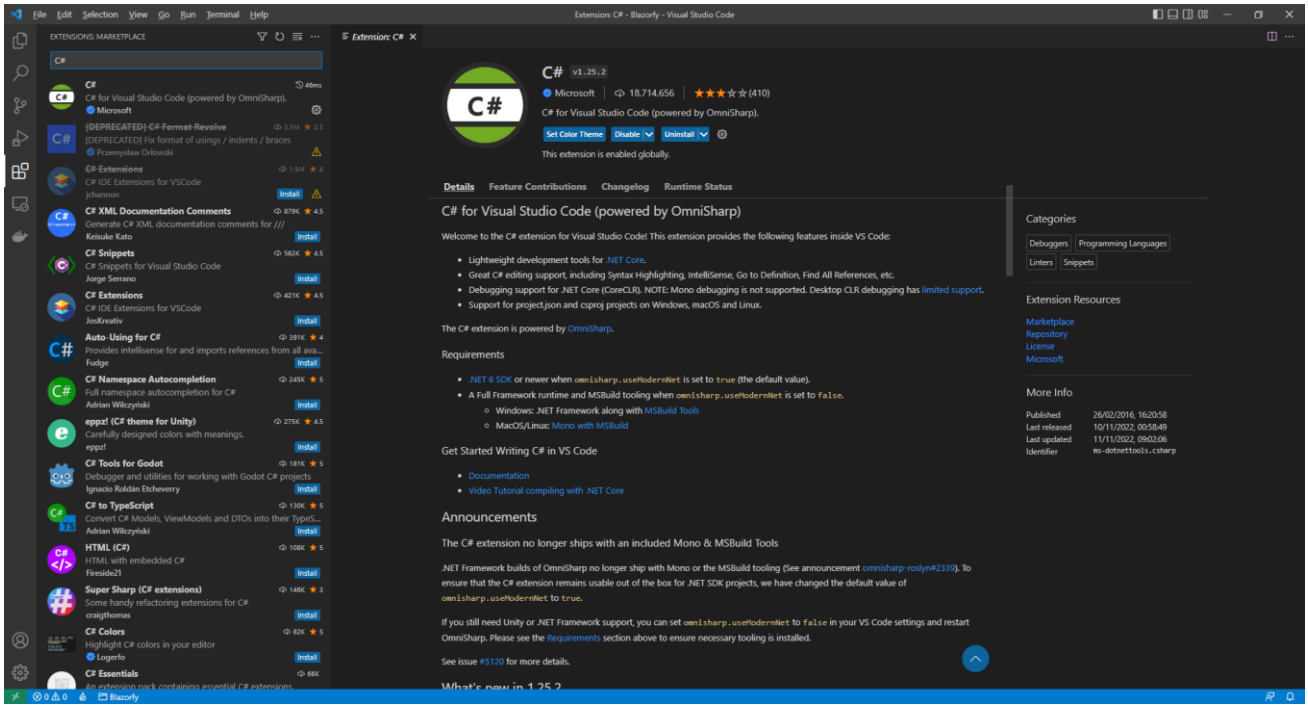


Once **Visual Studio Code** has opened from the **Menu** choose **File** then **Open Folder...** then select the **Folder** for your Application e.g. *C:\Workshop\Blazorfy*. Then to open the **Folder** choose **Select Folder** then one it has been opened Select the **Yes, I trust the authors** option in the **Do you trust the authors of the files in this folder?** if this is displayed which will open the **Workspace**.

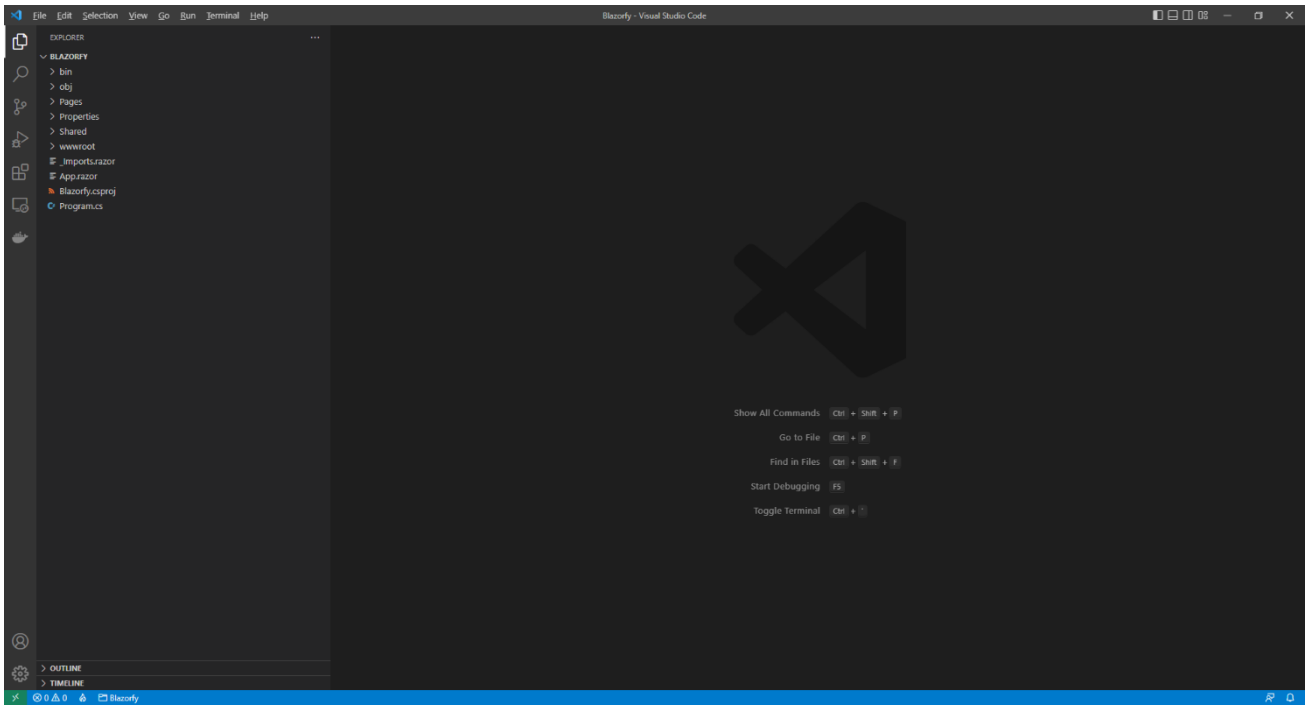


Extension

Then in **Visual Studio Code** select the **Extensions** option from the **Sidebar** then under **Recommended** and then **Install** the **Extension** for **C#** from **Microsoft**:

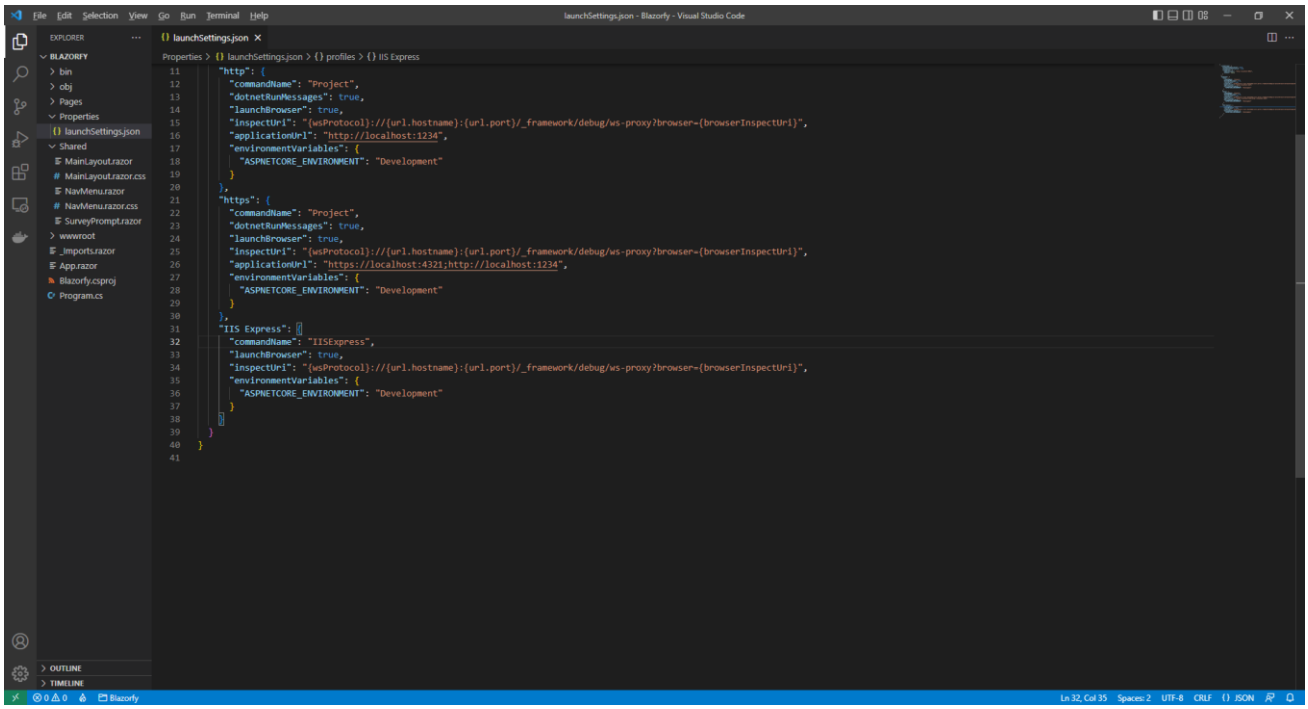


Then once the **Extension** has been installed then you can select the **Explorer** in **Visual Studio Code**



Settings

While still in **Visual Studio Code** from the **Explorer**, which should be the top option from the **Sidebar** in **Visual Studio Code** for **Blazorfy** open **Properties** by selecting the **>** next to it in **Explorer** and select **launchSettings.json** as follows:



Once **launchSettings.json** has been selected look for **applicationUrl** in **launchSettings.json** there may be more than one, and you will see something like *http://localhost:5107* where the digits may be different. Find anything that starts with **http** in **applicationUrl** and change the number to **1234** e.g. *http://localhost:1234*

```
1234
```

Find any entry that starts with **https** within **applicationUrl** and change the number to **4321** e.g. *https://localhost:4321* as follows:

```
4321
```

These changes to **launchSettings.json** will ensure when you start the Application that the address it launches is either of the ones for **http** and **https** as this has been set up in the **Dashboard** to be used as the **Redirect URI** in **Spotify for Developers**.

Imports

While still in **Visual Studio Code** from the **Explorer** for **Blazorfy** select **_Imports.razor** then below **`@using Blazorfy.Shared`** type in the following:

```
@using Spotify.NetStandard.Responses
```

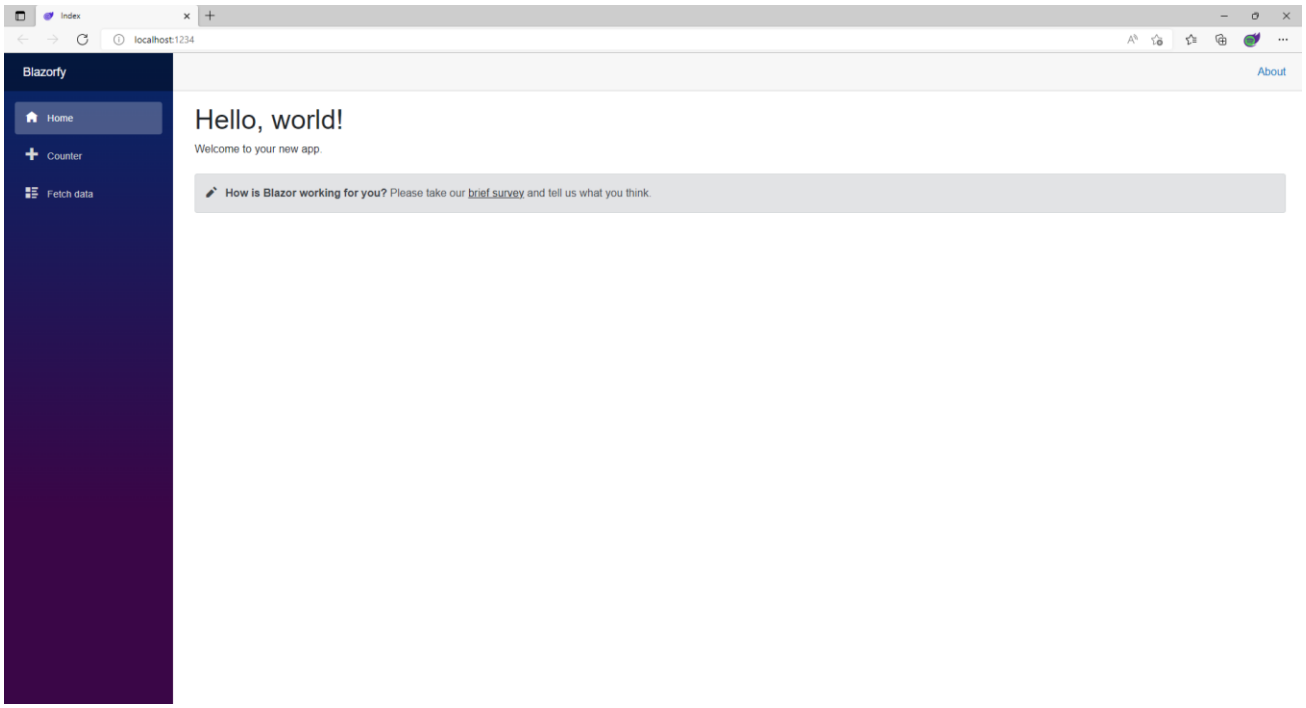
This will allow the **Package** of *Spotify.NetStandard* to be used correctly later in the **Workshop**.

You can then go to the **Menu** in **Visual Studio Code** and select **File** and then **Save All**.

Once done, while still in **Visual Studio Code**, select **Terminal** and then **New Terminal** and then once the **Terminal** has appeared type in the following command and then press **Enter**.

```
dotnet watch
```

Once done this will **Build** and **Start** the Application and display it in your **Browser** with *http://localhost:1234* or *https://localhost:4321* in the **Address Bar** as follows:



Make sure to keep the **Browser** open throughout the **Workshop**. However if you accidently close the **Browser** then you can return to **Visual Studio Code** and select the **Terminal** and then press **Ctrl+C** in **Windows** or **Command+C** on **Mac** on the **Keyboard** and then in the **Terminal** type **`dotnet watch`** again which should relaunch the **Browser** or if you close **Visual Studio Code** then you can just launch **Visual Studio Code** again then from the **Terminal** type **`dotnet watch`** to launch the **Browser**.

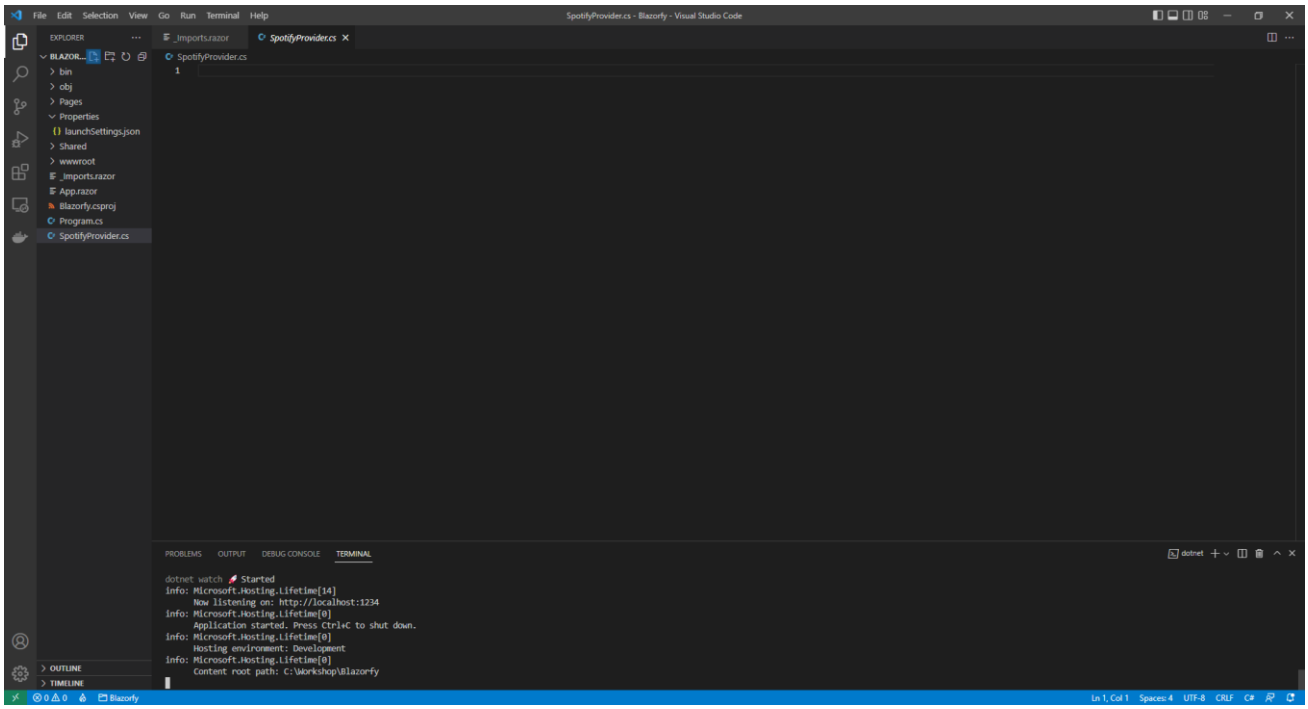
Provider

The next we will start to write a `class` which represents something in **C#**, in this case will be **Provider**. This will allow you to group the main functionality to use **Spotify** in one place and create reusable code, this concept in software is known as *Don't Repeat Yourself* or *DRY*.

Within **Visual Studio Code** from the **Explorer** move the **Cursor** over **Blazorfy** you will see a **New File...** option, if you select this and then type in the name as follows and then press **Enter**:

```
SpotifyProvider.cs
```

Once you press **Enter** after typing in the name you should see a blank *SpotifyProvider.cs* or you can select it from the **Explorer** in **Visual Studio Code** so you can see it as follows:



Should you make any mistakes with the **C#** in this **Workshop** then you will see **Errors** in the **Terminal** when you **Save** any changes. So if you see any **Errors** double check you haven't missed anything, the key thing to remember is balance, you will be using a lot of curly braces that open like so **{** but will always have a counterpart of **}** this also applies to square brackets that will have both **[** and **]** and rounded brackets of **(** and **)** so it is a good idea to check if these are balanced, if you see any double-quotes or **"** then you should always expect to see another **"** nearby. Where you see any semi colons or **;** remember to include them, sometimes the smallest mistake that is easy to fix makes it work once corrected!

Should you make any mistakes with the **HTML** or **Razor** these may be harder to spot and may just not look correct in the **Browser** so make sure any angled brackets you see should open with **<** then you should expect to see **>** nearby although you might see one on their own in **C#** but for **C#** that's okay!

Errors will give you an idea of where to look for the mistake, they will often give a line number which you can check against the value shown at the bottom of **Visual Studio Code** you can always **Copy** and **Paste** any code in the **Workshop** but read through what you copied to see if you understand what it is doing!

Namespaces

While still in **Visual Studio Code** at the top of *SpotifyProvider.cs* from the **Explorer** type in the following:

```
using Blazored.LocalStorage;
using Microsoft.AspNetCore.Components;
using Spotify.NetStandard.Client;
using Spotify.NetStandard.Client.Authentication;
using Spotify.NetStandard.Client.Interfaces;
using Spotify.NetStandard.Requests;
using Spotify.NetStandard.Responses;

namespace Blazorfy;

// Provider Class
```

C# has **`namespaces`** that group together related functionality and you can use existing functionality by including them at the top of a **`class`** with **`using`** and in this case they are for the **Packages** that were added for *Blazored.LocalStorage* and *Spotify.NetStandard* along with one that is needed from **.NET**.

Also please check these have been typed in correctly or you can **Copy** and **Paste** as in **C#** casing matters, for example *spotify.netstandard.client* is wrong but *Spotify.NetStandard.Client* is correct.

There is also a **`namespace`** of **`Blazorfy`** which will help group together the objects for the **Workshop** and finally there is a **Comment** which is anything with **`//`** in front of it below such as **`// Provider Class`** below which the **`class`** will be defined in the next part of the **Workshop**.

If you need to format any **Code** you have **Copy** and **Pasted** in **Visual Studio Code** you can do so with **Shift+Alt+F** on **Windows** or **Shift+Option+F** on **Mac** or right-click in any file and select **Format Document**.

Class

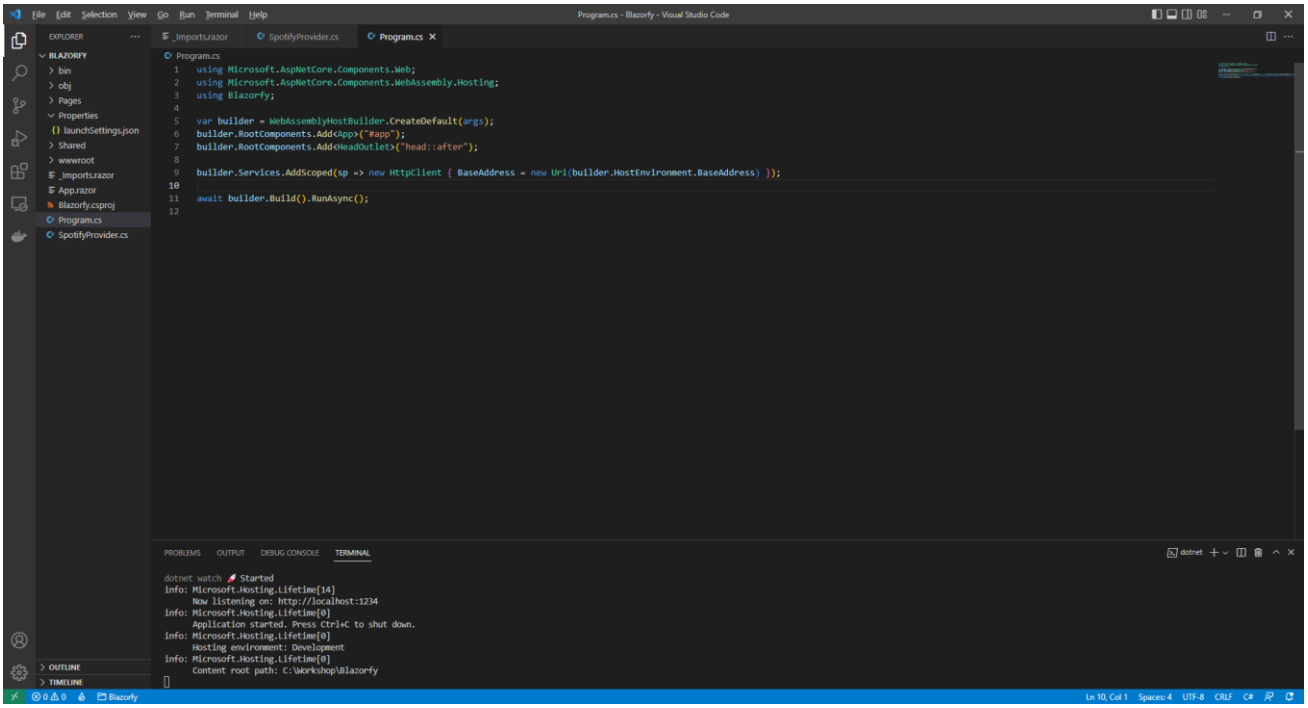
While in **Visual Studio Code** for *SpotifyProvider.cs* below the **Comment** of **`// Provider Class`** type in the following which will define the structure of **`class`** for the **Provider** with **Comments** to help you put things in the right place later in the **Workshop**:

```
public class SpotifyProvider
{
    // Constants

    // Members

    // Private Methods

    // Constructor

    // Property

    // Login Method

    // Logout Method

    // Is Logged In Method

    // Handle Code Method

    // User Method

    // List Method

    // Search Method

}
```

This **`class`** will be populated with the functionality for the **Provider** and will be used throughout the **Workshop** and can use the **Comments** so you know where you need to put things in the **`class`**.
You will be guided through each part step-by-step but the next part will be to add **`class`** so it can be used later in the next part of the **Workshop**.

Program

In **Visual Studio Code** you will also see a *Program.cs* file in the **Explorer** that when selected should be similar to the following:



Within *Program.cs* add **`using`** for *Blazored.LocalStorage* below **`using Blazorfy;`** by typing in the following:

```
using Blazored.LocalStorage;
```

Then while still in *Program.cs* and above the **`await builder.Build().RunAsync();`** type in the following:

```
builder.Services.AddBlazoredLocalStorage();
builder.Services.AddScoped<SpotifyProvider>();
```

You can then go to the **Menu** in **Visual Studio Code** and select **File** and then **Save All**, you may see in the **Terminal** a message saying **Do you want to restart your app - Yes (y) / No (n) / Always (a) / Never (v)?** you can select the **Terminal** then type **y** for **Yes** or **a** for **Always** to keep what you have done so far.

This will add what is needed by *Blazored.LocalStorage* and will also add the **`class`** of **SpotifyProvider** to be available to the **Dependency Injection** system used in **Blazor. Dependency Injection** allows specific functionality to be provided to an application to anywhere that needs it. In **C#** an **Instance** of a **`class`** is needed in order for it to be used but by adding the **`class`** this way we can get **Dependency Injection** to do it for us, if you want to know more about it this concept then you can read up on it after you have completed the **Workshop**.

At this point you should have modified the files of *launchSettings.json* and *Program.cs* along with creating a file called *SpotifyProvider.cs* you can go over the previous steps now to double-check you've done everything correctly then proceed to implementing **Authentication** in the next section of the **Workshop**.

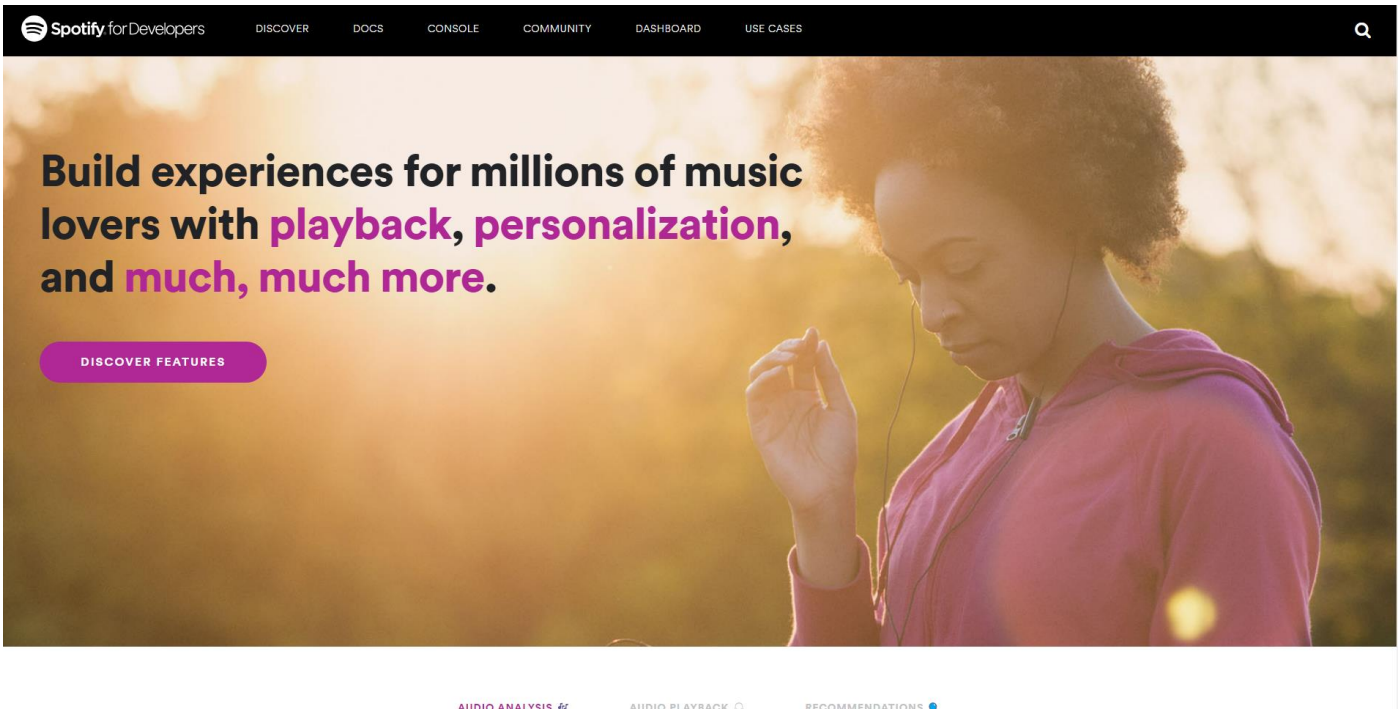
Authentication

Account

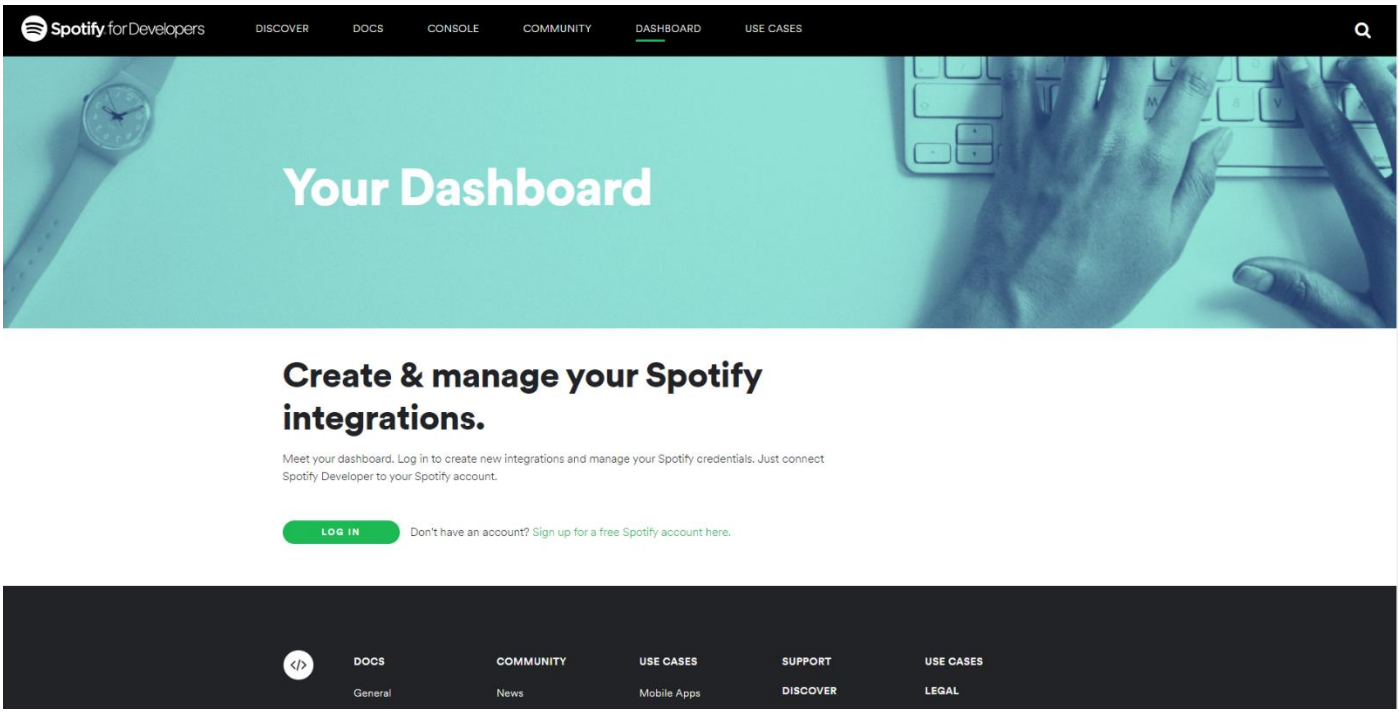
In this part of the **Workshop** you will learn how to **Login** to **Spotify** and also **Logout** if needed. You don't need to have an **Account** as you will use one of the previously setup **Accounts**. You will already have **Number** assigned for the **Account** and will also be provided with the **Password** but you can copy the corresponding **Client Id** from here when you need it later from the following **Table**:

#	Username	Client Id
1	techonthetyne1@tutorialr.com	e145ef235fb54627810d182c12076642
2	techonthetyne2@tutorialr.com	db1599c5960d40e2a24dd272792d391f
3	techonthetyne3@tutorialr.com	695d0ee412a04eb2a3fbd120376f228a
4	techonthetyne4@tutorialr.com	3ef9788fc92b464fbb5423fbf4a20d8c
5	techonthetyne5@tutorialr.com	b074f692870a47cf97e70861feda0147
6	techonthetyne6@tutorialr.com	b06e9f2538db45ef98f1ab595eba73cd
7	techonthetyne7@tutorialr.com	39a9c71a2b05436c864448fc67ce7f4f
8	techonthetyne8@tutorialr.com	de3ff52f03214014be051ab91d5ef37c
9	techonthetyne9@tutorialr.com	16eb7697bc4c4ec2bf14bcce4e4316fa
10	techonthetyne10@tutorialr.com	6ad8db7304a34bc5ba7fd0d8034df336
11	techonthetyne11@tutorialr.com	d21c4bfca4d74abda7f04d8fb8f2b631
12	techonthetyne12@tutorialr.com	f1984003479c4c3ca5be549157de4c88
13	techonthetyne13@tutorialr.com	5d296f3b990040b0b3a5b629ef6420d0
14	techonthetyne14@tutorialr.com	2517788b3a8f440ea989d99bb358a1b4
15	techonthetyne15@tutorialr.com	c3952faa8a6d4e07ac02a8869663d2af
16	techonthetyne16@tutorialr.com	66886119ceca4672900027ca5c168bf9
17	techonthetyne17@tutorialr.com	a9fe31d3e97a4cf7a8c686c2c305ae6a
18	techonthetyne18@tutorialr.com	2cde5d4d7a3d4247b24d077817f0f9b3
19	techonthetyne19@tutorialr.com	be8af1a7cf134e03865f274c491a7649
20	techonthetyne20@tutorialr.com	79f49441f6e14ff79664ec060e07a7b8
21	techonthetyne21@tutorialr.com	a5bb5dfd5e214e07b2853943003197ca
22	techonthetyne22@tutorialr.com	3681202ecc134a05b2a449c1d4425b0c
23	techonthetyne23@tutorialr.com	a86f56ec49234a5286c5fff31462d894
24	techonthetyne24@tutorialr.com	95bd1ea033bf458a953541ccec202c30
25	techonthetyne25@tutorialr.com	3cf4392ac1d9419199f6da98331db256
26	techonthetyne26@tutorialr.com	068a034bc93445b0983649814698e46b
27	techonthetyne27@tutorialr.com	3d29ad83c1ae421bbbf4874c21de1011
28	techonthetyne28@tutorialr.com	bcb8f65a2af24335ab2522d98d42e990
29	techonthetyne29@tutorialr.com	6573c00065de4091ac396cbc3fe22024
30	techonthetyne30@tutorialr.com	ee5642bc26554a33bf2c0b37169b87e5
31	techonthetyne31@tutorialr.com	45ea098fbfa7459f8d1c5afd7f2ef09e
32	techonthetyne31@tutorialr.com	57ea0de9e9254c0d94399f25e69cc979
33	techonthetyne33@tutorialr.com	26dc706d610841d38321515e751b5ea8
34	techonthetyne34@tutorialr.com	e7e7e54c80ed4c288828e554fa483677
35	techonthetyne35@tutorialr.com	9932221f47794afda531b58cdcd51094
36	techonthetyne36@tutorialr.com	1bea6faff23c4130b931e5d480ff8833
37	techonthetyne37@tutorialr.com	dc324de7dd9144ee879ac303110d0e76
38	techonthetyne38@tutorialr.com	b7fbd28fe83e4864909c9017c3b691c7
39	techonthetyne39@tutorialr.com	2f983251830f4ea9a03fad17a2d37f3b
40	techonthetyne40@tutorialr.com	153910e67ed245c7bf1326d2b4d71fc7

Optional as you can skip to **Constants** if you are using one of the **Accounts** for the **Workshop** but if not you can **Create** your own **Account** on **Spotify** by first launching another **Browser** and go to the **Spotify for Developers** website at developer.spotify.com.

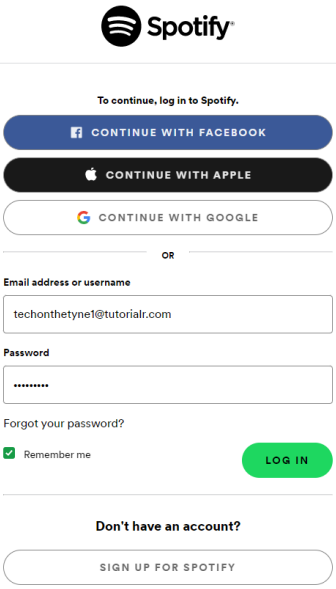


Once on the website for **Spotify for Developers** selection option for **Dashboard** as follows:

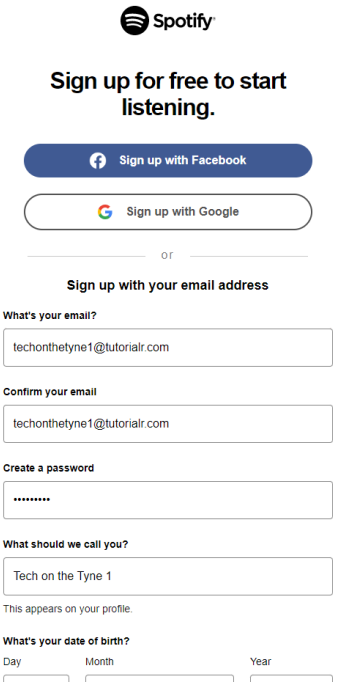


The **Dashboard** is where you can **Create** and **Manage** integrations that use **Spotify**.

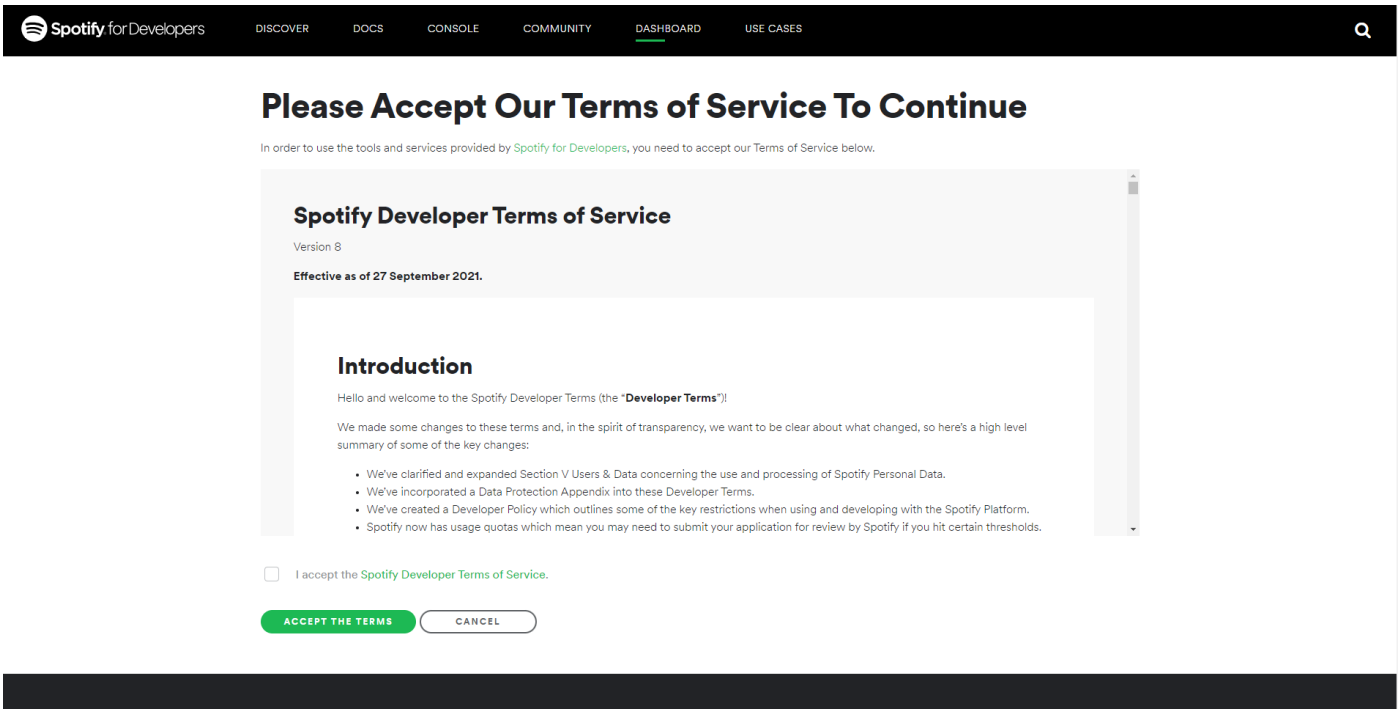
If you have an existing **Account** for **Spotify** select **Log In** and then type in your **Email address or username** and **Password** on the following screen and then select **Log In**:



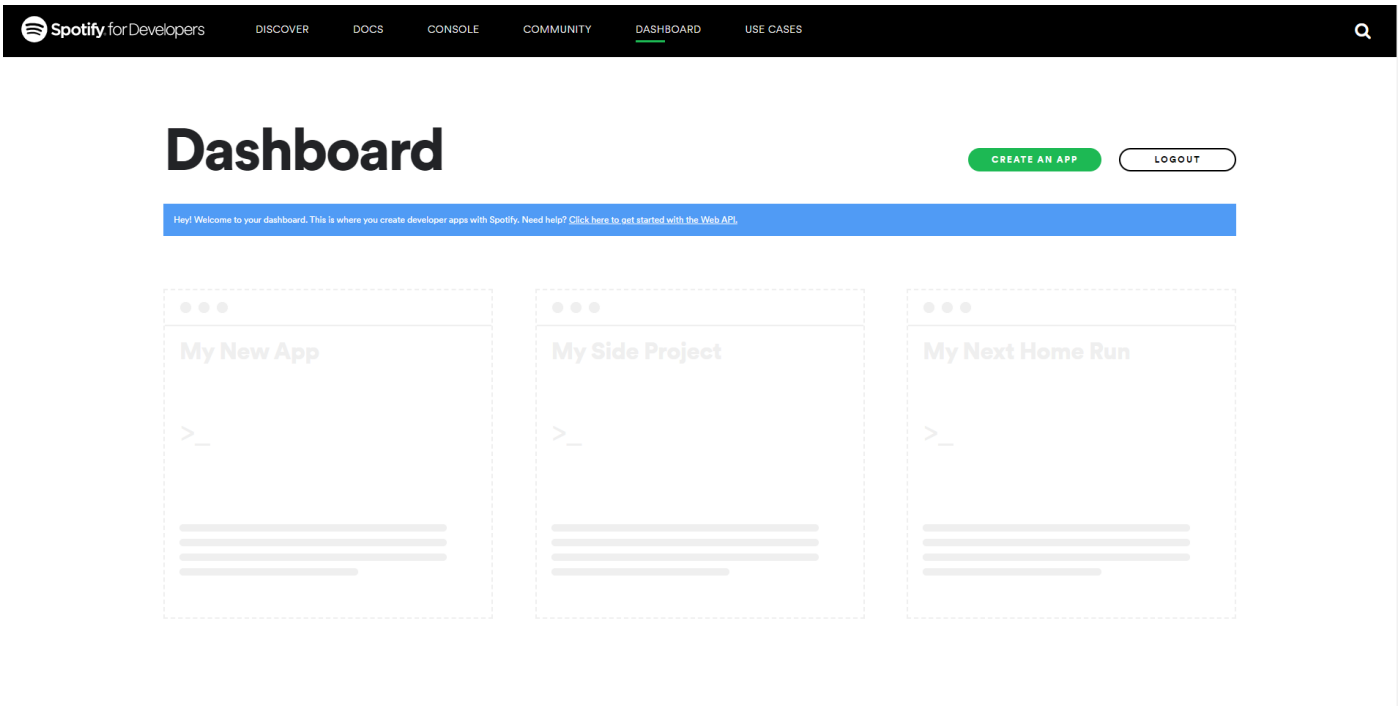
If don't have an existing **Account** for **Spotify** then select **Sign-up for a free Spotify account here** and then select **Sign up for Spotify** and then fill in the details on the following page:



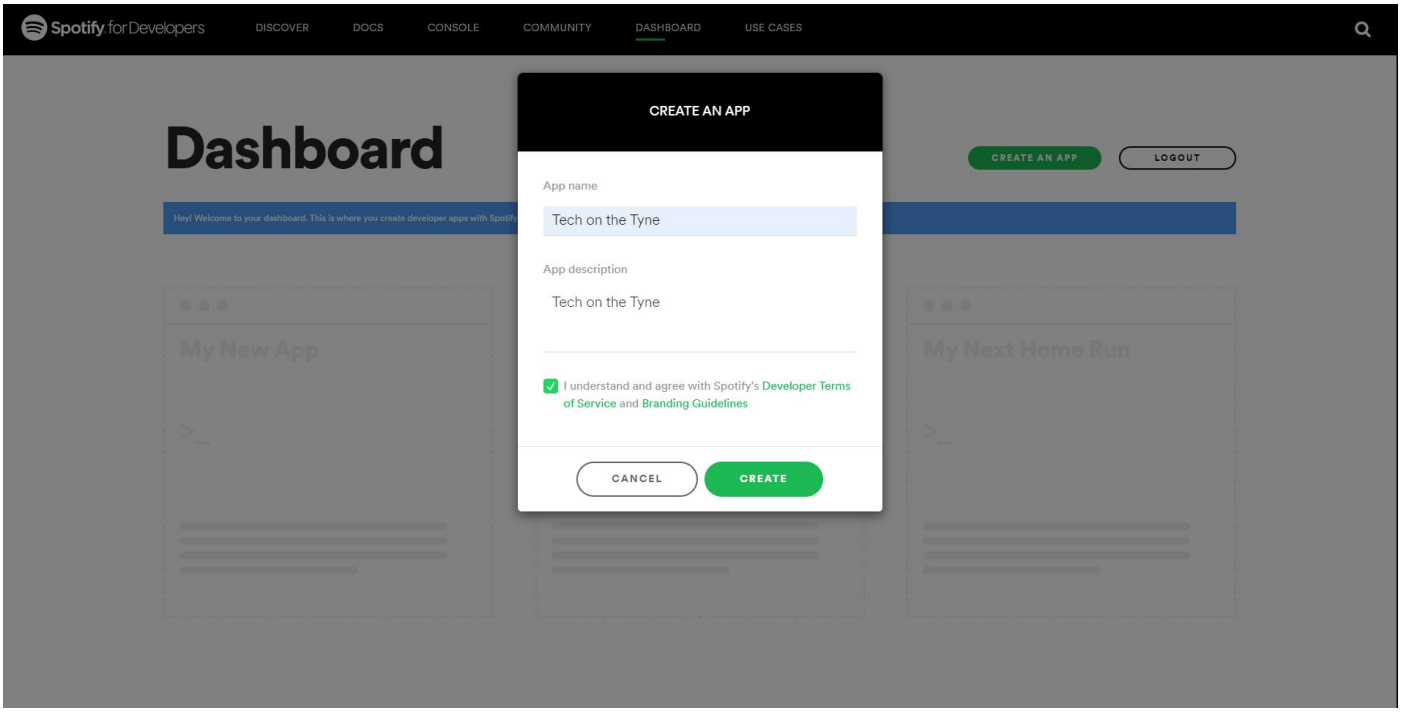
Once the **Account** has been created or if have logged into an existing **Account** for **Spotify** then you should get the following option to **Accept** the Terms and Conditions, just read through this then select the **I accept the Spotify Developer Terms of Service** and then select **Accept the Terms** as follows:



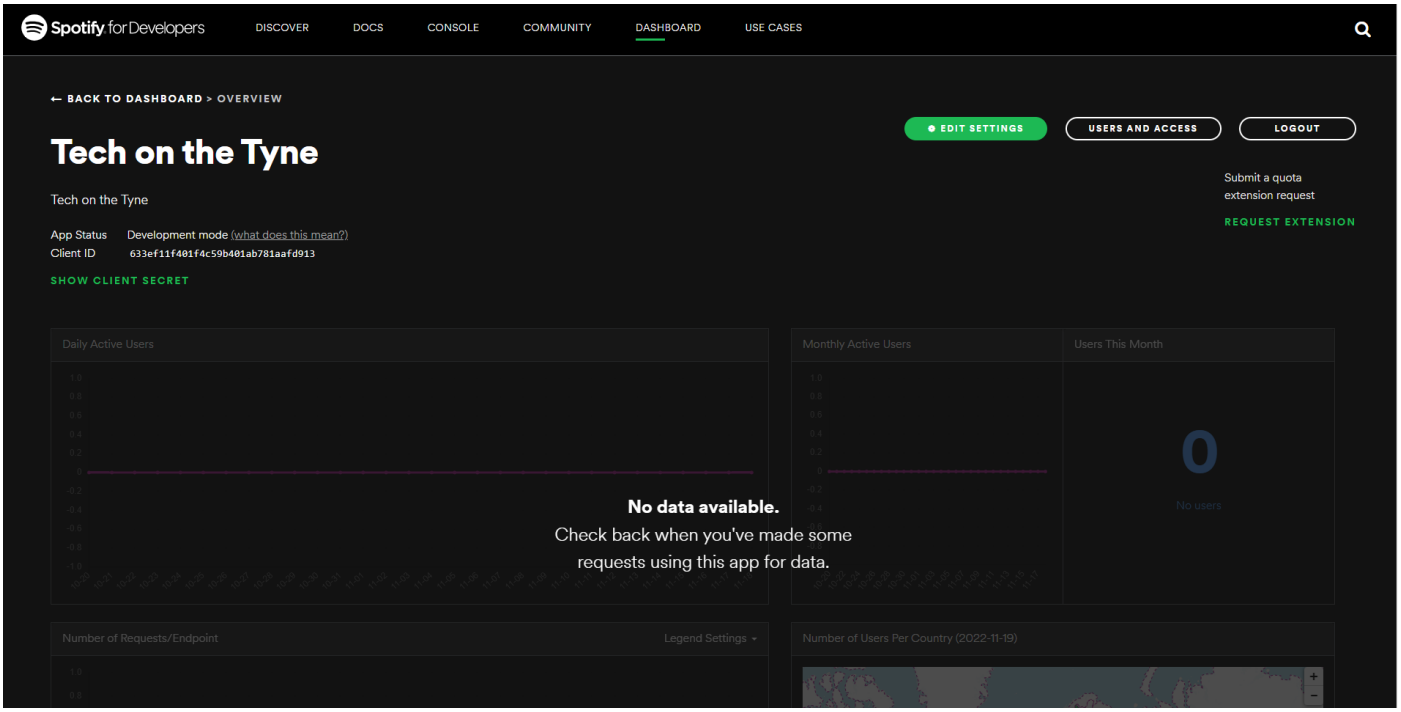
Once you have selected **Accept the Terms** then you will be taken to the **Dashboard** as follows:



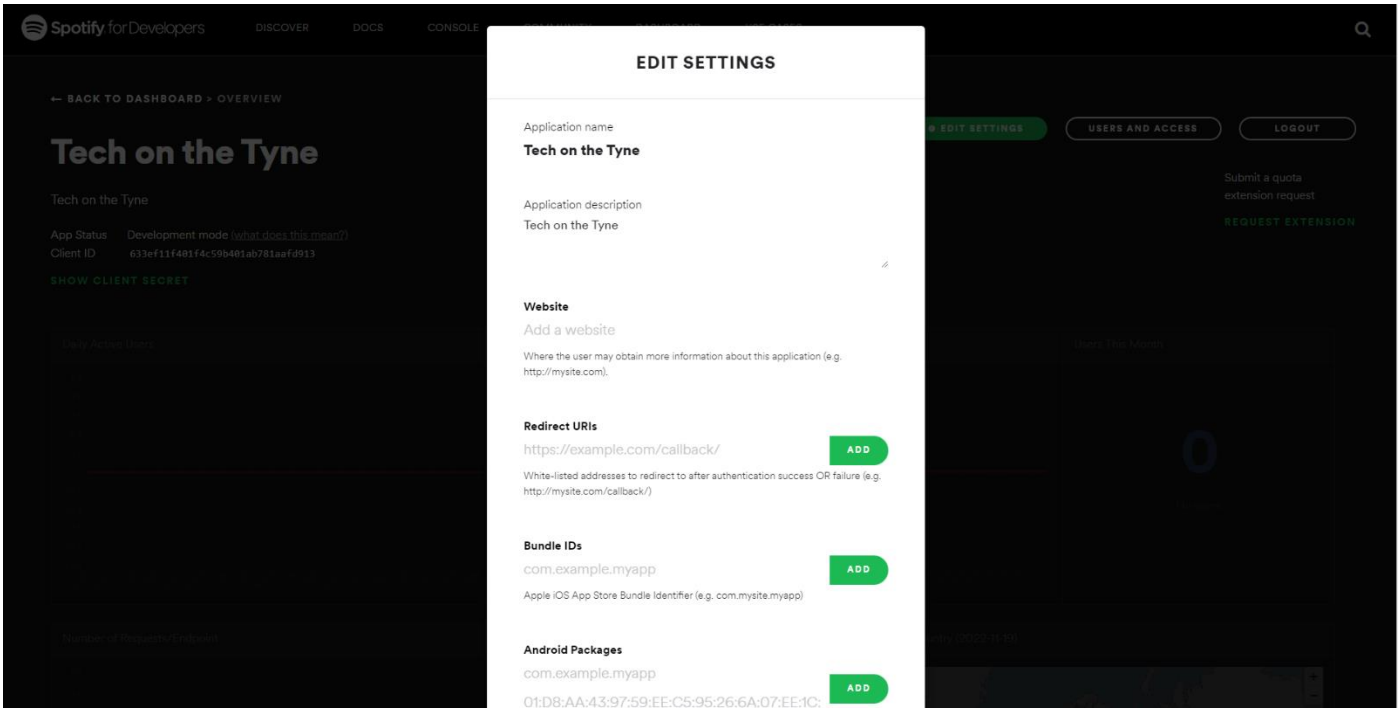
Once in the **Dashboard** you can then select **Create an App** and then enter the **App Name** as *Tech on the Tyne* the **App Description** as *Tech on the Tyne* and then after reading the **Developer Terms of Service** and **Branding Guidelines** select the option for **I understand and agree with Spotify's Developer Terms of Service and Branding Guidelines** then select the **Create** option.



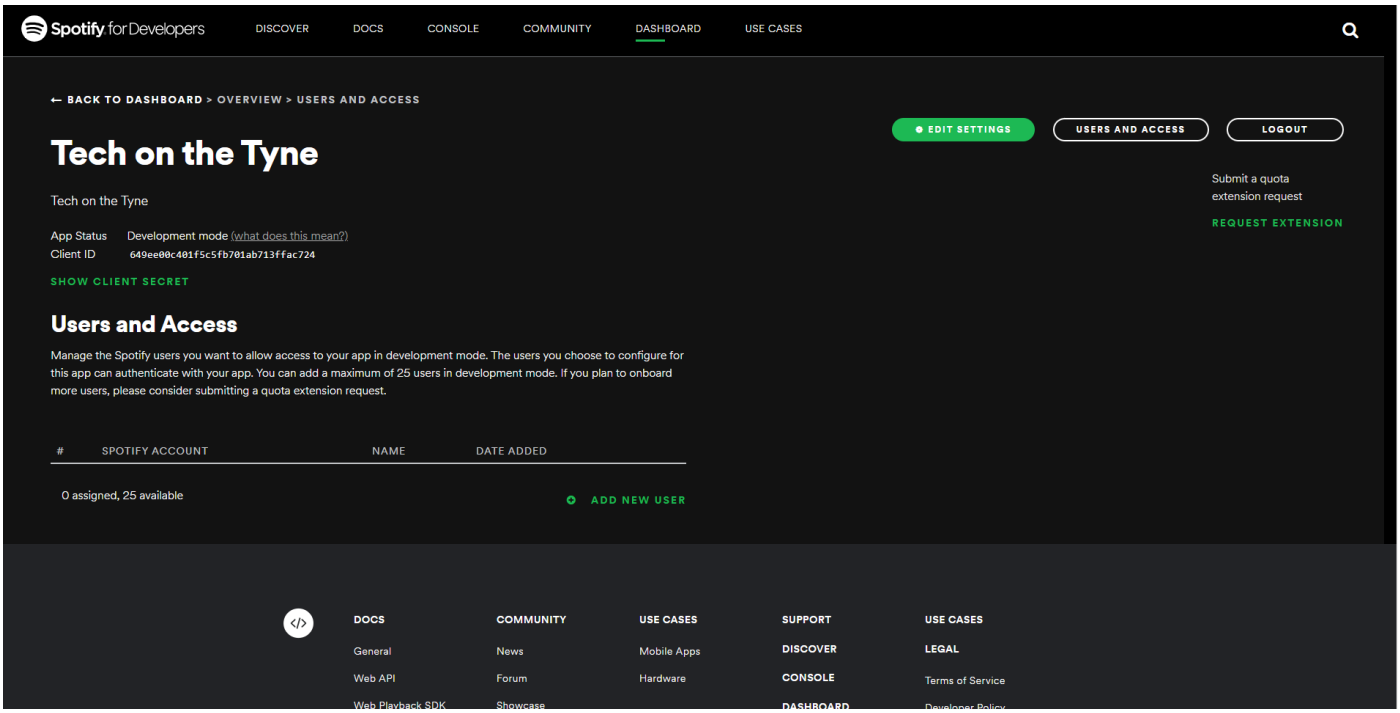
Once the **App** has been created you will see the following **Overview** where you will need to **Copy** and **Paste** the **Client ID** to somewhere so it can used later in the **Workshop** as follows:



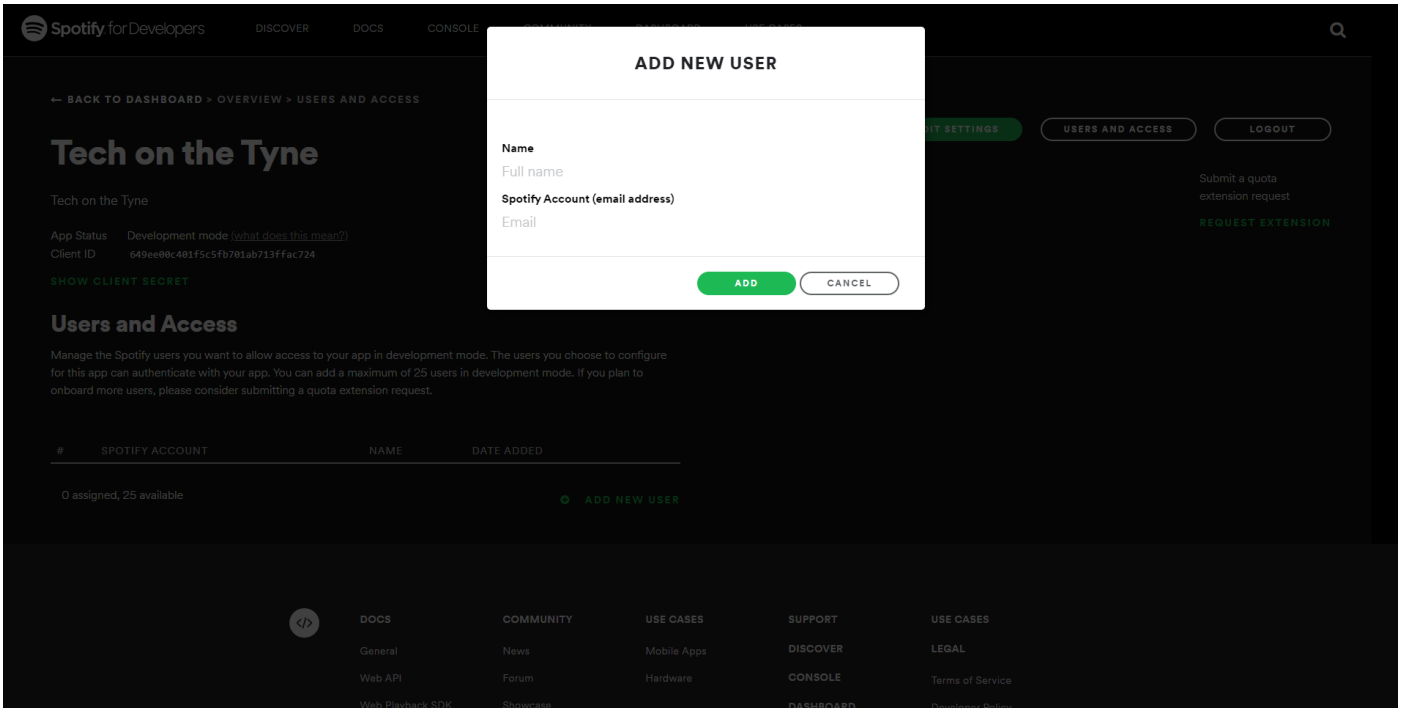
Then select the **Edit Settings** from the **Overview** for the **App** and then within **Edit Settings** in the section for **Redirect URIs** and in the box with *https://example.com/callback* type in **https://localhost:4321/** and select **Add** and then in the box with *https://example.com/callback* type in **http://localhost:1234/** and then select **Add** from the following screen:



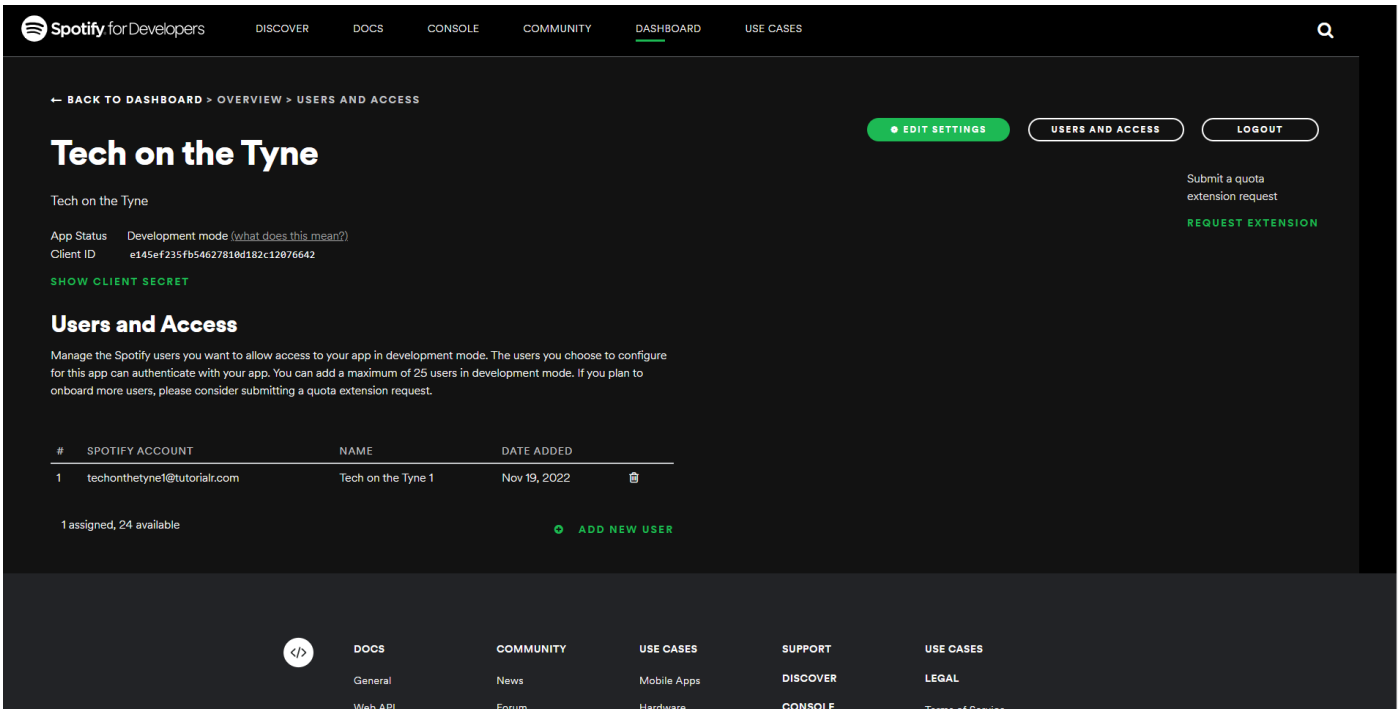
Then from the **Overview** for the **App** select **Users and Access** as follows:



Then from page for **Users and Access** select the option for **Add New User** and then type in the **Name** and **Spotify Account (email address)** for your **Account** on **Spotify** and then select Add as follows:



Once your **Account** shows up in the **Users and Access** and you have the **Client Id** then you can proceed with the rest of the **Workshop**.



Constants

Back in **Visual Studio Code** from the **Explorer** select *SpotifyProvider.cs* which should have the following contents from the **Start** of the **Workshop**:

```
using Blazored.LocalStorage;
using Microsoft.AspNetCore.Components;
using Spotify.NetStandard.Client;
using Spotify.NetStandard.Client.Authentication;
using Spotify.NetStandard.Client.Interfaces;
using Spotify.NetStandard.Requests;
using Spotify.NetStandard.Responses;

namespace Blazorfy;

// Provider Class
public class SpotifyProvider
{
    // Constants

    // Members

    // Private Methods

    // Constructor

    // Property

    // Login Method

    // Logout Method

    // Is Logged In Method

    // Handle Code Method

    // User Method

    // List Method

    // Search Method

}
```

Then in **Visual Studio Code** within *SpotifyProvider.cs* you will define **Constants** with **`const`** such as the **`client_id`** as a **`string`** for text and how many and the maximum number of items that will be retrieved later as an **`int`** for numbers. These will only be used inside the **`class`** so are declared with **`private`**.

Below the **Comment** of **`// Constants`** on type the following, replacing **`clientid`** with your **Client Id**

```
private const string client_id = "clientid";
private const int total = 50;
private const int max = 100;
```

Members

While still in *SpotifyProvider.cs* in **Visual Studio Code** you will define some **Members** by typing below the **Comment** of **`// Members`** the following:

```
private readonly NavigationManager _navigation;
private readonly ILocalStorageService _storage;
private readonly ISpotifyApi _api;
private readonly Uri _redirectUri;
private AccessToken? _token;
```

Members represent values within the **`class`** and these are used within the **`class`** so are marked **`private`** with the first four being set in the **Constructor** later so can use **`readonly`** to not set them again there.

`NavigationManager` is needed to both get the current address or **`URI`** of the page and to redirect to a particular page or **`URI`** when needed.

`ILocalStorageService` is from the **Package** of *Blazored.LocalStorage* and is an **Interface** which allows functionality to be exposed from a **`class`** but be abstracted so the functionality could change but as long as that stays the same anything using the **Interface** will still work, with this functionality being the ability to load or save information needed for the **Workshop** in the **Browser**.

`ISpotifyApi` is also an **Interface** and is from the **Package** of *Spotify.NetStandard* which will be used to perform the functionality needed from **Spotify** which will also use the **`Uri`** which will be passed to **Spotify** and an **`AccessToken`** is defined which will be returned from **Spotify** when have successfully logged in.

Methods

While still in *SpotifyProvider.cs* in **Visual Studio Code** you will define some **Methods** by typing below the **Comment** of **`// Private Methods`** the following:

```
private Uri GetCurrentUri() =>
    _navigation.ToAbsoluteUri(_navigation.Uri);

private async Task SetTokenAsync(AccessToken? token) =>
    await _storage.SetItemAsync(nameof(_token), token);
```

These **Methods** will only be used within the class itself so again are declared with **`private`**. The first one will help get the current address of the page of the **Browser** and uses the **`NavigationManager`** that was declared previously. The second **Method** will use the **`ILocalStorageService`** to store the **`AccessToken`** from **Spotify**. These **Methods** use the Arrow Syntax with the **`=>`** for an Expression Body which is useful when they only have one line to save space.

Constructor

While still in *SpotifyProvider.cs* in **Visual Studio Code** you will define the **Constructor** by typing below the **Comment** of `// Constructor` the following:

```
public SpotifyProvider(
    HttpClient client,
    NavigationManager navigation,
    ILocalStorageService storage)
{
    _storage = storage;
    _navigation = navigation;
    _redirectUri = new Uri(GetCurrentUri().GetLeftPart(UriPartial.Path));
    _api = SpotifyClientFactory.CreateSpotifyClient(client, client_id).Api;
}
```

Constructor sets up a `class` as well as allowing any other `class` or `interface` to be provided using **Dependency Injection** which in this case includes `HttpClient` which will provide the ability to communicate that is used in the **Method** of `SpotifyClientFactory.CreateSpotifyClient` along with the `client_id`. Both the `NavigationManager` and `ILocalStorageService` are also provided and the redirection `Uri` is also set by using the **Method** of `GetCurrentUri`.

Property

While still in *SpotifyProvider.cs* in **Visual Studio Code** you will define a **Property** by typing below the **Comment** of `// Property` the following:

```
public bool IsLoggedIn =>
    _token != null;
```

A **Property** is the best way to expose values outside of a `class` in **C#** which is also done by using `public` so that this value is available to anywhere else that uses this `class`. This **Property** will be used to indicate if the **Account** has been logged or not with a `bool` which is a `true` or **false** value by checking the `AccessToken` which also had a question mark or `?` which means it can have `null` as the value which is what is being checked by using `!=` which means not equal to, so when the value is `null` the **Property** will be `false` and if the value is not `null` the **Property** will be `true`.

You can then go to the **Menu** in **Visual Studio Code** and select **File** and then **Save All** you may see in the **Terminal** a message saying **Do you want to restart your app - Yes (y) / No (n) / Always (a) / Never (v)?** you can select the **Terminal** then type **y** for **Yes** or **a** for **Always** to keep what you have done so far.

Login

Next within *SpotifyProvider.cs* in **Visual Studio Code** you will define a **Method** by typing below the **Comment** of **`// Login Method`** the following:

```
public async Task LoginAsync()
{
    var responseUri = _api.GetAuthorisationCodeAuthUri(
        _redirectUri,
        nameof(SpotifyProvider),
        Scope.None,
        out string codeVerifier);
    await _storage.SetItemAsync(nameof(codeVerifier), codeVerifier);
    if (responseUri != null)
        _navigation.NavigateTo(responseUri.ToString());
}
```

There's a few things going on in this **Method** the first thing to notice is it has **`public`** which allows it to be used outside the **`class`** it also has **`async`** and **`Task`** as this **Method** performs some functionality asynchronously which means something will happen then the result of this action will be waited for with **`await`** and then the application can continue.

The first thing the **Method** does is get the **`Uri`** needed to **Authenticate** with **Spotify** and the **`Uri`** to redirect back to once this is done is provided along with a **state** using **`nameof(SpotifyProvider)`** and a **Scope** which controls what access to **Account** specific functionality is needed but in this case there's nothing of that nature required so *None* is used. Another value is also output using the **`out`** which is a **Code Verifier** which is needed to complete the **Authentication** process.

The next thing the **Method** does is to store the **Code Verifier** in the **Storage** of the **Browser** and it is this **Method** that is asynchronous then if there was a **`Uri`** returned from the first part of the **Method** then finally the **`NavigationManager`** is used to redirect to it on **Spotify**.

Logout

Then within *SpotifyProvider.cs* in **Visual Studio Code** you will define another **Method** by typing below the **Comment** of **`// Logout Method`** the following:

```
public async Task LogoutAsync()
{
    await SetTokenAsync(_token = null);
    _navigation.NavigateTo(_redirectUri.ToString(), true);
}
```

This will be used to log out of an **Account** and it does this by setting the **`AccessToken`** to **`null`** and then storing that using the **Method** of **`SetTokenAsync`** then finally it will use **`NavigationManager`** to redirect to the **`Uri`** for redirection and will force the page to refresh when doing so.

Logged In

Then within *SpotifyProvider.cs* in **Visual Studio Code** you will define the next **Method** by typing below the **Comment** of **`// Is Logged In Method`** the following:

```
public async Task<bool> IsLoggedInAsync()
{
    _token ??= await _storage.GetItemAsync<AccessToken>(nameof(_token));
    if (_token != null)
    {
        if (_token.Expiration < DateTime.UtcNow)
        {
            await LogoutAsync();
        }
        else
        {
            _api.Client.SetToken(_token);
        }
        return IsLoggedIn;
    }
    return false;
}
```

This **Method** checks and does a few things so let's break it down, overall it will **`return`** a value that is either **`true`** to indicate the **Account** is logged in or **`false`** if it is not.

The first thing it does is use the **`ILocalStorageService`** to set the **`AccessToken`** you'll also notice the use of the **`??=`** which is known as the null-coalescing assignment **Operator**, but in plain English it means it will only perform the action to get the **`AccessToken`** if it has not already been set to something, that is it will still be **`null`**.

The **`AccessToken`** is then checked to see **`if`** it is not **`null`** by using the **`!=`** or not equal to **Operator** and if this is **`false`** the next part will be skipped and the **`return`** from the **Method** will be **`false`**.

If the **`AccessToken`** is not **`null`** was **`true`** then next thing is to check if the **`AccessToken`** has expired by comparing the **`Expiration`** against the current date and time in *UTC* and if expired we logout using **`LogoutAsync`** or if it has not expired we will to use a **Method** in the **`Client`** for **`ISpotifyApi`** to set the **`AccessToken`**.

Then we can just **`return`** the **Property** of **`IsLoggedIn`** to indicate the **Account** is logged in.

Handle Code

Next within *SpotifyProvider.cs* in **Visual Studio Code** you will define the next **Method** by typing below the **Comment** of `// Handle Code Method` the following:

```
public async Task<bool> HandleCodeAsync(string? code)
{
    if (code != null)
    {
        string codeVerifier =
            await _storage.GetItemAsync<string>(nameof(codeVerifier));
        _token = await _api.GetAuthorisationCodeAuthTokenAsync(
            GetCurrentUri(),
            _redirectUri,
            nameof(SpotifyProvider),
            codeVerifier);
        await SetTokenAsync(_token);
        _navigation.NavigateTo(_redirectUri.ToString());
    }
    return await IsLoggedInAsync();
}
```

This **Method** will be used as part of the process of logging into **Account** when the process is competed in the **Browser** for **Spotify** it will redirect back to the redirection `Uri` that was specified along with returning a **Code** which will be handled with this **Method**.

The first thing is that the value passed in is checked to see if it is not `null` with the `!=` or not equal to **Operator** and if it does have a value then it can then get the **Code Verifier** from the **Storage** in the **Browser** then this is used with the **Method** of `GetAuthorisationCodeAuthTokenAsync` in `ISpotifyApi`. The response `Uri` is needed which will be used to get the **Code** that was passed along from **Spotify** and the same **State** is used as before and the **Code Verifier** is also provided.

The next thing is the **Method** for `SetTokenAsync` is called to store the `AccessToken` and then the `NavigationManager` will be used to reload the page to complete the logging in process, should no **Code** be provided then the result of the **Method** of `IsLoggedInAsync` will be used instead.

User

Finally within *SpotifyProvider.cs* in **Visual Studio Code** you will define the **Method** to get the **User** for the **Account** by typing below the **Comment** of `// User Method` the following:

```
public async Task<PrivateUser> GetUserAsync() =>
    await _api.GetUserProfileAsync();
```

This **Method** uses the Arrow Syntax with the `=>` for an Expression Body which is useful when a **Method** only has one line to save space and will get the **User** for the Account which will be used in a **Component** which will be created in the next part of the **Workshop**.

Component

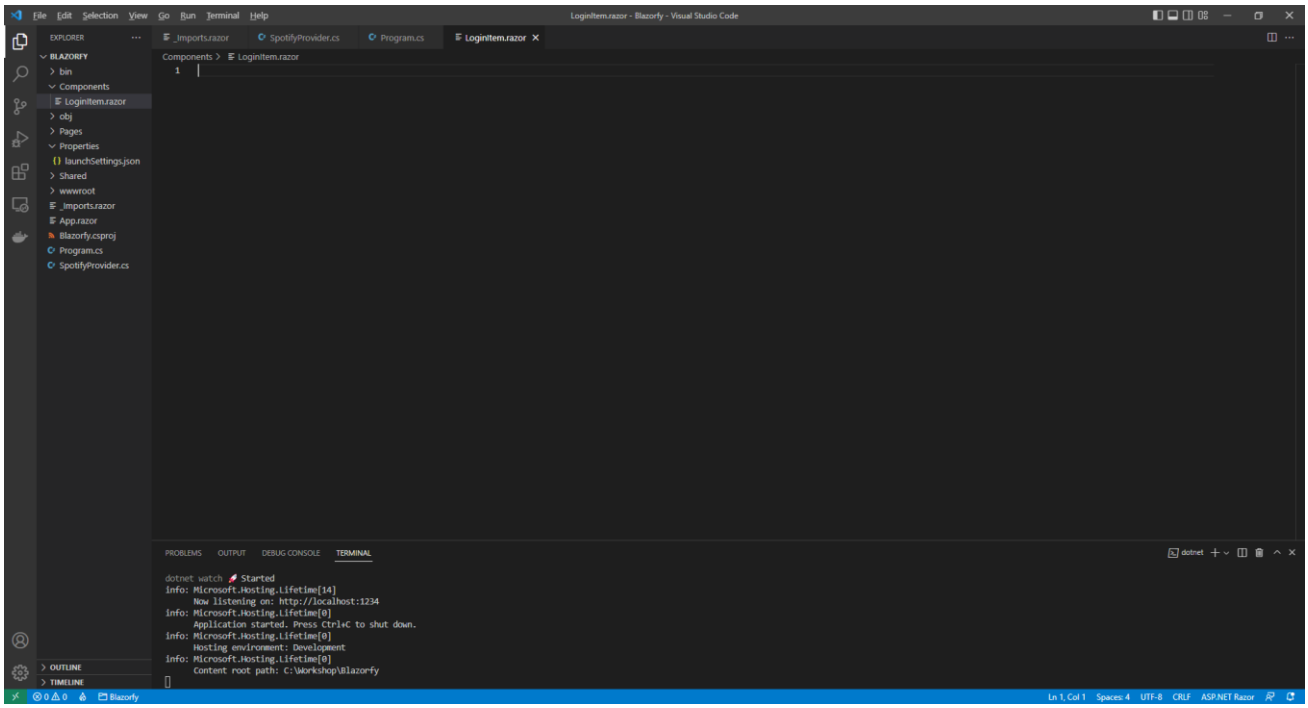
Within **Visual Studio Code** from the **Explorer** and move the **Cursor** over the **Blazorfy** you will see a **New Folder...** option next to the **New File...** option if you select **New Folder...** and then type in the name as follows then press **Enter**:

```
Components
```

With the **Folder** for **Components** selected you should then select the **New File...** option and type in the name as follows then press **Enter**:

```
LoginItem.razor
```

This will form the basis of a **Razor Component** which is also known as a **Blazor Component** in **Blazor** or just **Component** in the **Workshop** and for now you should have a blank **Component** as follows:



Components allow you to reuse or define either some functionality or some **Razor** and **HTML** to create a piece or **Component** of an application that you can see in **Blazor**.

Within *LoginItem.razor* in **Visual Studio Code** you can define the **Component** by typing in the following:

```
@namespace Blazorfy
@inject SpotifyProvider _provider;
@if (Value)
{
    <button class="btn btn-danger" @onclick="_provider.LogoutAsync">Logout</button>
    <span class="badge bg-primary">@Item?.DisplayName</span>
}
else
{
    <button class="btn btn-success" @onclick="_provider.LoginAsync">Login</button>
}

@code
{
    [Parameter]
    public bool Value { get; set; }

    public PrivateUser? Item { get; set; }

    protected override async Task OnParametersSetAsync()
    {
        if (Value)
        {
            Item = await _provider.GetUserAsync();
        }
    }
}
```

The first part of the **Component** is the **`namespace`** for the application which is **`Blazorfy`** then the next part will provide the **Instance** of the **`SpotifyProvider`** using **Dependency Injection** with **`inject`**.
There's a **Property** for **`Value`** which can be either **`true`** or **`false`** as it is a **`bool`** and will be used to indicate to the **Component** that the **Account** is logged in or not.

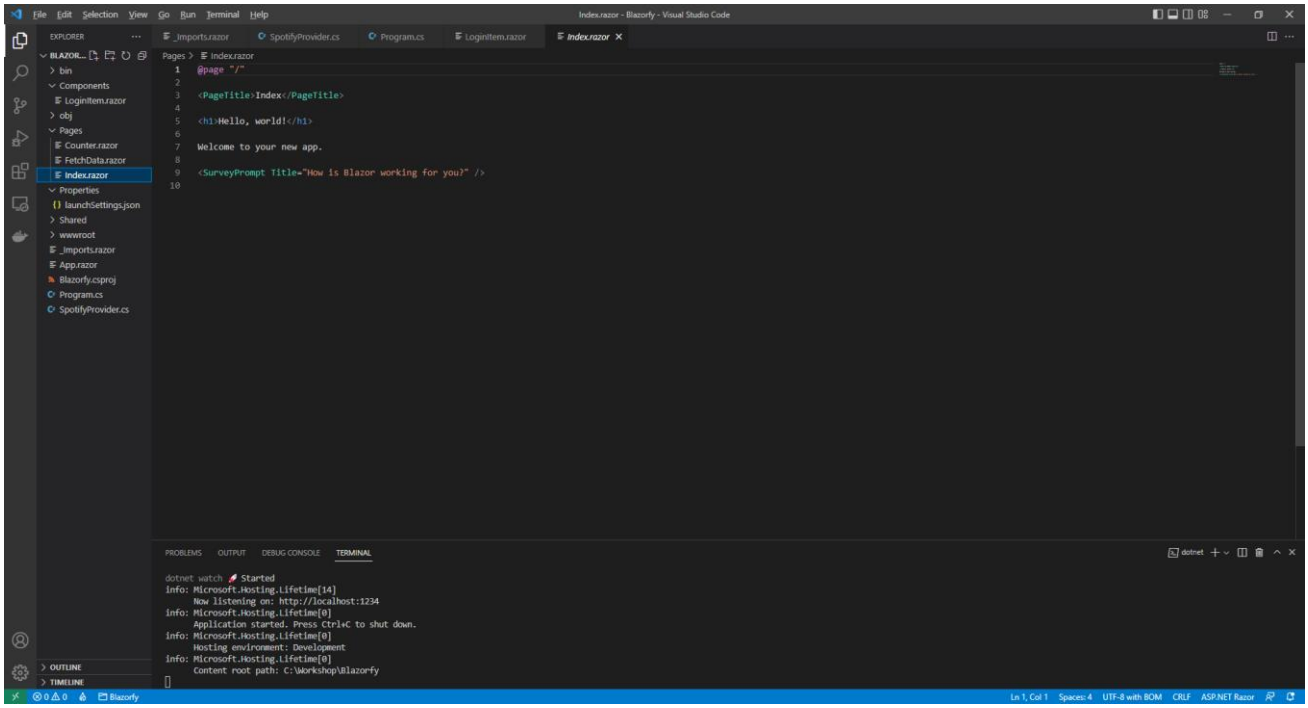
If the **Account** is logged in, which will be provided to the **Component**, then the **Property** for **Value** will be **`true`** the with the **DisplayName** of the currently logged in **Account** and a **`button`** to **Logout** will be displayed which will call the **Method** of **`LoginAsync`** from the **`class`**. Should the **Property** for **`Value`** be **`false`** then the option to **Login** will be displayed which will call the **Method** of **`LogoutAsync`**.

There's also an additional **Property** for a **`PrivateUser`** which is for the user of the **Account** from **Spotify** then there is a special **Method** where the implementation of which has been overridden to provide our own denoted with **`override`** in this case it is for **`OnParametersSetAsync`** which is called when the **Properties** for the **Component** are set by **Blazor** and within this we get the details for the user of the **Account**.

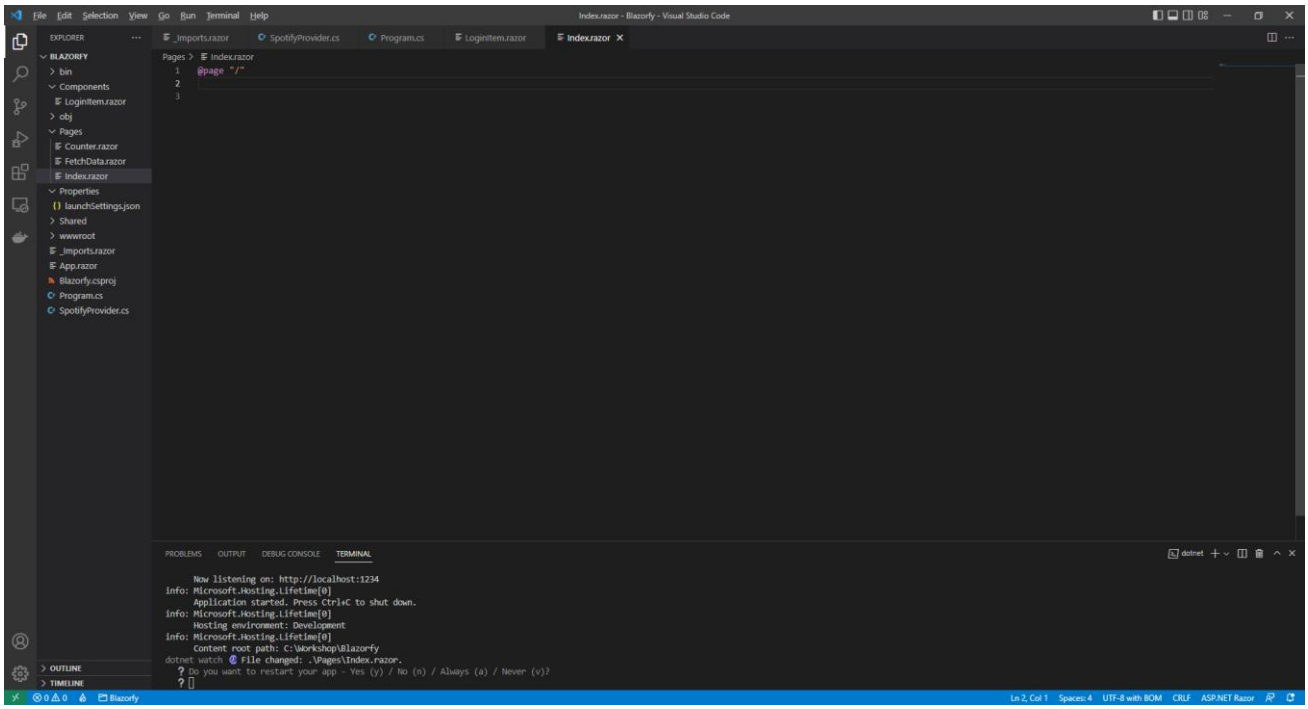
You might be wondering why **`Value`** is used here rather than checking if the **Account** is logged in directly, well we can take advantage of the fact when a value is passed into a **Component** and that value changes it will cause the **Component** to be updated rather than having to write that functionality ourselves as **Blazor** will automatically output a **Component** again should the value passed into it change.

Index

Next within **Visual Studio Code** from the **Explorer** for **Blazorfy** open **Pages** by selecting the **>** next to it and select **index.razor**, where you will see what is currently being displayed in the **Browser** as follows:



You will need to remove everything except the **`@page "/"`** at the top of the file so it appears as follows:



Within *Index.razor* in **Visual Studio Code** you can define the new **Page** by typing in below **`@page "/"`** the following which will also include some **Comments** to help you place some items later in the **Workshop**:

```
@inject SpotifyProvider _provider;
<LoginItem Value="@_provider.IsLoggedIn" />
@if (_provider.IsLoggedIn)
{
    // Items Output

}
@code
{
    [Parameter]
    [SupplyParameterFromQuery]
    public string? Code { get; set; } = null;

    // Items Property

    protected override async Task OnParametersSetAsync()
    {
        if (await _provider.HandleCodeAsync(Code))
        {
            // Items List

        }
    }
}
```

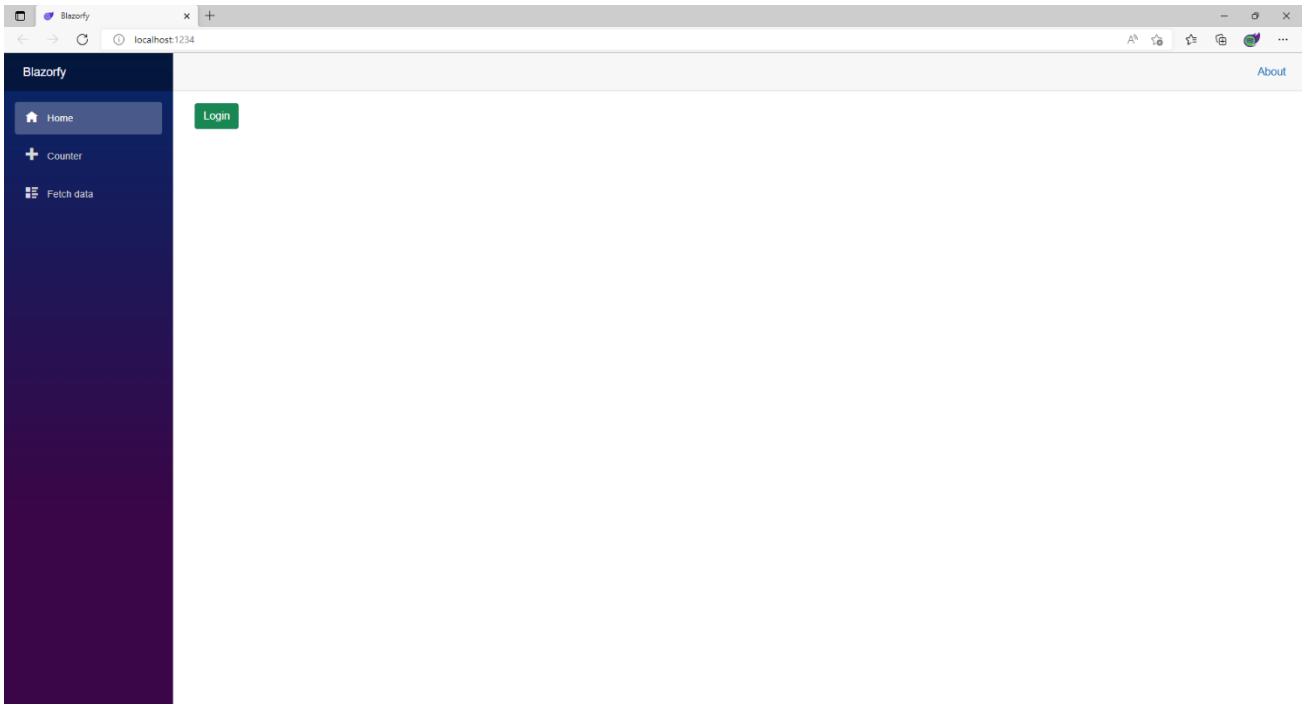
This **Page** now includes the same line to provide the **Instance** of the **`SpotifyProvider`** using **Dependency Injection** with **`inject`** as the **Component**. Then there is the **Component** with the **`Value`** being provided with the **Property** for **`IsLoggedIn`** from the **`class`**. This is then followed by the same **Property** being used in an **`if`** which will be used later in the **Workshop** to display some items.

There is also **Code** for the **Page** which includes a **Property** for the **Code** which will be provided to this page by **Spotify** after completing the login process this is set to be a **Parameter** with an **Attribute** which is within square brackets of **`[`** and **`]`** so that **Blazor** expects this to be set along with another **Attribute** to tell **Blazor** to get this value from the **Query String** which is part of the **`Uri`** returned from **Spotify**.

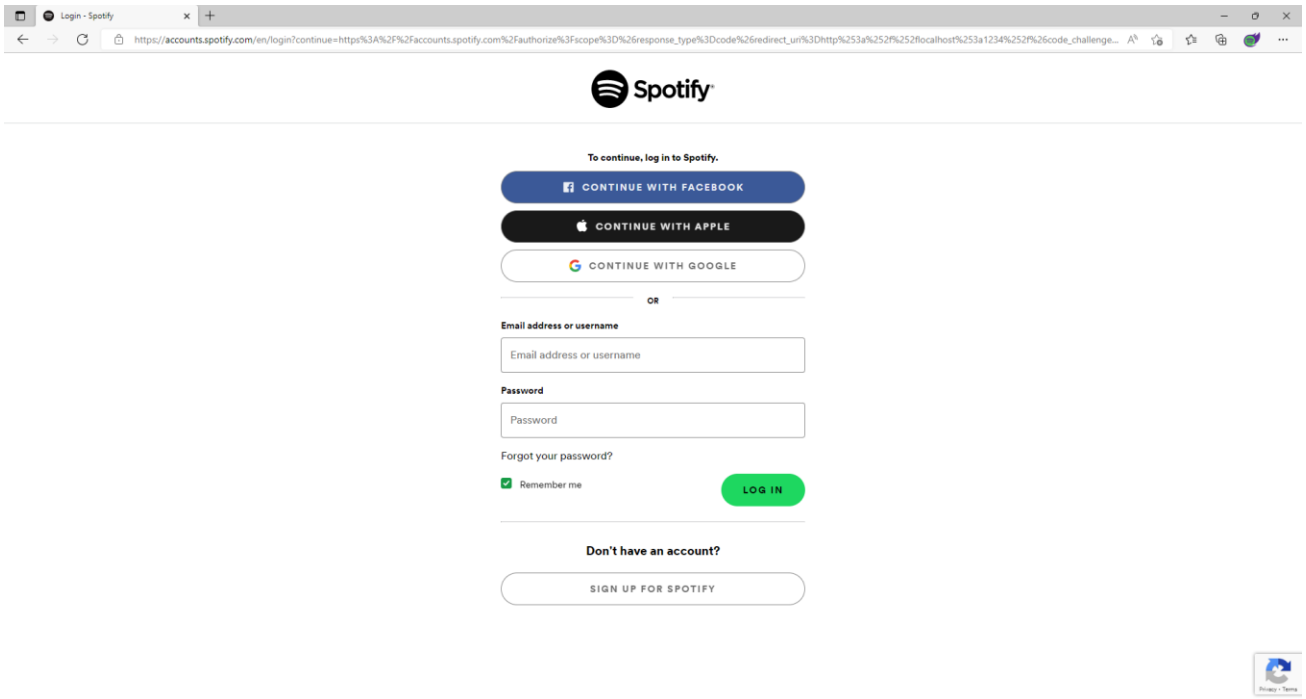
Then there is a special **Method** where the implementation of which has been overridden to provide our own denoted with **`override`** in this case it is for **`OnParametersSetAsync`** which is called when the **Properties** for the **Component** are set by **Blazor** and within this we will call the **Method** for **`HandleCodeAsync`** providing the **Code** that was obtained that will complete the logging process within the application.

You can then go to the **Menu** in **Visual Studio Code** and select **File** and then **Save All** you may see in the **Terminal** a message saying **Do you want to restart your app - Yes (y) / No (n) / Always (a) / Never (v)?** you can select the **Terminal** then type **y** for **Yes** or **a** for **Always** to keep what you have done so far.

If you return to the **Browser** you will see an option to **Login** as shown below:

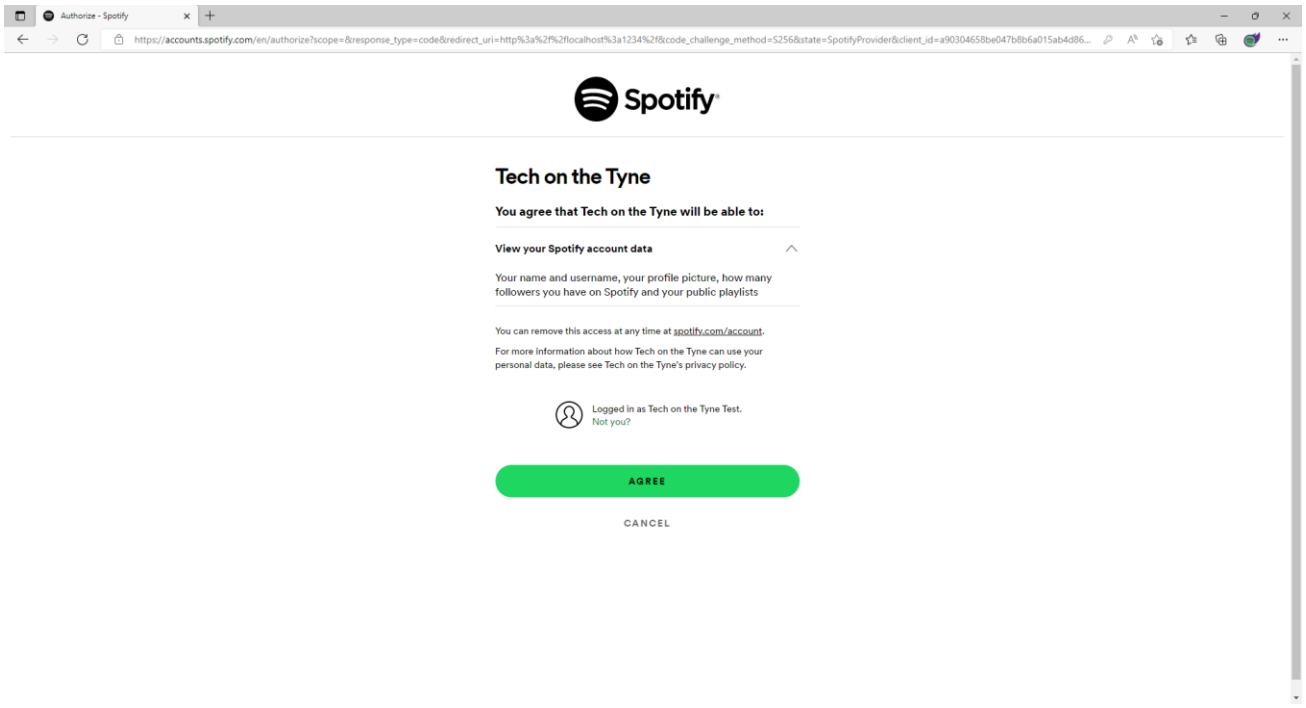


If you select the **Login** option you will be presented with something like the following where you should type in the **Email Address** for your **Account** and the **Password** and then select **Log In**



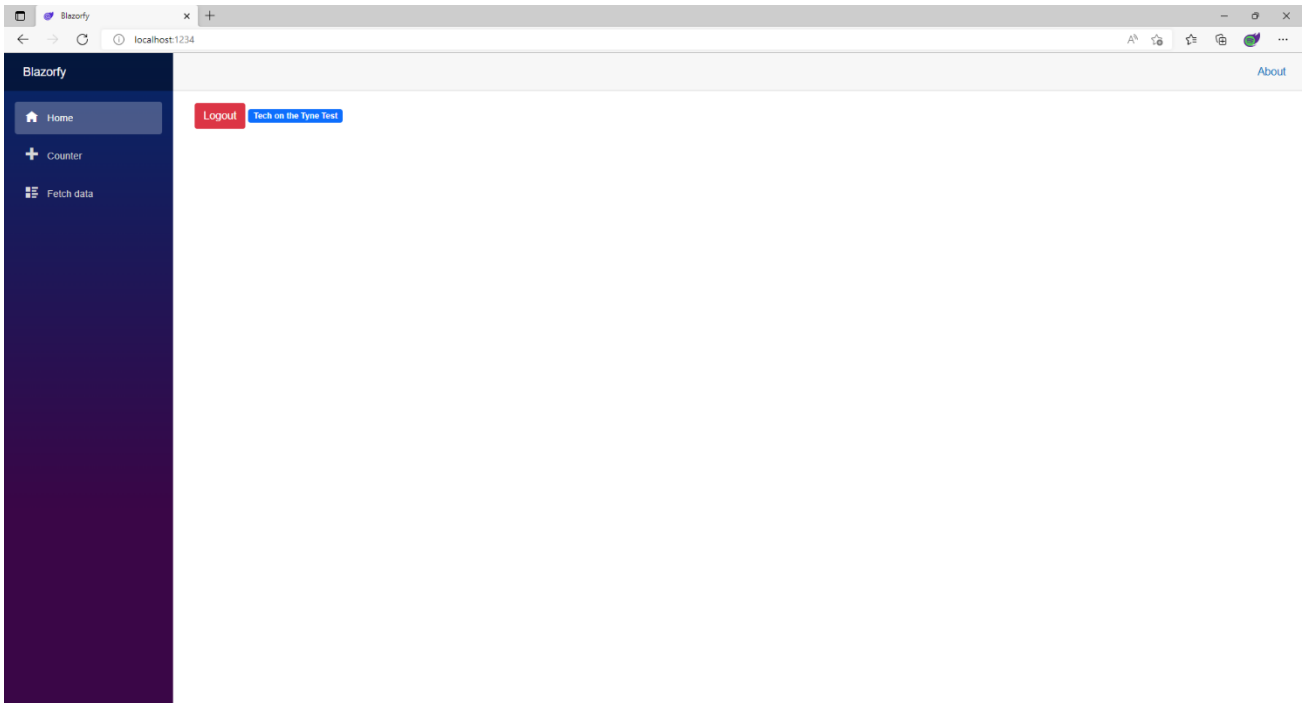
If for any reason you don't see this screen then go over the previous steps in the **Workshop** to make sure you haven't missed anything and also make sure you have an active **Internet** connection. If you do see this screen correctly then you can type in the **Email Address** from the list of **Accounts** for your number along with the **Password** and then select **Log In** you can leave the **Remember Me** option ticked so you can skip this step next time you try to login. If you get an **Error** at the bottom of the **Browser** and you don't see any mistakes then you can **Refresh** the **Browser** and that might fix the problem!

Once you have selected **Log In** you should see something like the following **Authorise** page displayed:



This will mention "**Blazorfy Workshop**" followed by your **Number** then you just need to select **Agree**.

Once you have selected **Agree** you will be redirected from **Spotify** back to **Blazorfy** and you should see the following which will display "**Tech on the Tyne**" followed by your **Number** next to the **Logout** option.



You can then select **Logout** which should Refresh the page and display **Login** and with that you have completed the **Authentication** part of the **Workshop**.

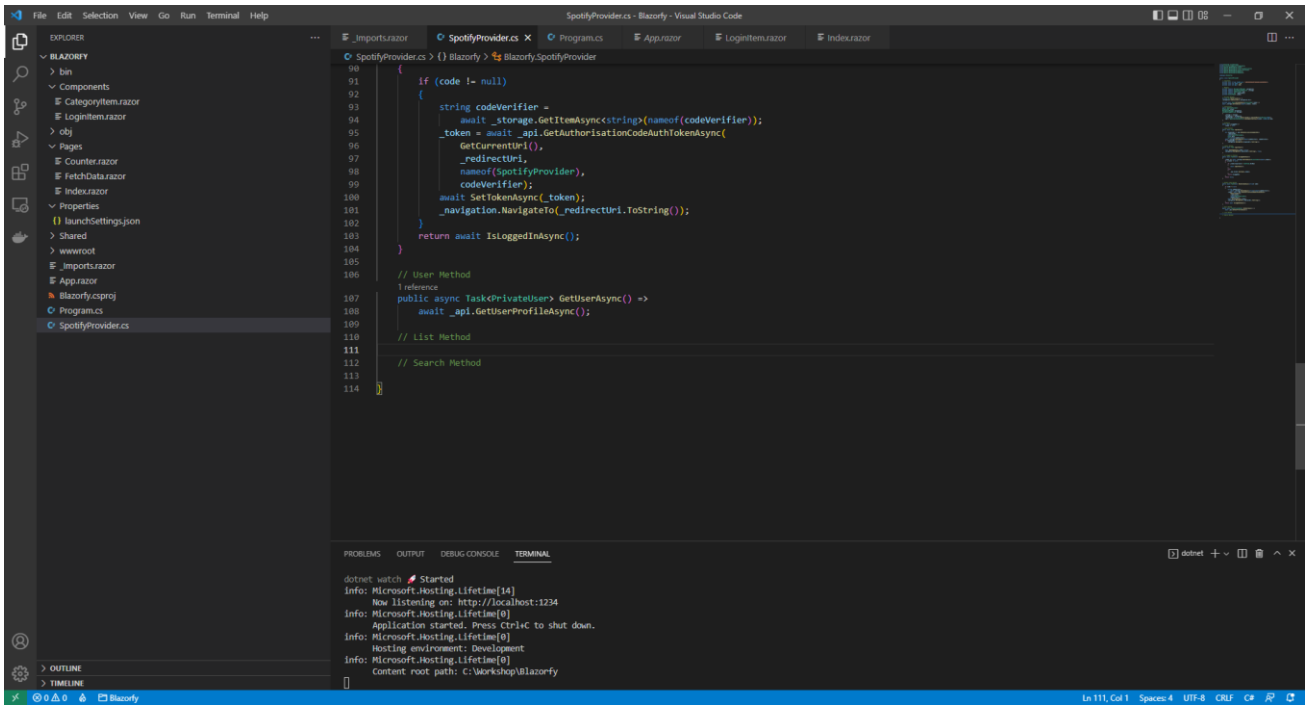
After the **Break** you will get a chance to add more **Pages** and some more functionality to the **Provider** to display information from the **Library** of content from **Spotify**!

Library

In this part of the **Workshop** you will learn how to get information from the **Library** of content from **Spotify** including **Categories**, **Playlists**, **Albums** and **Podcasts**. You'll build **Components** to display the details, add **Pages** for content from the **Library** and even create a **Component** that will allow you to see the same details using the **Spotify** application on your *iPhone* or *Android Phone* if you have **Spotify**!

Categories

Here we'll update the index page to show **Categories**, to do this return to **Visual Studio Code** for **Blazorfy** and then from **Explorer** select *SpotifyProvider.cs* as follows:



You should also still have open the **Browser** showing the Login option, if you don't then in **Visual Studio Code** if it was still open select the **Terminal** and then press **Ctrl+C** in **Windows** or **Command+C** on **Mac** on the **Keyboard**, or if **Visual Studio Code** was closed relaunch **Visual Studio Code** and then select the **New Terminal** then in the **Terminal**. With the **Terminal** in **Visual Studio Code** open type the following which should relaunch the **Browser**.

```
dotnet watch
```

Then in **Visual Studio Code** within *SpotifyProvider.cs* you will define part of a **Method** that will be used to display the **Categories**, so below the **Comment** of `// List Method` type the following **Method**:

```
public async Task<List<TItem>> ListAsync<TItem>(string? id = null)
where TItem : class
{
    var results = new List<TItem>();
    var page = new Page() { Limit = total };
    int count;
    do
    {
        Paging<TItem>? items = null;
        // Categories
        if (typeof(TItem) == typeof(Category))
        {
            items = await _api.GetAllCategoriesAsync(page: page)
            as Paging<TItem>;
        }
        // Playlists

        // Albums

        if (items != null)
        {
            results.AddRange(items.Items);
            page.Offset += total;
        }
        count = items?.Count ?? 0;
    }
    while (count > 0 && results.Count < max && count == total);
    return results;
}
```

This **Method** is a bit more complicated so feel free to **Copy** and **Paste** it into **Visual Studio Code** instead of typing it out. This **Method** uses a concept known as **Generics** which allows the **`type`** of a **`class`** to vary, when we had **`string`** and **`int`** before those were **`types`**. In this case there will be a **`List`** of a **`class`** which will be returned from this **Method**.

The first part of the **Method** uses the **Generic** syntax and also has a **`List`** of the items that will be returned. Then there is a **`Page`** that will be used to return the items up to the total that was defined earlier and there is a **Variable** of **`count`** to keep track of how many items have been retrieved.

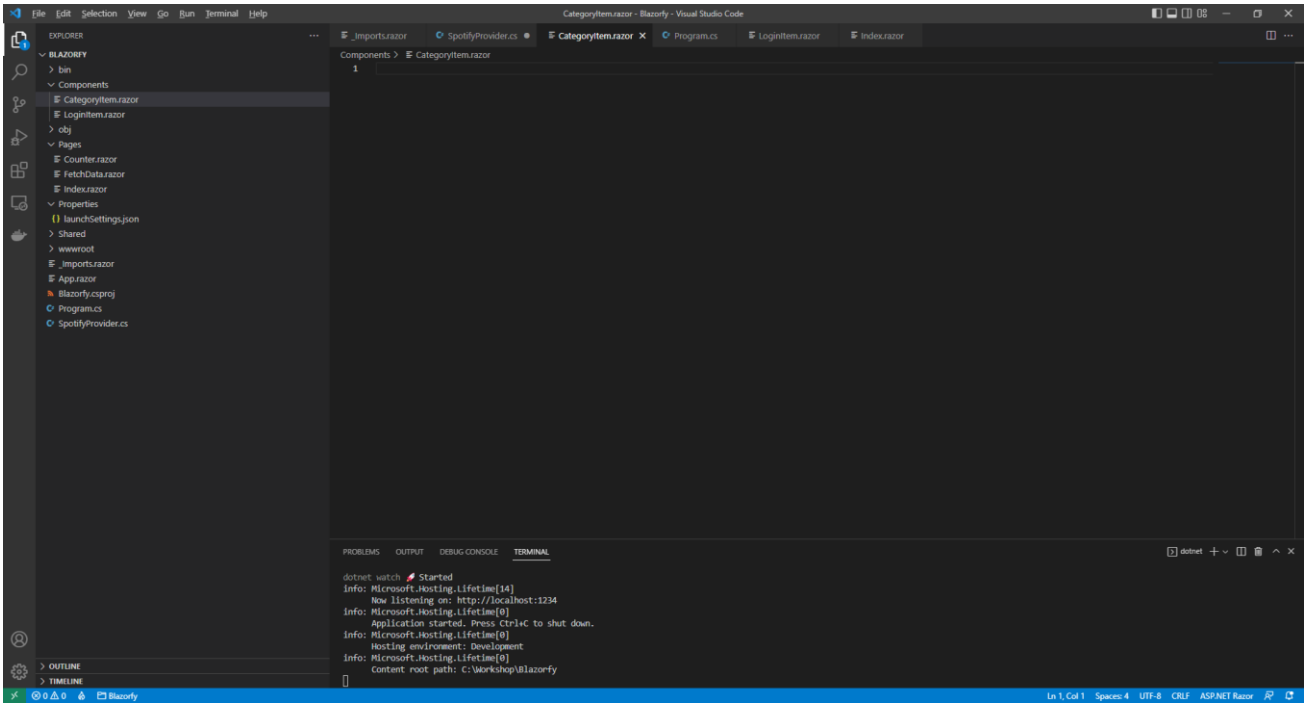
This method then has a **`do`** – **`while`** which will keep looping when there are a number of items with **`count`**, the number of total items is less than the maximum with **`max`** and there are still items.
The **Spotify.NetStandard** package uses a **`Paging`** object to contain anything returned from **Spotify**. When we provide the **`type`** of **`Category`** it will use the **Method** of **`GetAllCategoriesAsync`** which will be of **`Paging<Category>`** which is converted to **`Paging<TItem>`** but **`TItem`** will actually be **`Category`** in this case.

Then there's some logic to add what was obtained to results and go to the next **`Page`** where the loop will continue until any of the conditions being checked for are no longer satisfied. If you'd like to learn more about **Generics** then you can search for **.NET Generics** online.

Then while still in **Visual Studio Code** from **Explorer** select the **Folder** for **Components** then with the **Folder** for **Components** selected you should then select the **New File...** option and type in the name as follows then press **Enter**:

```
CategoryItem.razor
```

This will form the basis of another **Component** and will be a blank **Component** as follows:

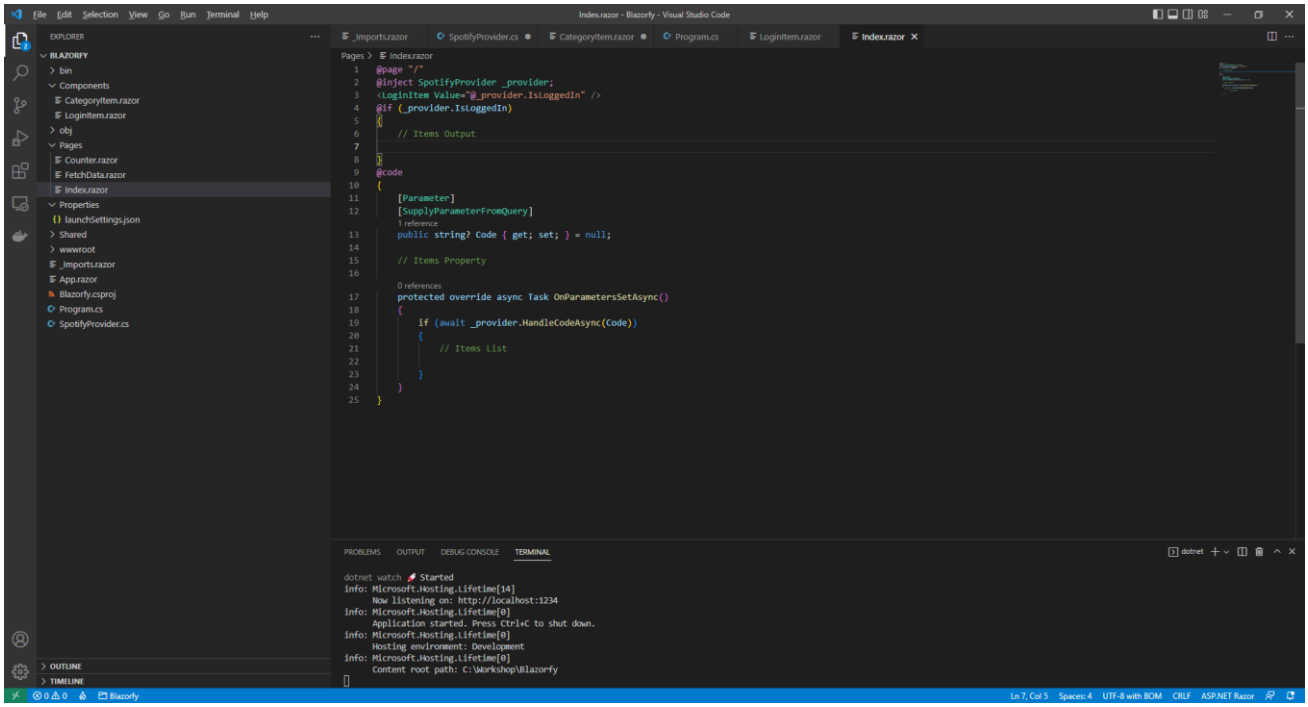


Within *CategoryItem.razor* in **Visual Studio Code** you can define this **Component** by typing in the following:

```
@namespace Blazorfy
<a href="playlists/@Value.Id">
    <div class="card">
        @if (Value.Images.Count > 0)
        {
            <img class="card-img-top"
            src="@Value.Images[0].Url" alt="@Value.Name" />
        }
        <div class="card-body">
            <h5 class="card-title">
                @Value.Name
            </h5>
        </div>
    </div>
</a>

@code
{
    [Parameter]
    public Category Value { get; set; } = new();
}
```

The first part of the **Component** is the **namespace** for the application which is **Blazorfy**. Then there is a link to a **Page** that will be created in the next part of the **Workshop** for **Playlists** which will pass through an **Id**.
Then there is some **HTML** to define the layout of the **Category Item** this includes a check to see if there are any **Images** with **if** that if **true** will then use the **img** to display the first image since **Images** is an **Array** you use the **[** and **]** to provide an **Index** in this case *0* to get it and we also set the **alt** of the **img** to the **Name** of the **Category** which is also displayed within a **h5** and the **Category** itself will be provided to the **Property** for **Value**.

Within **Visual Studio Code** from the **Explorer** for **Blazorfy** open **Pages** by selecting the **>** next to it and select **index.razor** as follows:



Within *Index.razor* in **Visual Studio Code** below the **Comment** of **`// Items Property`** type the following:

```
public List<Category> Items { get; set; } = new();
```

This **Property** will represent the **`List`** of **`Category`** that will be obtained from the **Provider**.

You can then go to the **Menu** in **Visual Studio Code** and select **File** and then **Save All** you may see in the **Terminal** a message saying **Do you want to restart your app - Yes (y) / No (n) / Always (a) / Never (v)?** you can select the **Terminal** then type **y** for **Yes** or **a** for **Always** to keep what you have done so far.

While still in *Index.razor* in **Visual Studio Code** below the **Comment** of **`// Items List`** type the following:

```
Items = await _provider.ListAsync<Category>();
```

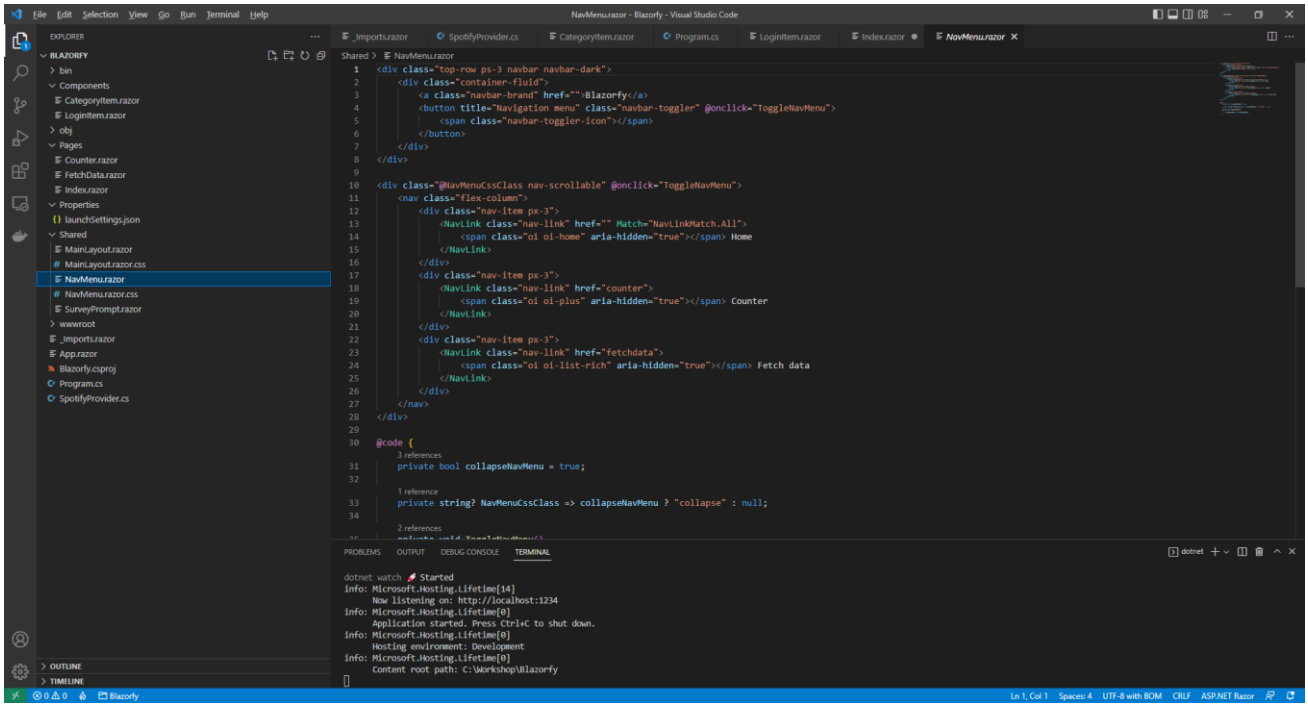
This will use the **Method** of **`ListAsync`** that was defined earlier in the **Provider** with the **`type`** of **`Category`**.

Then while still in *Index.razor* and below the **Comment** of **`// Items Output`** type the following:

```
<h1>Categories</h1>
<div class="container">
    <div class="row row-cols-1 row-cols-md-4 p-2 g-2">
        @foreach (var item in Items)
        {
            <div class="col">
                <CategoryItem Value="@item" />
            </div>
        }
    </div>
</div>
```

This will be used to output the **Categories** from the **Property** of **`Items`** and also uses the **Component** of **`CategoryItem`** to display them.

Then in **Visual Studio Code** from the **Explorer** for **Blazorfy** open **Shared** by selecting the **>** next to it in **Explorer** and select **NavMenu.razor** as follows:

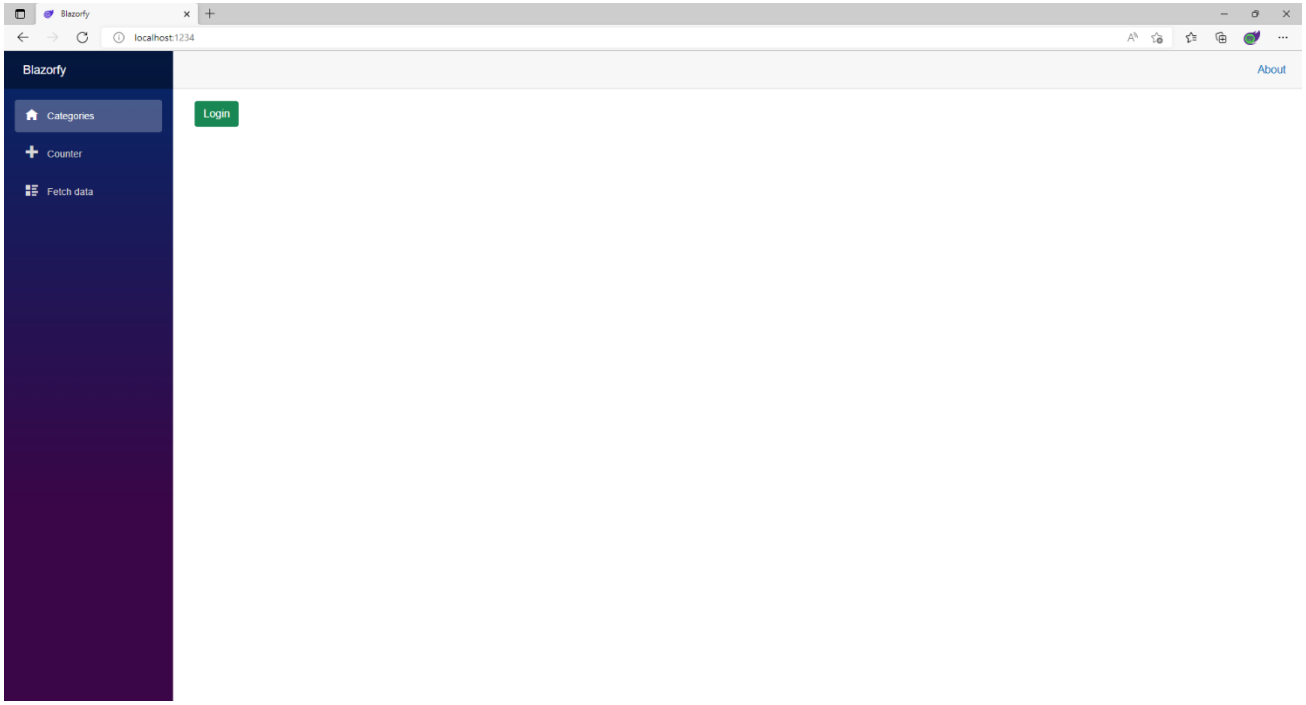


With *NavMenu.razor* selected where it says *Home* change this to say the following:

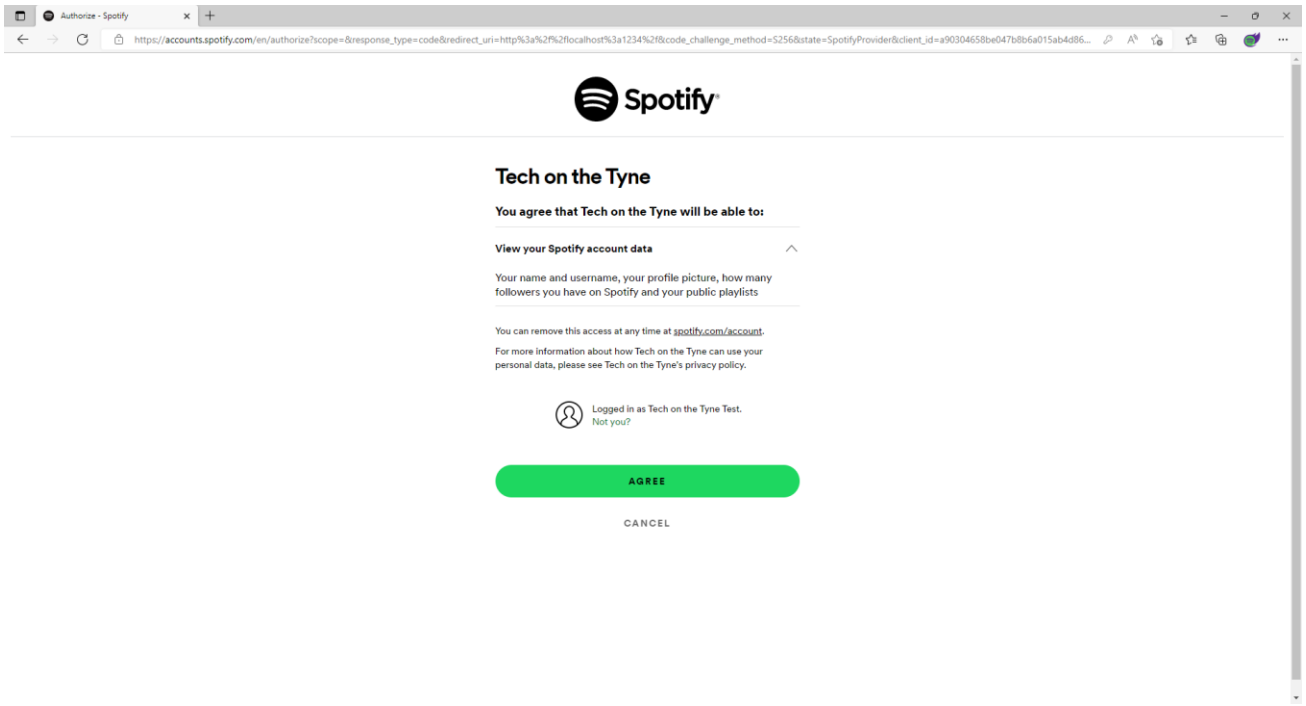
```
Categories
```

Then you can then go to the **Menu** in **Visual Studio Code** and select **File** and then **Save All** you may see in the **Terminal** a message saying **Do you want to restart your app - Yes (y) / No (n) / Always (a) / Never (v)?** you can select the **Terminal** then type **y** for **Yes** or **a** for **Always** to keep what you have done so far.

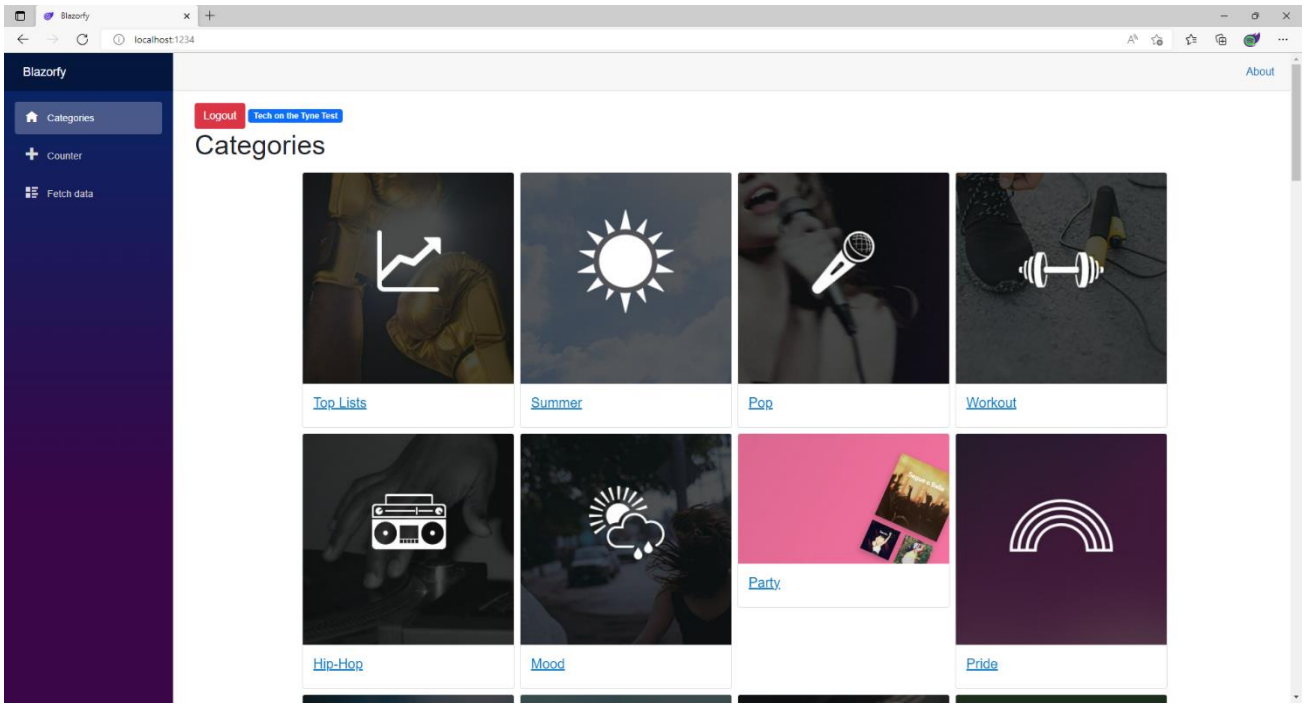
If you return to the **Browser** you will see an option to **Login** and the first option should be **Categories** as shown below:



Once you have selected **Login** you should see something like the following **Authorise** page displayed:



Once you have selected **Agree** you will be redirected from **Spotify** back to **Blazorfy** and you should see the following which will display "**Blazor Workshop**" followed by your **Number** next to the **Logout** option and then below this will be the **Categories** similar to as follows:



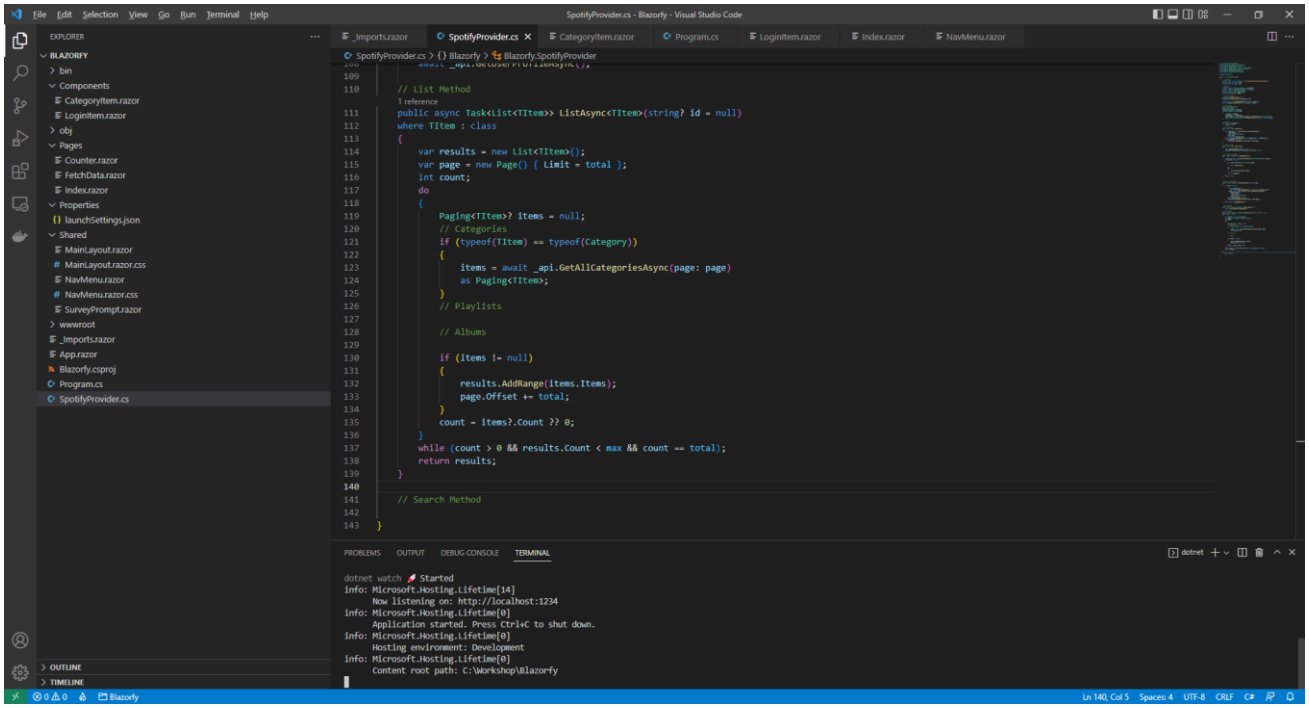
If you don't see this list of **Categories** then go through the previous steps in the **Workshop** and check that you've not missed anything, you can **Copy** and **Paste** anything you're not sure of from this if needed.

Each of the **Categories** has a link to a **Page** which will be created in the next part of the **Workshop** where you'll get to build the **Page** for **Playlists** along with a special **Component** that you can use with the **Spotify** mobile application for *iPhone* and *Android*.

You'll probably notice, if this is still the case that one of the images is not correct for **Party**, as part of **Spotify for Developers** there is a **Forum** where you can report issues and I've reported this incorrect asset to them, should this have been resolved then all the images should appear consistently.

Playlists

We'll add a new **Page** to show and **Search** for **Playlists** to do this return to **Visual Studio Code** for **Blazorfy** and then from **Explorer** select *SpotifyProvider.cs* as follows:



You should also still have open the **Browser** showing the Login option, if you don't then in **Visual Studio Code** if it was still open select the **Terminal** and then press **Ctrl+C** in **Windows** or **Command+C** on **Mac** on the **Keyboard**, or if **Visual Studio Code** was closed relaunch **Visual Studio Code** and then select the **New Terminal** then in the **Terminal**. With the **Terminal** in **Visual Studio Code** open type the following which should relaunch the **Browser**.

```
dotnet watch
```

Then in **Visual Studio Code** within *SpotifyProvider.cs* you will define part of a **Method** that will be used to display the **Categories**, so below the **Comment** of `// Search Method` type the following **Method**:

```
public async Task<List<TItem>> SearchAsync<TItem>(string query)
where TItem : class
{
    var results = new List<TItem>();
    var page = new Page() { Limit = total };
    int count;
    do
    {
        Paging<TItem>? items = null;
        var searchType = new SearchType()
        {
            Playlist = typeof(TItem) == typeof(SimplifiedPlaylist),
            Album = typeof(TItem) == typeof(Album),
            Show = typeof(TItem) == typeof(SimplifiedShow)
        };
        var content = await _api.SearchForItemAsync(query, searchType, page: page);
        // Playlists
        if (typeof(TItem) == typeof(SimplifiedPlaylist))
        {
            items = content.Playlists as Paging<TItem>;
        }
        // Albums

        // Podcasts

        if (items != null)
        {
            results.AddRange(items.Items);
            page.Offset += total;
        }
        count = items?.Count ?? 0;
    }
    while (count > 0 && results.Count < max && count == total);
    return results;
}
```

This **Method** is a bit more complicated so feel free to **Copy** and **Paste** it into **Visual Studio Code** instead of typing it out but you can still go through it to see if you can understand what is going on in the **Method**.

This **Method** also uses **Generics** and is very similar to **Method** for **`ListAsync`**. It has a few differences, this time it sets up a **`SearchType`** which has values that can be **`true`** or **`false`** to indicate the kind of **Search** and are using the **`type`** of the **Generic** to set this accordingly then using this with **`SearchForItemAsync`** and then for each **`type`** are getting the values for that.

There is also the use of **`typeof`** which is used to get the **`type`** of the **`class`** that is being used with the **Method**, this is useful as we can use this to define different behaviour depending on what **`type`** it is.

You'll have noticed in the previous **Method** and this one for the **`count`** there are **`?.`** and **`??`** being used, the **`?.`** is known as the *Elvis Operator* and this will treat the value of **`Count`** as **`null`** if the value of items is and then **`??`** can then be used to use *0* in place of the **`null`**.

While still within *SpotifyProvider.cs* in **Visual Studio Code** in the **Method** of **`ListAsync`** below the **Comment** of **`// Playlists`** type the following:

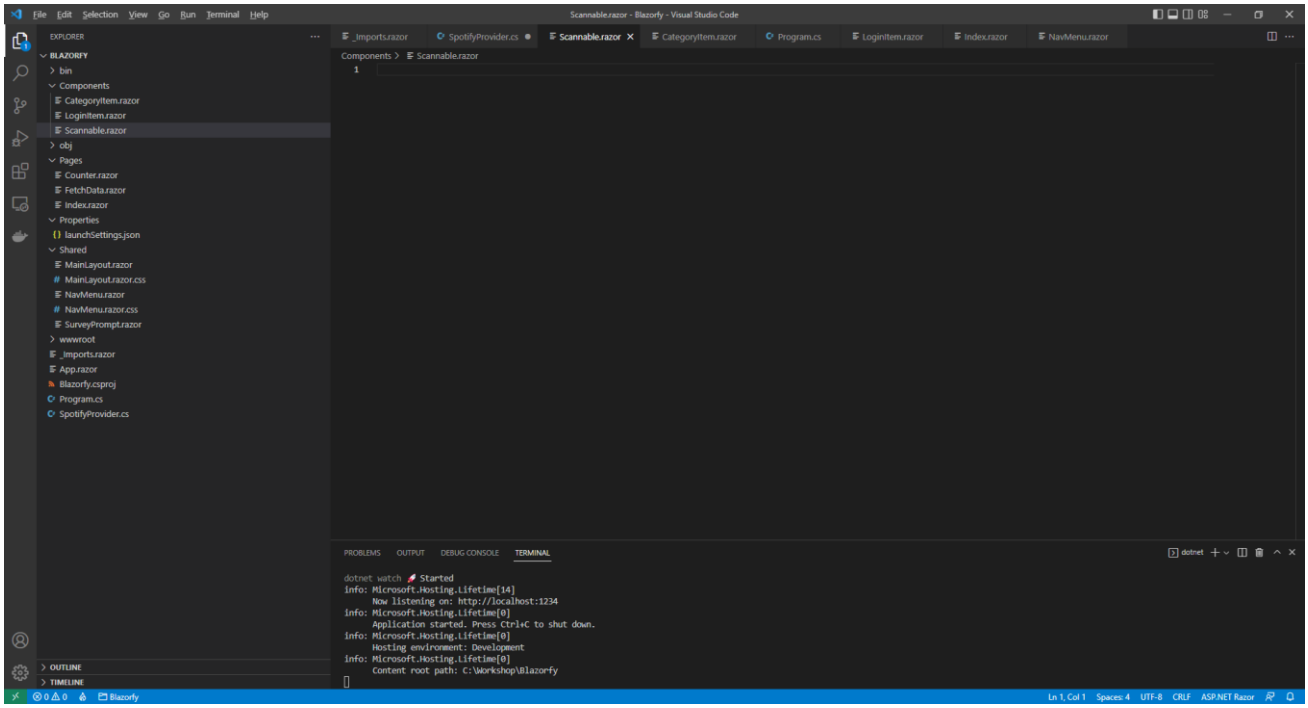
```
if (typeof(TItem) == typeof(SimplifiedPlaylist))
{
    items = await _api.GetCategoryPlaylistsAsync(id, page: page)
    as Paging<TItem>;
}
```

This part of the **Method** will use **`GetCategoryPlaylistsAsync`** to get the **Playlists** for a **Category**.

Then while still in **Visual Studio Code** from **Explorer** select the **Folder** for **Components** then with the **Folder** for **Components** selected you should then select the **New File...** option and type in the name as follows then press **Enter**:

```
Scannable.razor
```

This will form the basis of a shared **Component** and will be a blank **Component** as follows:



Within *Scannable.razor* in **Visual Studio Code** you can define the **Component** by typing in the following:

```
@namespace Blazorfy
<img class="img-fluid"
src="https://scannables.scdn.co/uri/plain/svg/5c2d91/white/640/@Value" />

@code
{
    [Parameter]
    public string Value { get; set; } = string.Empty;
}
```

This **Component** displays a special code that can be scanned using the **Spotify** application on *iPhone* or *Android* in an **img** and will use a passed in **Value** which forms part of **src**.

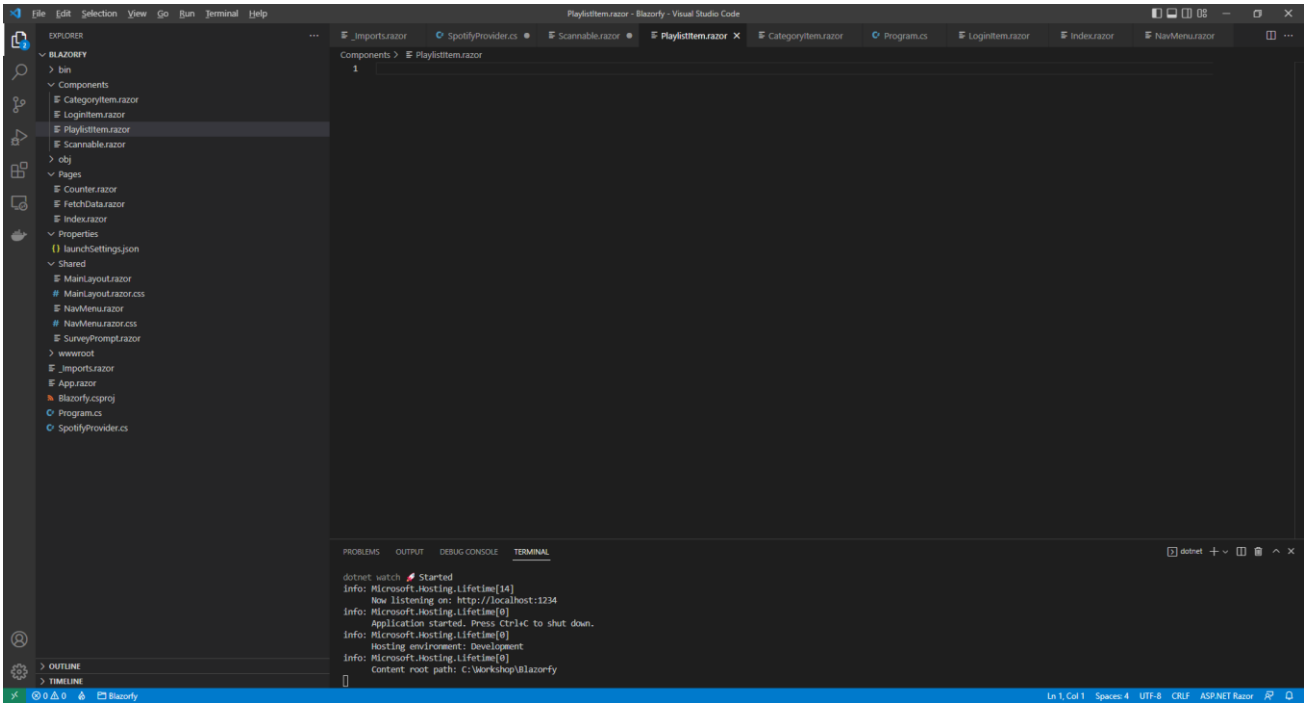
We can break down how the code or "scannable" itself is generated like so, the **svg** part of the **src** is the format of the image which can also be *png* or *jpeg* for those image formats.
The **5c2d91** part of the **src** is the background colour of the code and **white** is the foreground colour which can also be *black* then there is a number, in this case it is *640* which is the width of the image.
Finally the end part is the *URI* for a **Spotify** item such as a **Playlist** and other things you can share from **Spotify**, you can find out more about **Spotify Codes** at spotifycodes.com.

Then you can then go to the **Menu** in **Visual Studio Code** and select **File** and then **Save All** you may see in the **Terminal** a message saying **Do you want to restart your app - Yes (y) / No (n) / Always (a) / Never (v)?** you can select the **Terminal** then type **y** for **Yes** or **a** for **Always** to keep what you have done so far.

Then while still in **Visual Studio Code** from **Explorer** select the **Folder** for **Components** then with the **Folder** for **Components** selected you should then select the **New File...** option and type in the name as follows then press **Enter**:

```
PlaylistItem.razor
```

This will form the basis of another **Component** and will be a blank **Component** as follows:



Within *PlaylistItem.razor* in **Visual Studio Code** you can define the **Component** by typing in the following:

```
@namespace Blazorfy
<div class="card">
    @if (Value.Images.Count > 0)
    {
        <img src="@Value.Images[0].Url" class="card-img-top" alt="@Value.Name" />
    }
    <Scannable Value="@Value.Uri" />
    <div class="card-body">
        <h5 class="card-title">
            @Value.Name
        </h5>
    </div>
</div>

@code
{
    [Parameter]
    public SimplifiedPlaylist Value { get; set; } = new();
}
```

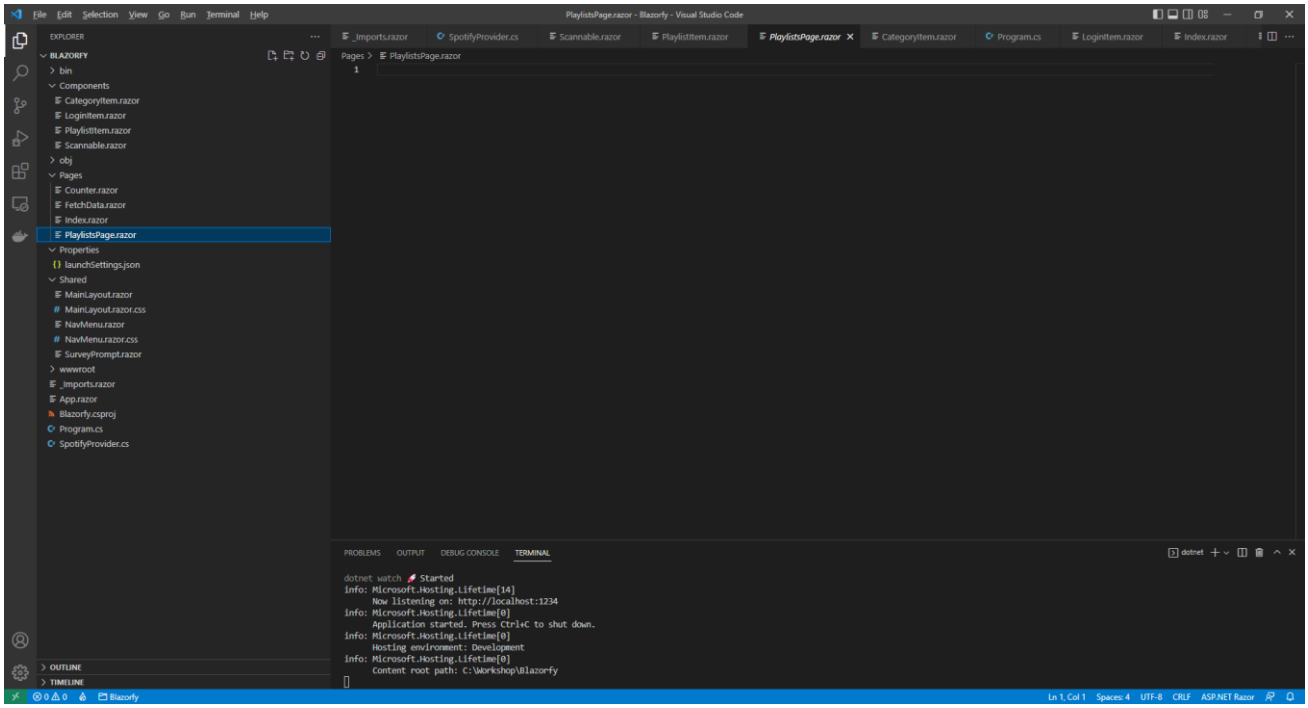
The first part of the **Component** is the **namespace** for the application which is **Blazorfy**. Then there is some **HTML** to define the layout of a **Playlist Item** this includes a check to see if there are any **Images** with **if** that if **true** will then use the **img** to display the first image since **Images** is an **Array** you use the **[** and **]** to provide an **Index** in this case *0* to get it and we also set the **alt** of the **img** to the **Name** of the **Playlist** which is also displayed within a **h5** and the **Playlist** itself will be provided to the **Property** for **Value**. You'll also see the inclusion of the **Component** for the **Scannable** which is provided with the **Uri** of the **Playlist**.

Then you can then go to the **Menu** in **Visual Studio Code** and select **File** and then **Save All** you may see in the **Terminal** a message saying **Do you want to restart your app - Yes (y) / No (n) / Always (a) / Never (v)?** you can select the **Terminal** then type **y** for **Yes** or **a** for **Always** to keep what you have done so far.

Then in **Visual Studio Code** from **Explorer** select the **Folder** for **Pages** then with the **Folder** for **Pages** selected you should then select the **New File...** option and type in the name as follows then press **Enter**:

```
PlaylistsPage.razor
```

This will form the basis of a **Page** and will be a blank **Page** as follows:



Within *PlaylistsPage.razor* in **Visual Studio Code** you can define the **Page** by typing in the following:

```
@page "/playlists/"
@page "/playlists/{id}"
@inject SpotifyProvider _provider;
<LoginItem Value="@_provider.IsLoggedIn" />
@if (_provider.IsLoggedIn)
{
    // Items Output

}

@code
{
    public List<SimplifiedPlaylist> Items { get; set; } = new();

    [Parameter]
    [SupplyParameterFromQuery]
    public string? Search { get; set; }

    [Parameter]
    public string? Id { get; set; }

    // Items Method

}
```

This **Page** includes two **page** directives which create the **Routes** needed to navigate to this **Page** this includes the one from the **Category Item** which will provide the **Id** and another which will be used from the **Menu** later.

There is also an **`inject`** to provide the **Instance** of the **`SpotifyProvider`** using **Dependency Injection**. Then there is the **Component** of **`LoginItem`** with the **`Value`** being provided with the **Property** for **`IsLoggedIn`** from the **`class`**.

There is also the **Property** for **`Items`** to be displayed in the **Page** along with a **Property** for the **Search** that will be provided, which will be from a query to the page from a **Form** which is denoted with the **Attribute** of **`SupplyParameterFromQuery`** or one for **Id** should this be provided as a **`Parameter`**.

While still within *PlaylistsPage.razor* in **Visual Studio Code** and below the **Comment** for **`// Items Method`** type the following **Method**:

```
protected async override Task OnParametersSetAsync()
{
    Items.Clear();
    if (await _provider.IsLoggedInAsync())
    {
        if (Search != null)
        {
            Items = await _provider.SearchAsync<SimplifiedPlaylist>(Search);
        }
        else
        {
            if (Id != null)
            {
                Items = await _provider.ListAsync<SimplifiedPlaylist>(Id);
            }
        }
    }
}
```

This is a special **Method** where the implementation of which has been overridden to provide our own denoted with **`override`** in this case it is for **`OnParametersSetAsync`**.
This will use **`Clear`** on **`Items`** to reset it and then will check if the user is logged in, then it checks to see if **`Search`** has a value other than **`null`**, if it does then it will use the **Method** of **`SearchAsync`** and provide the value to this, otherwise it will then check if the **`Id`** has a value, which it will should this be from the link from the **Category Item** which provides this to the **Page** and will use the **Method** of **`ListAsync`** instead.

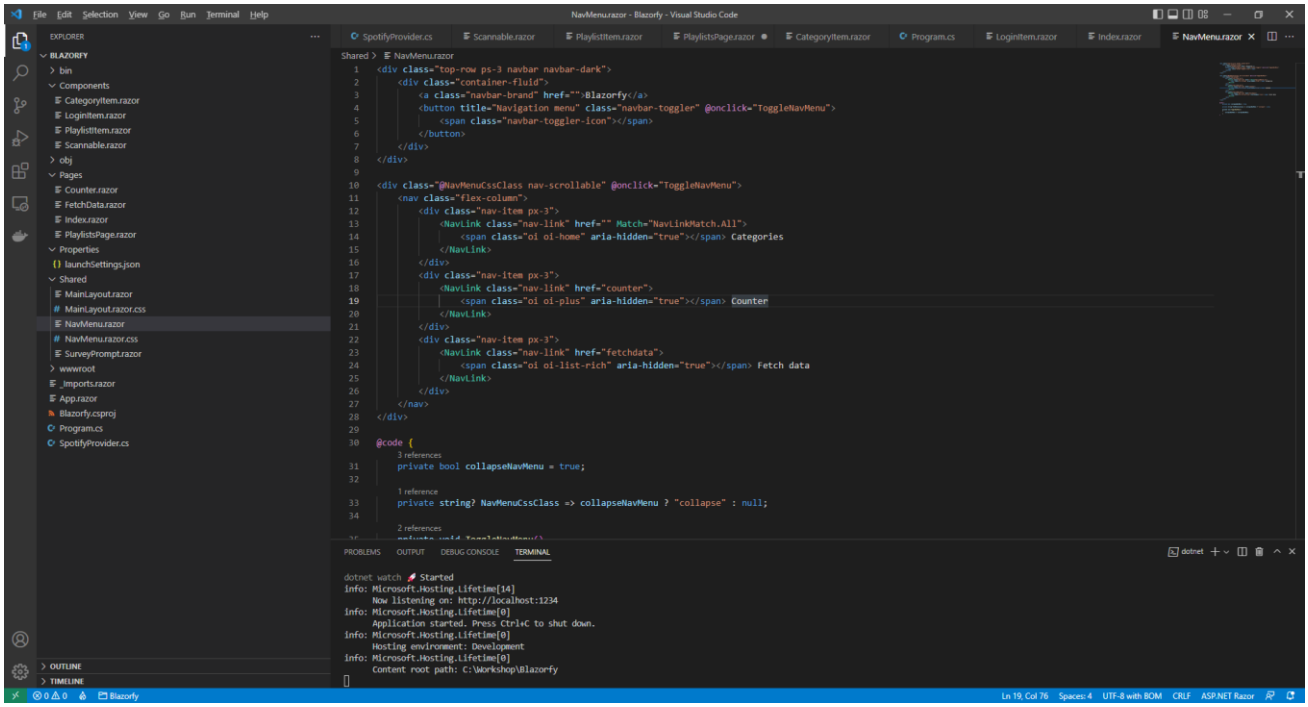
Finally while still within *PlaylistsPage.razor* in **Visual Studio Code** and below the **Comment** for **`// Items Output`** type the following:

```
@if (string.IsNullOrEmpty(Id))
{
    <h1>Search @Search</h1>
    <form @onsubmit="OnParametersSetAsync">
        <input type="text" @bind="Search" @bind:event="oninput" />
        <button class="btn btn-primary">Search</button>
    </form>
}
else
{
    <h1>Playlists</h1>
}
<div class="container">
    <div class="row row-cols-1 row-cols-md-4 p-2 g-2">
        @foreach (var item in Items)
        {
            <div class="col">
                <PlaylistItem Value="@item" />
            </div>
        }
    </div>
</div>
```

This defines how the **Page** will look, the first part will check if the **`Id`** is not present, that is not be **`null`** or an empty **`string`** then if this is the case then it will display a **`title`** along with **form** to perform a **Search** this has an **`input`** which is where the **Playlist** being looked for will be typed in and then there is a **`button`** to perform the **Search**.
Should the **`Id`** be present then it will display different **`title`**. Below these the **Component** for the **Playlist Item** will be used to display the **Playlists**.

Then you can then go to the **Menu** in **Visual Studio Code** and select **File** and then **Save All** you may see in the **Terminal** a message saying **Do you want to restart your app - Yes (y) / No (n) / Always (a) / Never (v)?** you can select the **Terminal** then type **y** for **Yes** or **a** for **Always** to keep what you have done so far.

Then in **Visual Studio Code** from the **Explorer** for **Blazorfy** open **Shared** by selecting the **>** next to it in **Explorer** and select **NavMenu.razor** as follows:



With *NavMenu.razor* selected there will be a section for *Counter* as follows:

```
<div class="nav-item px-3">
    <NavLink class="nav-link" href="counter">
        <span class="oi oi-plus" aria-hidden="true"></span> Counter
    </NavLink>
</div>
```

Change where it says *counter* in **href** to be as follows:

```
playlists
```

Then you will need to change where it says *oi-plus* in the **span** to be as follows:

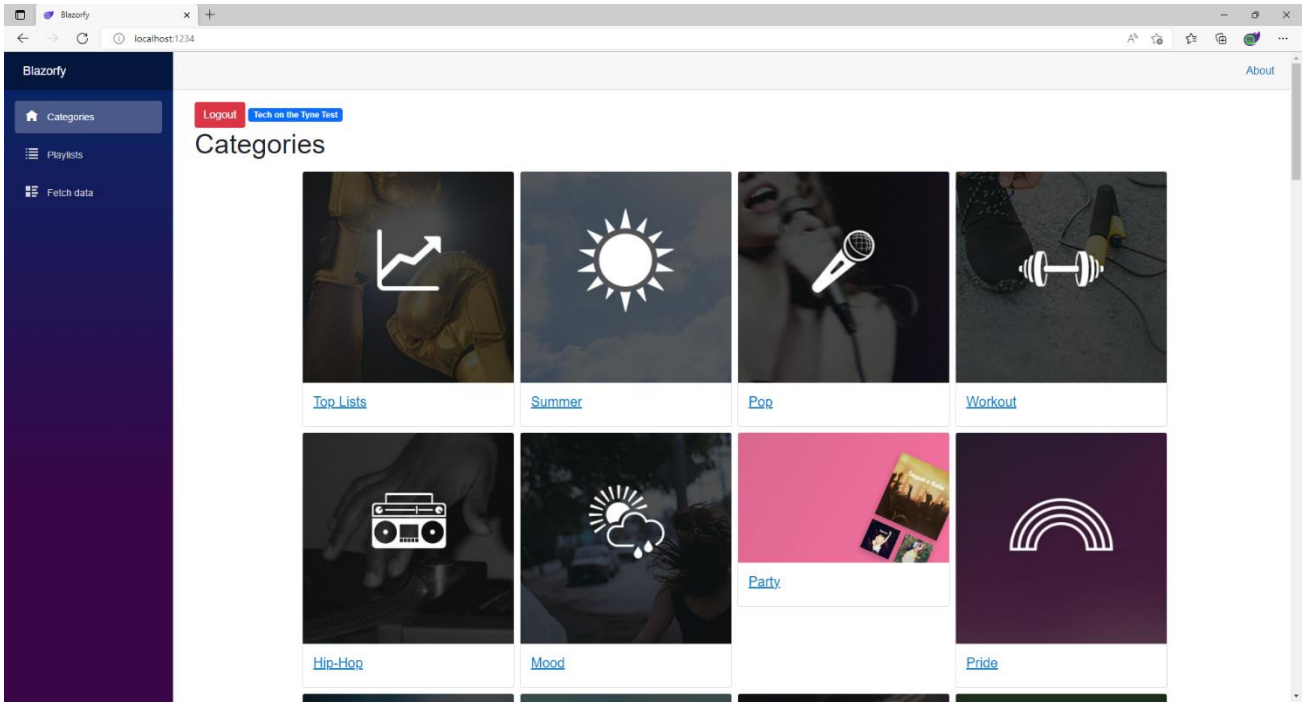
```
oi-list
```

Next you will need to change where it says *Counter* to be as follows:

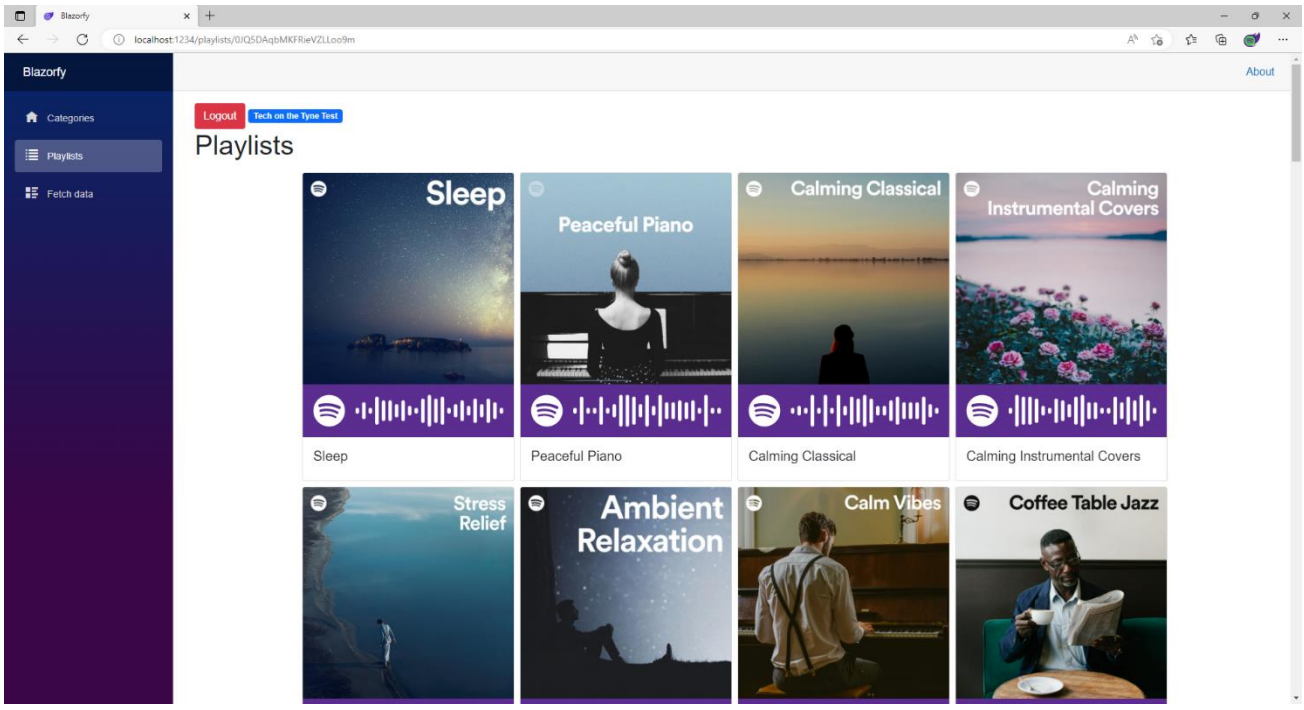
```
Playlists
```

Then you can then go to the **Menu** in **Visual Studio Code** and select **File** and then **Save All** you may see in the **Terminal** a message saying **Do you want to restart your app - Yes (y) / No (n) / Always (a) / Never (v)?** you can select the **Terminal** then type **y** for **Yes** or **a** for **Always** to keep what you have done so far.

If you return to the **Browser** you will see the **Categories** along with a link to **Playlists** as shown below:



You can then select one of the **Categories** and you should see a list of **Playlists** similar to as follows:

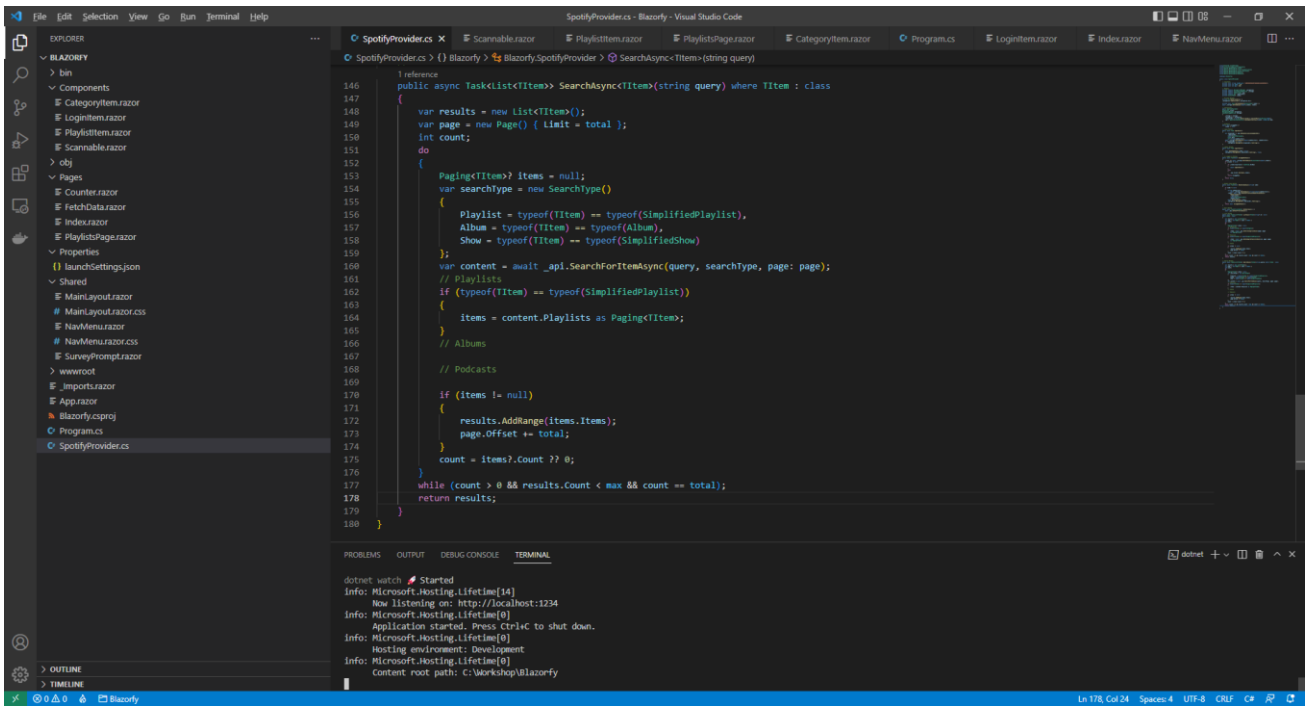


You can also select the **Playlists** option from the **Menu** and then type in your favourite **Artist** or anything else and then select **Search** to find related **Playlists**. You'll also notice underneath each **Playlist** is a **Spotify Code**, if you have **Spotify** on your *iPhone* or *Android* device you can use the option to scan these using the app and can then checkout the **Playlist** for yourself.

If you don't see anything then check that you've completed each part correctly and double-check that what you have is the same, if everything is working then you can proceed to the next part of the **Workshop**.

Albums

We'll add a new **Page** to show and **Search** for **Albums** to do this return to **Visual Studio Code** for **Blazorfy** and then from **Explorer** select *SpotifyProvider.cs* as follows:



You should also still have open the **Browser** showing the Login option, if you don't then in **Visual Studio Code** if it was still open select the **Terminal** and then press **Ctrl+C** in **Windows** or **Command+C** on **Mac** on the **Keyboard**, or if **Visual Studio Code** was closed relaunch **Visual Studio Code** and then select the **New Terminal** then in the **Terminal**. With the **Terminal** in **Visual Studio Code** open type the following which should relaunch the **Browser**.

```
dotnet watch
```

Then in **Visual Studio Code** within *SpotifyProvider.cs* you will define part of the **Method** that will be used to get **New Releases** of **Albums**. In the **Method** of **`ListAsync`** and below the **Comment** of **`// Albums`** type the following:

```
if (typeof(TItem) == typeof(Album))
{
    items = await _api.GetAllNewReleasesAsync(page: page)
    as Paging<TItem>;
}
```

This part of the **Method** will be used to get the **New Releases** from **Spotify** when the **`type`** is **`Album`**.

While still within *SpotifyProvider.cs* you will define part of the **Method** that will be used to **Search** for **Albums**. In the **Method** of **`SearchAsync`** and below the **Comment** of **`// Albums`** type the following:

```
if (typeof(TItem) == typeof(Album))
{
    items = content.Albums as Paging<TItem>;
}
```

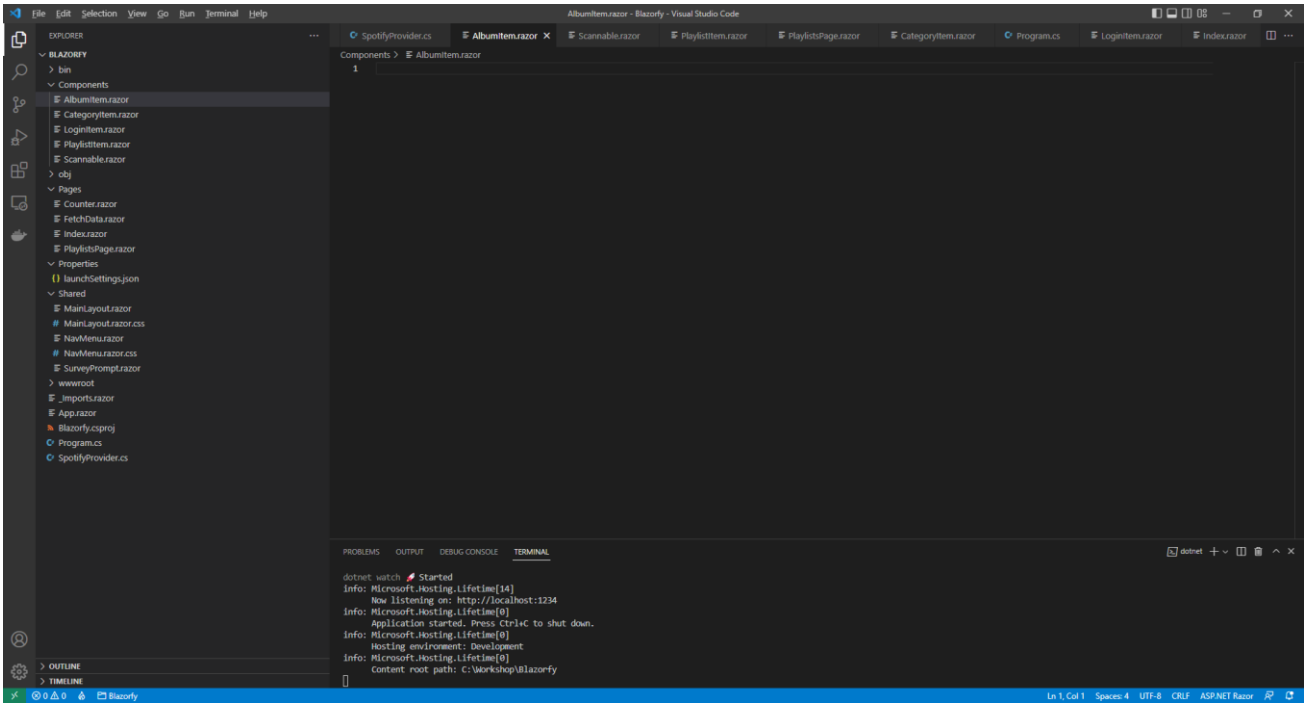
This will get the **`Albums`** that are returned when the **`SearchType`** is **Album**, which is determined in **`SearchAsync`** by checking if the **`type`** that is used is an **`Album`** and setting the value for this accordingly.

Then you can then go to the **Menu** in **Visual Studio Code** and select **File** and then **Save All** you may see in the **Terminal** a message saying **Do you want to restart your app - Yes (y) / No (n) / Always (a) / Never (v)?** you can select the **Terminal** then type **y** for **Yes** or **a** for **Always** to keep what you have done so far.

Then while still in **Visual Studio Code** from **Explorer** select the **Folder** for **Components** then with the **Folder** for **Components** selected you should then select the **New File...** option and type in the name as follows then press **Enter**:

```
AlbumItem.razor
```

This will form the basis of another **Component** and will be a blank **Component** as follows:



Within *AlbumItem.razor* in **Visual Studio Code** you can define the **Component** by typing in the following:

```
@namespace Blazorfy
<div class="card">
    @if (Value.Images.Count > 0)
    {
        <img src="@Value.Images[0].Url" class="card-img-top" alt="@Value.Name" />
    }
    <Scannable Value="@Value.Uri" />
    <div class="card-body">
        <h5 class="card-title">
            @Value.Name
        </h5>
    </div>
</div>

@code
{
    [Parameter]
    public SimplifiedAlbum Value { get; set; } = new();
}
```

The first part of the **Component** is the **namespace** for the application which is **Blazorfy**. Then there is some **HTML** to define the layout of an **Album Item** this includes a check to see if there are any **Images** with **if** that if **true** will then use the **img** to display the first image since **Images** is an **Array** you use the **[** and **]** to provide an **Index** in this case *0* to get it and we also set the **alt** of the **img** to the **Name** of the **Album** which is also displayed within a **h5** and the **Album** itself will be provided to the **Property** for **Value**.

You'll also see the inclusion of the **Component** for the **Scannable** which is provided with the **Uri** of the **Album**.

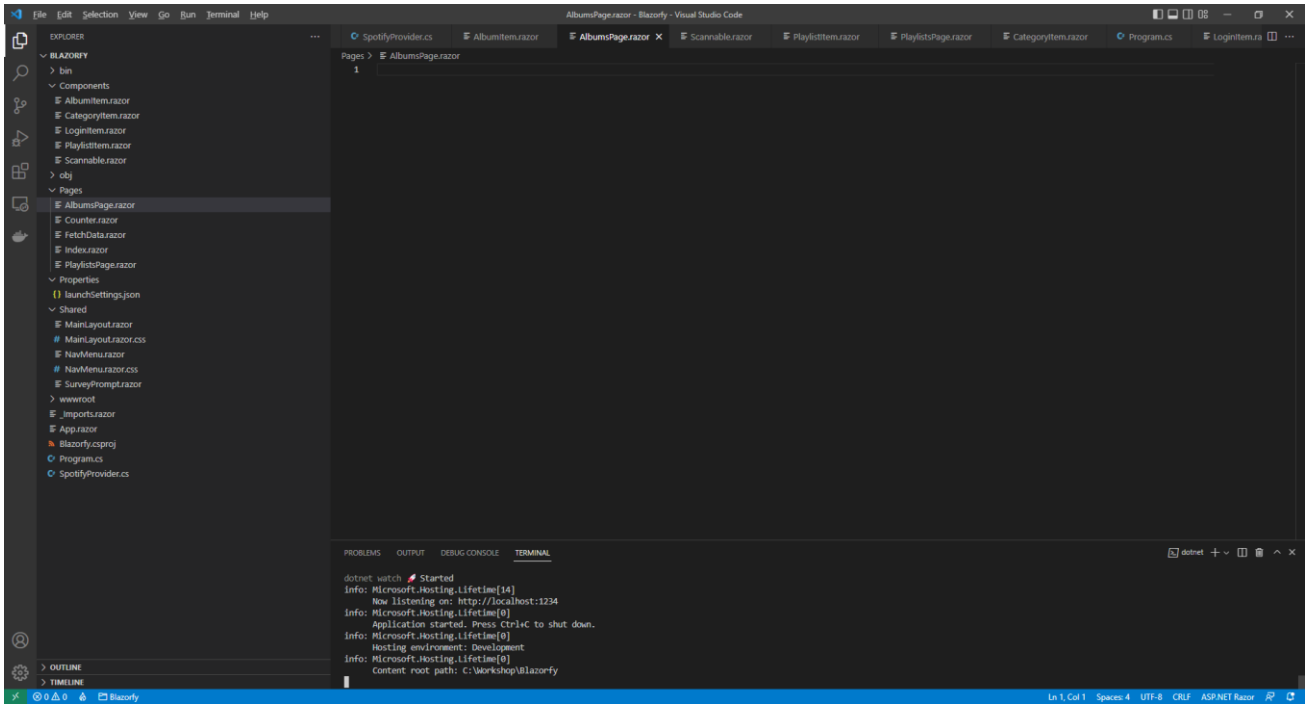
This **Component** is identical to the one for **Playlist** but you could **Optionally** include different information such as the **Artist** for the **Album** as **SimplifiedAlbum** has a **Property** for a **List** of **Artists** that could be used for this purpose that you could add to this **Component**.

Then you can then go to the **Menu** in **Visual Studio Code** and select **File** and then **Save All** you may see in the **Terminal** a message saying **Do you want to restart your app - Yes (y) / No (n) / Always (a) / Never (v)?** you can select the **Terminal** then type **y** for **Yes** or **a** for **Always** to keep what you have done so far.

Then in **Visual Studio Code** from **Explorer** select the **Folder** for **Pages** then with the **Folder** for **Pages** selected you should then select the **New File...** option and type in the name as follows then press **Enter**:

```
AlbumsPage.razor
```

This will form the basis of a **Page** and will be a blank **Page** as follows:



Within *AlbumsPage.razor* in **Visual Studio Code** you can define the **Page** by typing in the following:

```
@page "/albums"
@inject SpotifyProvider _provider;
<LoginItem Value="@_provider.IsLoggedIn" />
@if (_provider.IsLoggedIn)
{
    // Items Output

}

@code
{
    public List<Album> Items { get; set; } = new();

    [Parameter]
    [SupplyParameterFromQuery]
    public string? Search { get; set; }

    // Items Method

}
```

This **Page** includes a **page** directive which create a **Route** needed to navigate to this **Page** which will be used from the **Menu** later.

There is also an **`inject`** to provide the **Instance** of the **`SpotifyProvider`** using **Dependency Injection** and then there is the **Component** of **`LoginItem`** with the **`Value`** being provided with the **Property** for **`IsLoggedIn`** from the **`class`**.

There is also the **Property** for **`Items`** to be displayed in the **Page** along with a **Property** for the **Search** that will be provided, which will be from a query to the page from a **Form** which is denoted with the **Attribute** of **`SupplyParameterFromQuery`**.

While still within *PlaylistsPage.razor* in **Visual Studio Code** and below the **Comment** for **`// Items Method`** type the following **Method**:

```
protected override async Task OnParametersSetAsync()
{
    Items.Clear();
    if (await _provider.IsLoggedInAsync())
    {
        if (Search != null)
        {
            Items = await _provider.SearchAsync<Album>(Search);
        }
        else
        {
            Items = await _provider.ListAsync<Album>();
        }
    }
}
```

This is a special **Method** where the implementation of which has been overridden to provide our own denoted with **`override`** in this case it is for **`OnParametersSetAsync`**.
This will use **`Clear`** on **`Items`** to reset it and then will check if the user is logged in, then it checks to see if **`Search`** has a value other than **`null`**, if it does then it will use the **Method** of **`SearchAsync`** and provide the value to this with the **`type`** of **`Album`** otherwise it will use the **Method** of **`ListAsync`** with the **`type`** of **`Album`**.

Finally while still within *AlbumsPage.razor* in **Visual Studio Code** and below the **Comment** for **`// Items Output`** type the following:

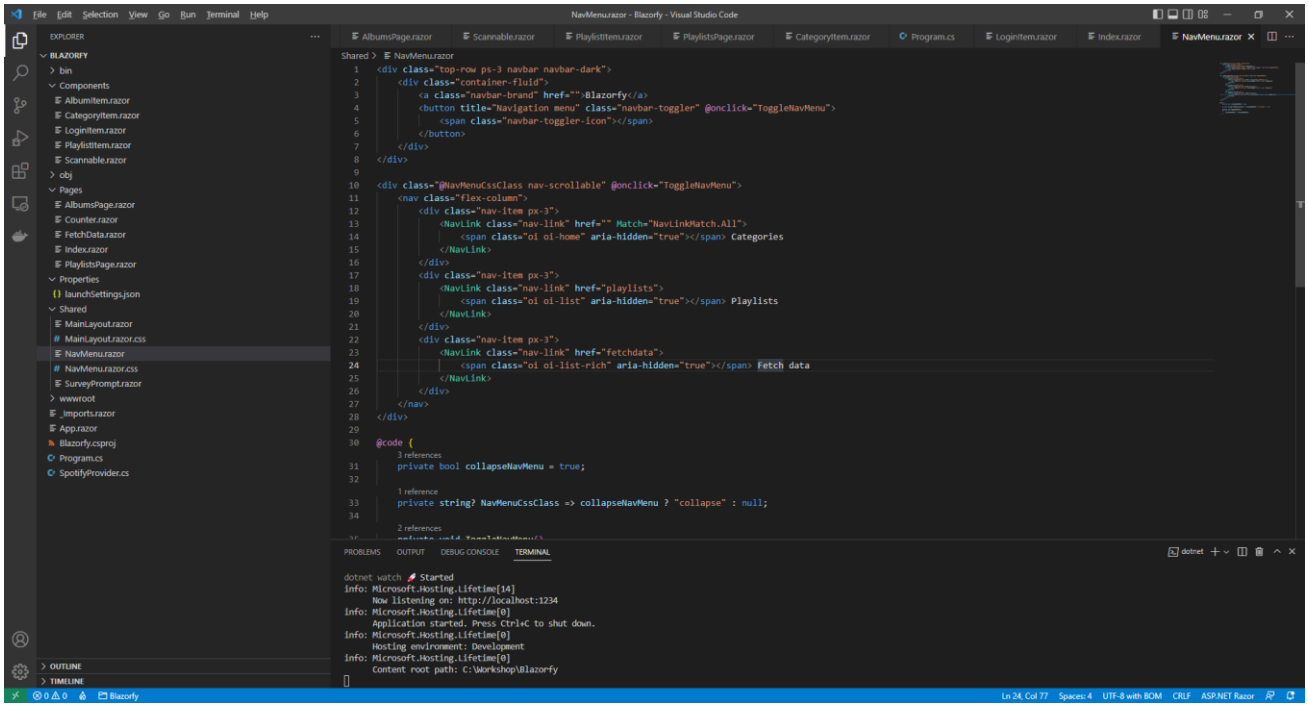
```
@if (string.IsNullOrWhiteSpace(Search))
{
    <h1>New Releases</h1>
}
else
{
    <h1>Search @Search</h1>
}
<form @onsubmit="OnParametersSetAsync">
    <input type="text" @bind="Search" @bind:event="oninput" />
    <button class="btn btn-primary">Search</button>
</form>
<div class="container">
    <div class="row row-cols-1 row-cols-md-4 p-2 g-2">
        @foreach (var item in Items)
        {
            <div class="col">
                <AlbumItem Value="@item" />
            </div>
        }
    </div>
</div>
```

This defines how the **Page** will look, the first part will check if the **`Search`** is not be **`null`** or an empty **`string`** then if this is the case then it will display a **`title`** as *New* Releases or as *Search*.

Then there is a **form** to perform a **Search** this has an **`input`** which is where the **Album** being looked for will be typed in and then there is a **`button`** to perform the **Search** and below this the **Component** for the **Album Item** will be used to display the **Albums**.

Then you can then go to the **Menu** in **Visual Studio Code** and select **File** and then **Save All** you may see in the **Terminal** a message saying **Do you want to restart your app - Yes (y) / No (n) / Always (a) / Never (v)?** you can select the **Terminal** then type **y** for **Yes** or **a** for **Always** to keep what you have done so far.

Then in **Visual Studio Code** from the **Explorer** for **Blazorfy** open **Shared** by selecting the **>** next to it in **Explorer** and select **NavMenu.razor** as follows:



With *NavMenu.razor* selected there will be a section for *Fetch Data* as follows:

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

Change where it says *fetchdata* in **href** to be as follows:

```
albums
```

Then you will need to change where it says *oi-list-rich* in the **span** to be as follows:

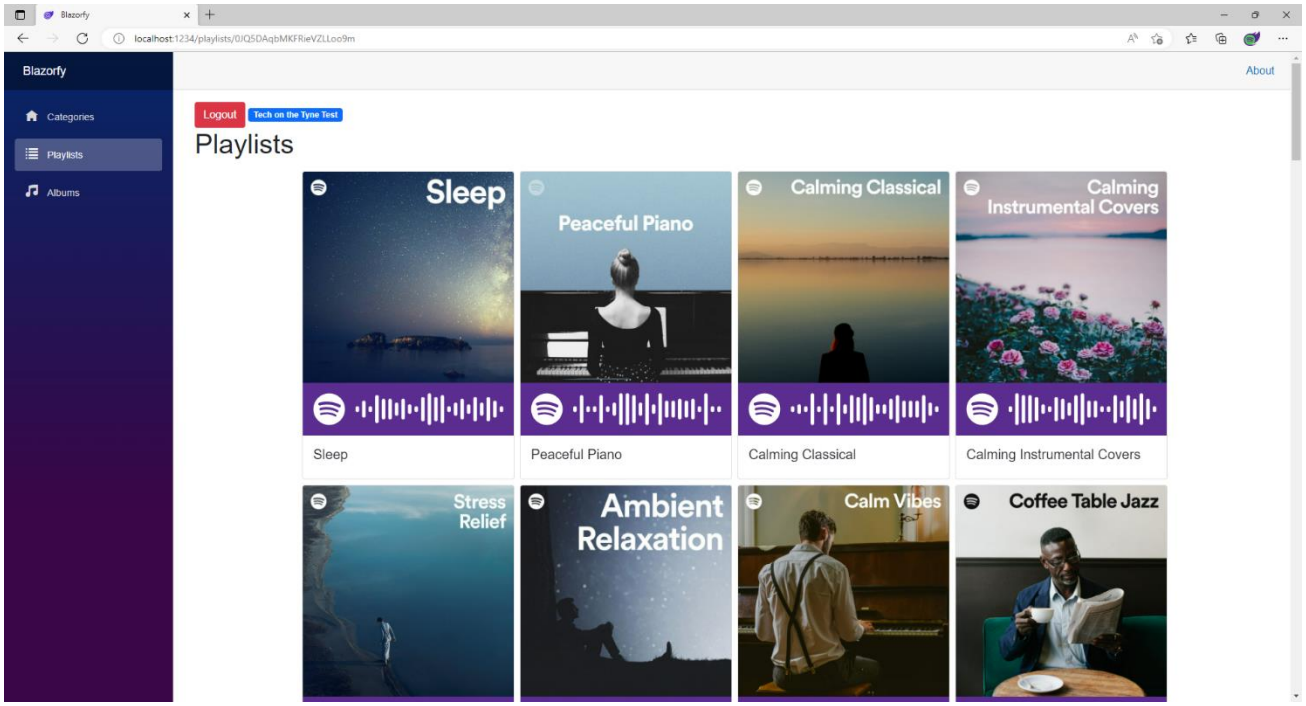
```
oi-musical-note
```

Next you will need to change where it says *Fetch Data* to be as follows:

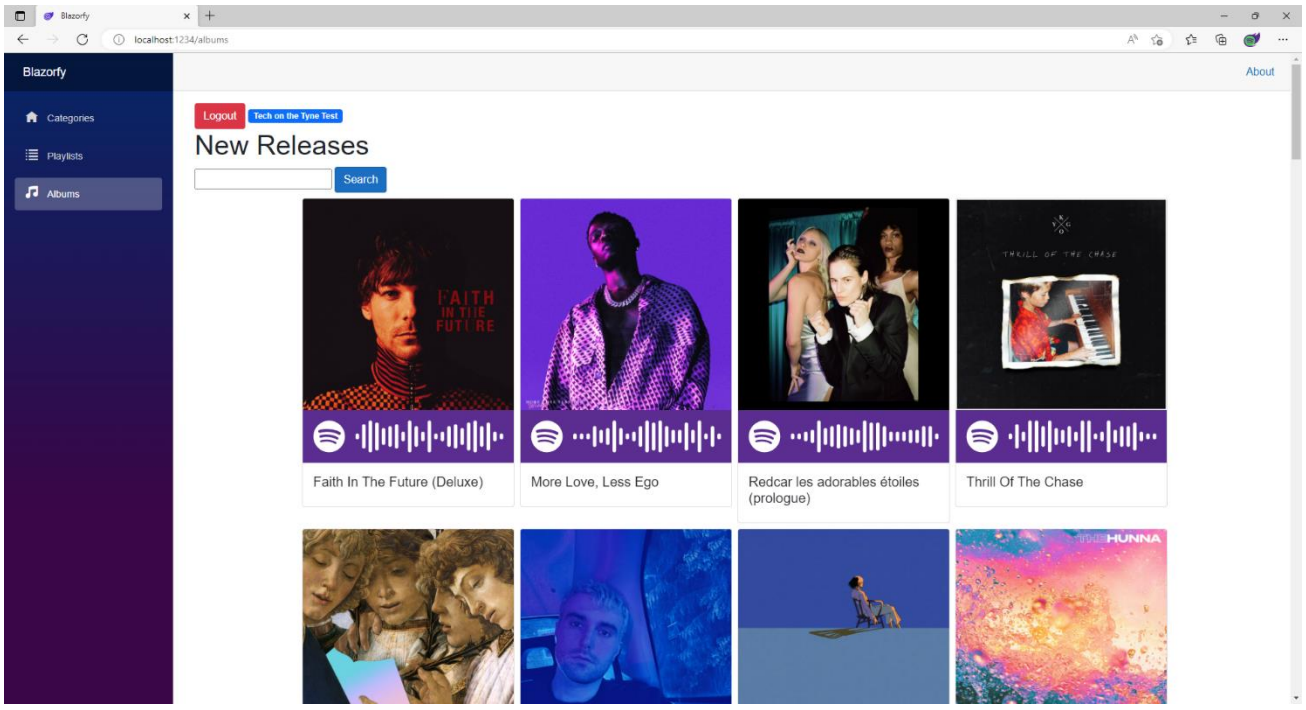
```
Albums
```

Then you can then go to the **Menu** in **Visual Studio Code** and select **File** and then **Save All** you may see in the **Terminal** a message saying **Do you want to restart your app - Yes (y) / No (n) / Always (a) / Never (v)?** you can select the **Terminal** then type **y** for **Yes** or **a** for **Always** to keep what you have done so far.

If you return to the **Browser** you should be where you left off but there will now be an **Albums** option in the **Menu** as shown below:



You can then select **Albums** and you should see a list of **New Releases** of **Albums** similar to as follows:

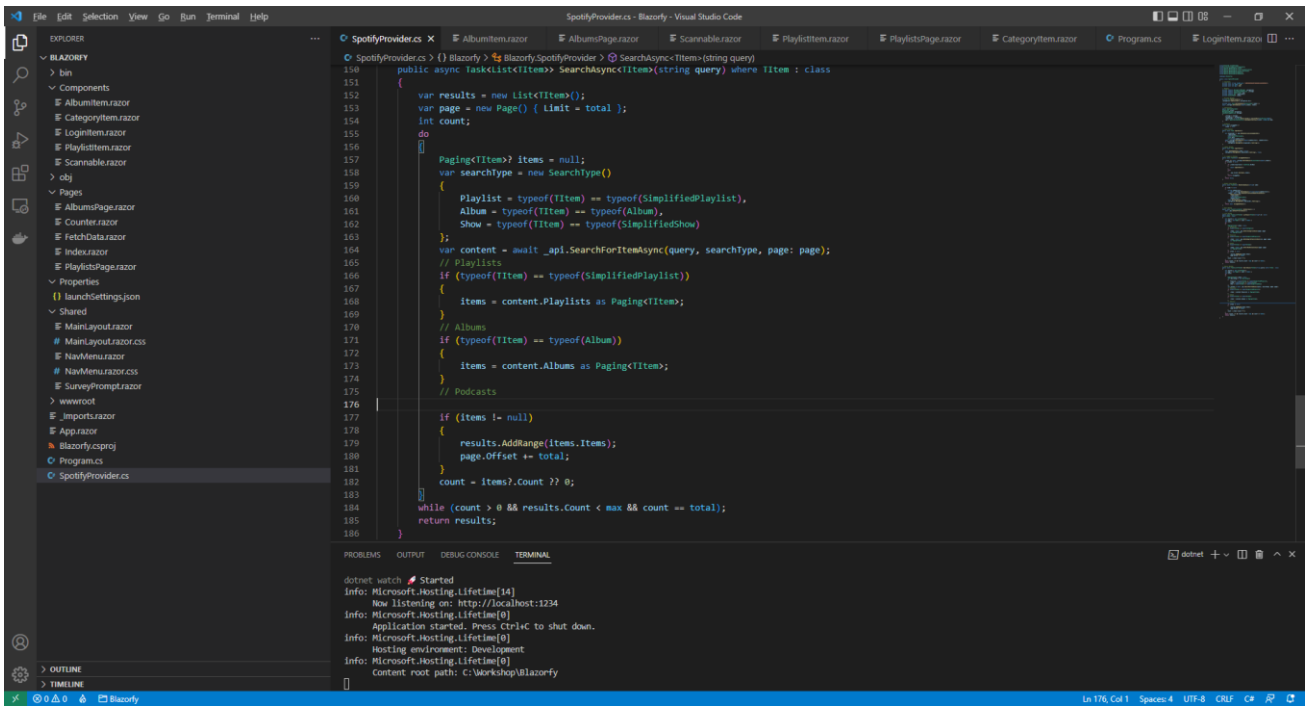


You can also type in your favourite **Album** and then select **Search** to find it. You'll also notice underneath each **Album** is a **Spotify Code**, if you have **Spotify** on your *iPhone* or *Android* device you can use the option to scan these using the app and can then checkout the **Album** for yourself.

If you don't see anything then check that you've completed each part correctly and double-check that what you have is the same, if everything is working then you can proceed to the next part of the **Workshop**.

Podcasts

We'll add a new **Page** to **Search** for **Podcasts**, to do this return to **Visual Studio Code** for **Blazorfy** and then from **Explorer** select *SpotifyProvider.cs* as follows:



You should also still have open the **Browser** showing the Login option, if you don't then in **Visual Studio Code** if it was still open select the **Terminal** and then press **Ctrl+C** in **Windows** or **Command+C** on **Mac** on the **Keyboard**, or if **Visual Studio Code** was closed relaunch **Visual Studio Code** and then select the **New Terminal** then in the **Terminal**. With the **Terminal** in **Visual Studio Code** open type the following which should relaunch the **Browser**.

```
dotnet watch
```

Then in **Visual Studio Code** within *SpotifyProvider.cs* you will define part of the **Method** that will be used to get **New Releases** of **Albums**. In the **Method** of **`SearchAync`** and below the **Comment** of **`// Podcasts`** type the following:

```
if (typeof(TItem) == typeof(SimplifiedShow))
{
    items = content.Shows as Paging<TItem>;
}
```

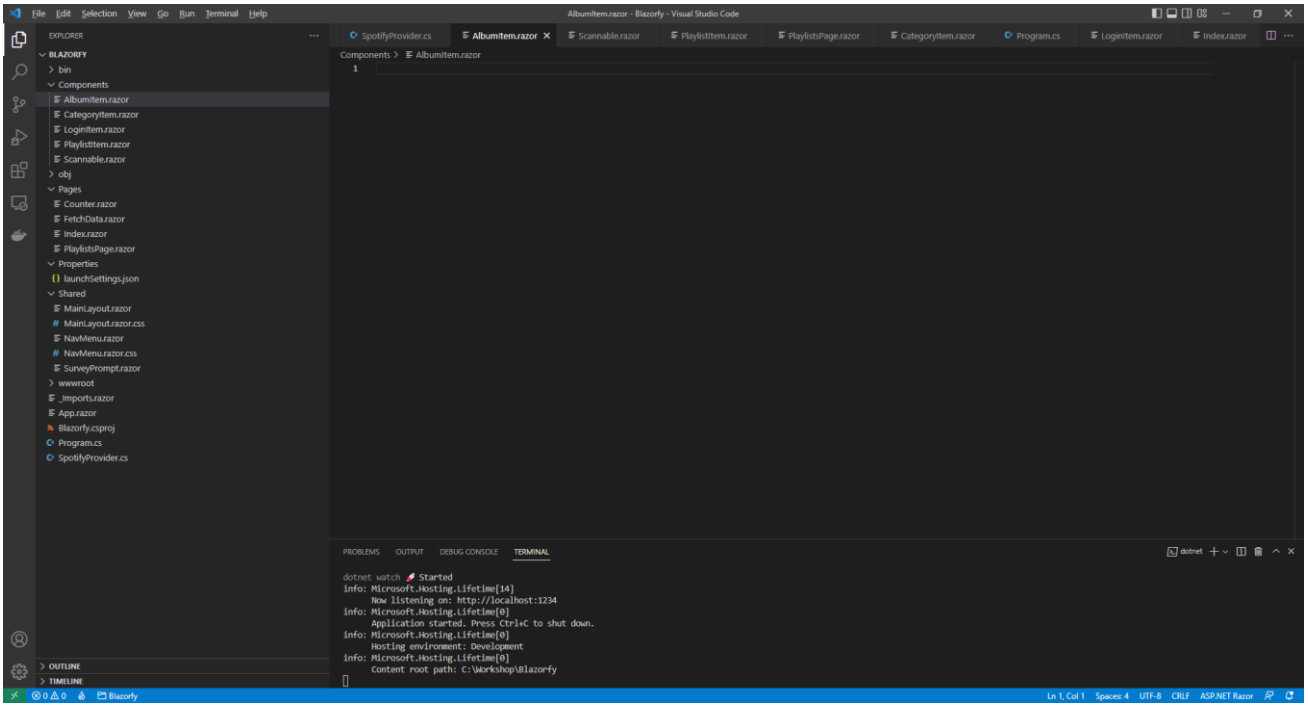
This part of the **Method** will be used to **Search** for Podcasts from **Spotify** when the **`type`** is **`SimplifiedShow`**.

Then you can then go to the **Menu** in **Visual Studio Code** and select **File** and then **Save All** you may see in the **Terminal** a message saying **Do you want to restart your app - Yes (y) / No (n) / Always (a) / Never (v)?** you can select the **Terminal** then type **y** for **Yes** or **a** for **Always** to keep what you have done so far.

Then while still in **Visual Studio Code** from **Explorer** select the **Folder** for **Components** then with the **Folder** for **Components** selected you should then select the **New File...** option and type in the name as follows then press **Enter**:

```
PodcastItem.razor
```

This will form the basis of another **Component** and will be a blank **Component** as follows:



Within *PodcastItem.razor* in **Visual Studio Code** you can define the **Component** by typing in the following:

```
@namespace Blazorfy
<div class="card">
    @if (Value.Images.Count > 0)
    {
        <img src="@Value.Images[0].Url" class="card-img-top" alt="@Value.Name" />
    }
    <Scannable Value="@Value.Uri" />
    <div class="card-body">
        <h5 class="card-title">
            @Value.Name
        </h5>
    </div>
</div>

@code
{
    [Parameter]
    public SimplifiedShow Value { get; set; } = new();
}
```

The first part of the **Component** is the **`namespace`** for the application which is **`Blazorfy`**. Then there is some **HTML** to define the layout of the **Podcast Item** this includes a check to see if there are any **Images** with **`if`** that if **`true`** will then use the **img** to display the first image since **Images** is an **Array** you use the **`[`** and **`]`** to provide an **Index** in this case *0* to get it and we also set the **alt** of the **img** to the **Name** of the **Podcast** which is also displayed within a **h5** and the **Podcast** itself will be provided to the **Property** for **`Value`**.

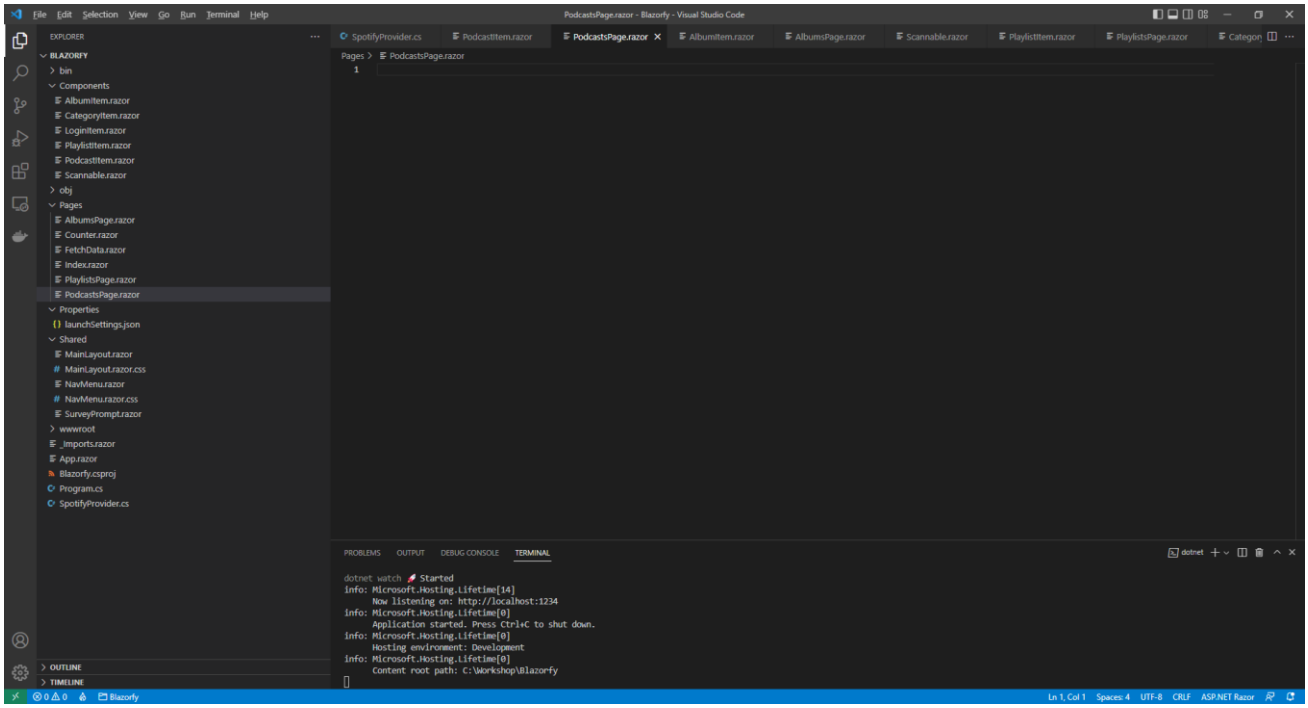
You'll also see the inclusion of the **Component** for the **`Scannable`** which is provided with the **`Uri`** of the **Podcast**, within **Spotify for Developers** a **Podcast** is called a **Show**.

Then you can then go to the **Menu** in **Visual Studio Code** and select **File** and then **Save All** you may see in the **Terminal** a message saying **Do you want to restart your app - Yes (y) / No (n) / Always (a) / Never (v)?** you can select the **Terminal** then type **y** for **Yes** or **a** for **Always** to keep what you have done so far.

Then in **Visual Studio Code** from **Explorer** select the **Folder** for **Pages** then with the **Folder** for **Pages** selected you should then select the **New File...** option and type in the name as follows then press **Enter**:

```
PodcastsPage.razor
```

This will form the basis of a **Page** and will be a blank **Page** as follows:



Within *PodcastsPage.razor* in **Visual Studio Code** you can define the entire **Page** by typing in the following:

```
@page "/podcasts"
@inject SpotifyProvider _provider;
<LoginItem Value="@_provider.IsLoggedIn" />
@if (_provider.IsLoggedIn)
{
    // Items Output

}

@code
{
    public List<SimplifiedShow> Items { get; set; } = new();

    [Parameter]
    [SupplyParameterFromQuery]
    public string? Search { get; set; }

    // Items Method

}
```

This **Page** includes a **page** directive which create a **Route** needed to navigate to this **Page** which will be used from the **Menu** later.

There is also an **`inject`** to provide the **Instance** of the **`SpotifyProvider`** using **Dependency Injection**. Then there is the **Component** of **`LoginItem`** with the **`Value`** being provided with the **Property** for **`IsLoggedIn`** from the **`class`**.

There is also the **Property** for **`Items`** to be displayed in the **Page** along with a **Property** for the **Search** that will be provided, which will be from a query to the page from a **Form** which is denoted with the **Attribute** of **`SupplyParameterFromQuery`**.

While still within *PodcastsPage.razor* in **Visual Studio Code** and below the **Comment** for **`// Items Method`** type the following **Method**:

```
protected override async Task OnParametersSetAsync()
{
    Items.Clear();
    if (await _provider.IsLoggedInAsync())
    {
        if (Search != null)
        {
            Items = await _provider.SearchAsync<SimplifiedShow>(Search);
        }
    }
}
```

This is a special **Method** where the implementation of which has been overridden to provide our own denoted with **`override`** in this case it is for **`OnParametersSetAsync`** and will populate the **`List`** of **`Items`** for the **`type`** of **`SimplifiedShow`** which represents a **Podcast** using the **Method** of **`SearchAsync`** and provide the value to this with **`type`** of **`SimplifiedShow`**.

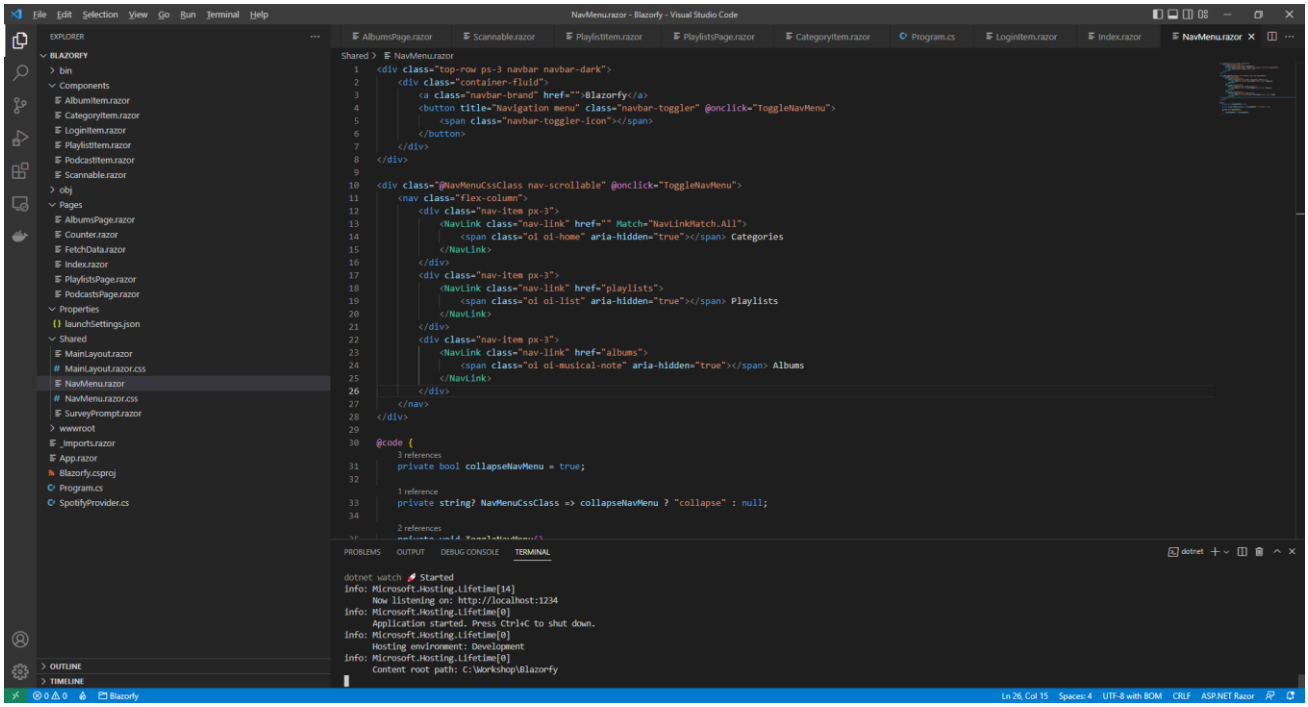
Finally while still within *PodcastsPage.razor* in **Visual Studio Code** and below the **Comment** for **`// Items Output`** type the following:

```
<h1>Search @Search</h1>
<form @onsubmit="OnParametersSetAsync">
    <input type="text" @bind="Search" @bind:event="oninput" />
    <button class="btn btn-primary">Search</button>
</form>
<div class="container">
    <div class="row row-cols-1 row-cols-md-4 p-2 g-2">
        @foreach (var item in Items)
        {
            <div class="col">
                <PodcastItem Value="@item" />
            </div>
        }
    </div>
</div>
```

This defines how the **Page** will look, there is a **form** to perform a **Search** this has an **`input`** which is where the **Podcast** being looked for will be typed in and then there is a **`button`** to perform the **Search** and below this the **Component** for the **Podcast Item** will be used to display the **Podcasts**.

You can then go to the **Menu** in **Visual Studio Code** and select **File** and then **Save All** you may see in the **Terminal** a message saying **Do you want to restart your app - Yes (y) / No (n) / Always (a) / Never (v)?** you can select the **Terminal** then type **y** for **Yes** or **a** for **Always** to keep what you have done so far.

Then in **Visual Studio Code** from the **Explorer** for **Blazorfy** open **Shared** by selecting the **>** next to it in **Explorer** and select **NavMenu.razor** as follows:



Then within *NavMenu.razor* after the ** ** of the **Albums** section in the **Menu** shown below:

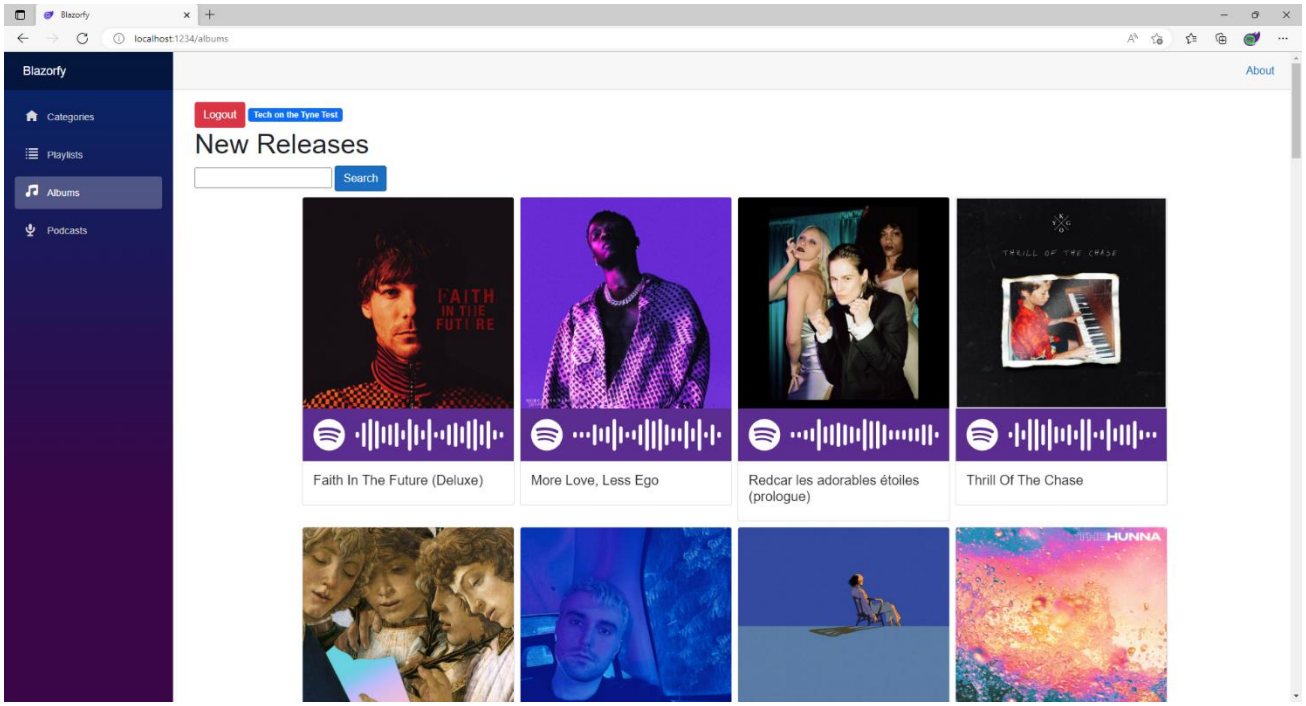
```
<div class="nav-item px-3">
    <NavLink class="nav-link" href="albums">
        <span class="oi oi-musical-note" aria-hidden="true"></span> Albums
    </NavLink>
</div>
```

You can add **Podcasts** section to the **Menu** by typing in the following:

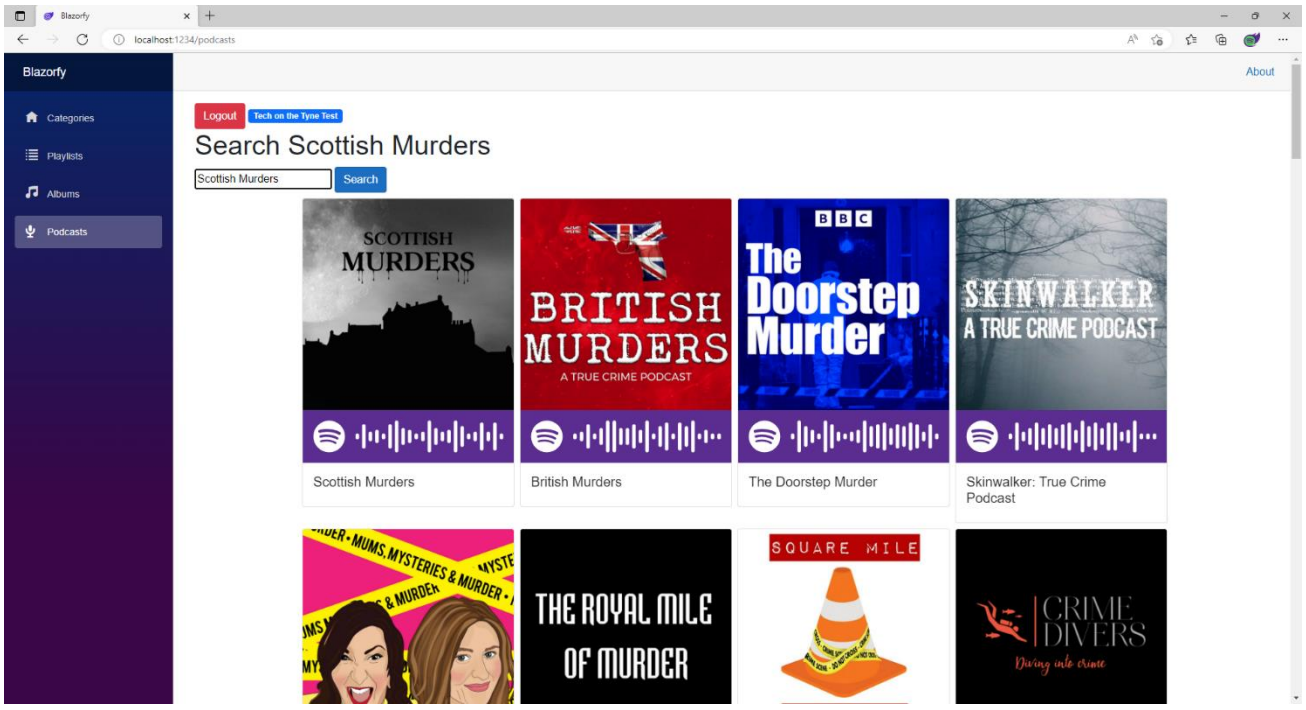
```
<div class="nav-item px-3">
    <NavLink class="nav-link" href="podcasts">
        <span class="oi oi-microphone" aria-hidden="true"></span> Podcasts
    </NavLink>
</div>
```

You can then go to the **Menu** in **Visual Studio Code** and select **File** and then **Save All** you may see in the **Terminal** a message saying **Do you want to restart your app - Yes (y) / No (n) / Always (a) / Never (v)?** you can select the **Terminal** then type **y** for **Yes** or **a** for **Always** to keep what you have done so far.

If you return to the **Browser** you should be where you left off but there will now be an **Podcasts** option in the **Menu** as shown below:



You can then select **Podcast** then search for your favourite and should see something similar to as follows:



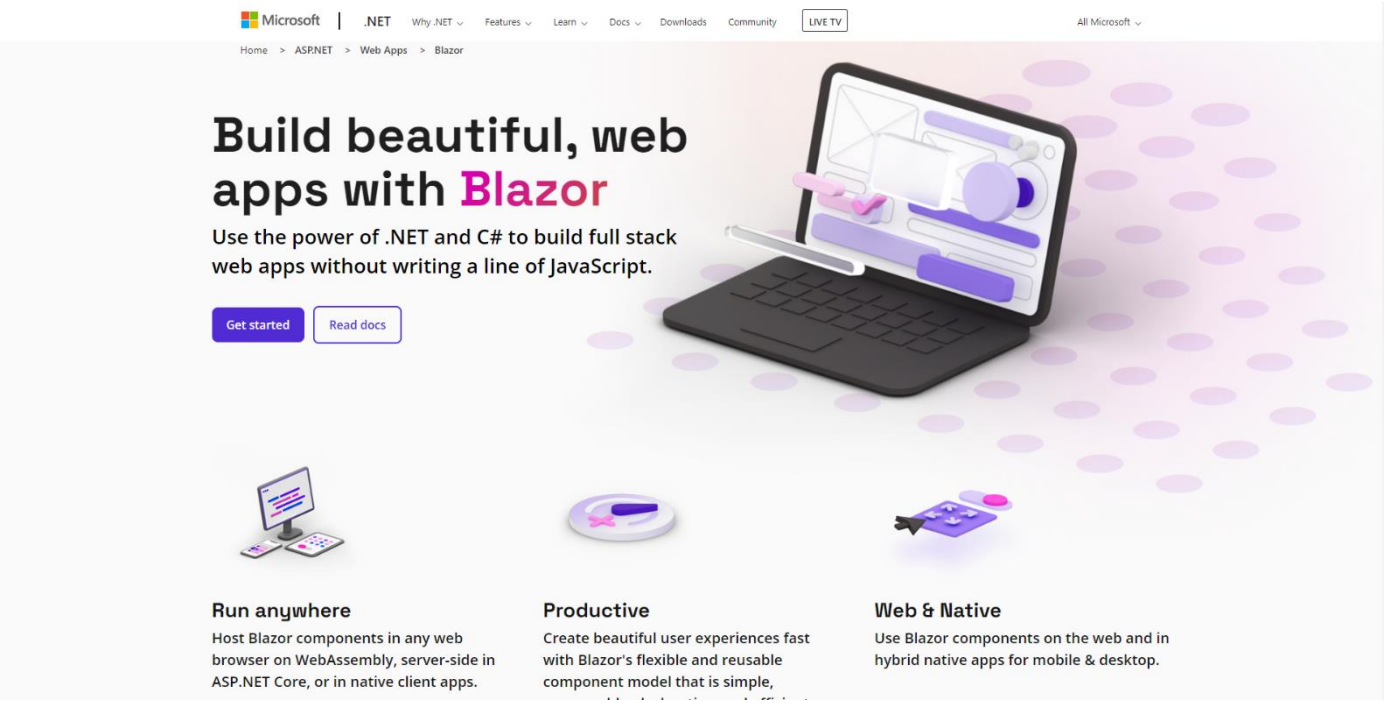
Again underneath each **Podcast** is a **Spotify Code**, if you have **Spotify** on your *iPhone* or *Android* device you can use the option to scan these using the app and can then subscribe to the **Podcast** yourself.

If you don't see anything then check that you've completed each part correctly and double-check that what you have is the same and then you can **Logout** and **Close** the **Browser** and **Visual Studio Code**.

Finish

Blazor

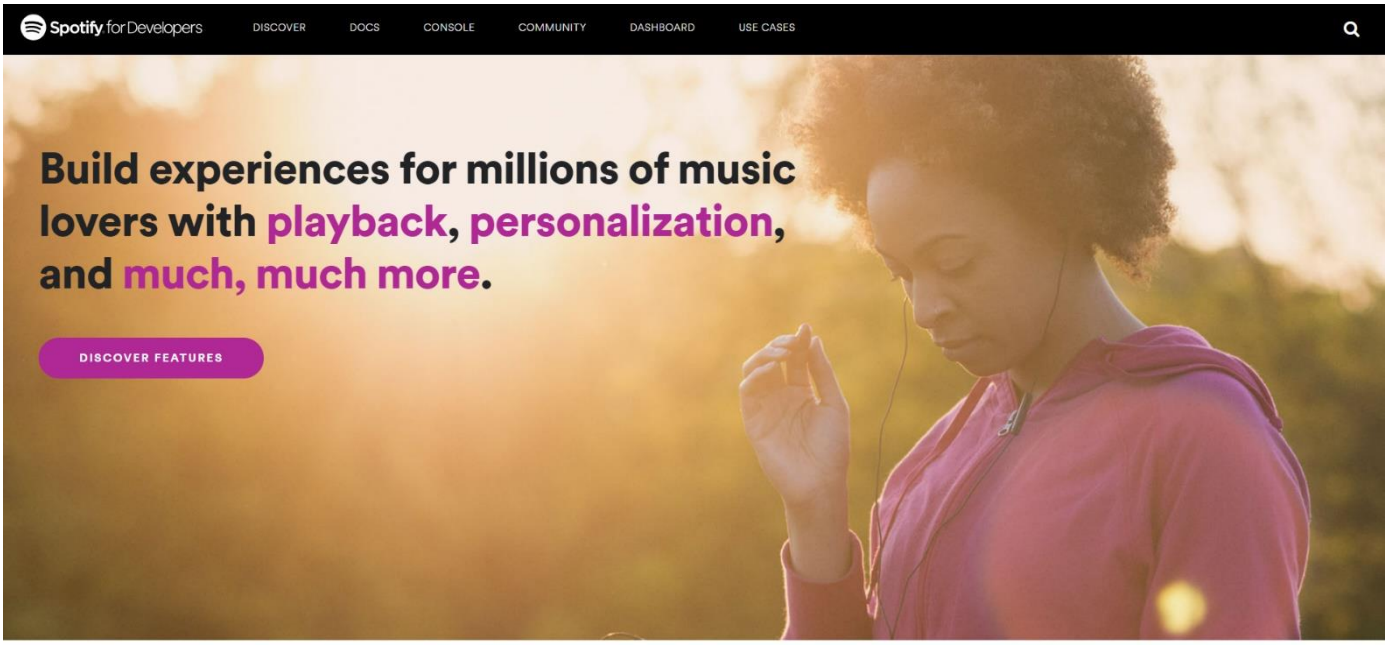
Blazor allows you to build interactive client web applications composed using **C#**, **HTML** and **CSS** that supports both **Client** using **Web Assembly** and **Server** using **ASP.NET** created by **Microsoft**.



Blazor allows you to develop either for **Server** where events are passed using **SignalR** to the **Client** or you can run your **C#** code directly on the **Client** in the **Browser** using **WebAssembly** and you can even re-use code between **Server** and **Client**. You can find out more about **Blazor** including documentation, examples and more at blazor.net.

Spotify

Spotify is a music streaming service that allows you to discover and play a variety of **Music** and **Podcasts** from your *iPhone* or *Android* or other devices and a **Web API** for developers with **Spotify for Developers**.



Spotify for Developers allows you to deliver your own experiences powered by a **Web API** for **Albums**, **Artists**, **Shows** and their **Episodes** along with **Audiobooks** and their **Chapters**. You can also **Search** content, show **User** information, manage **Playlists** and get **Categories** or **Genres**. You can also control playback with **Player** or see what **Markets** you can get **Spotify**.

Spotify for Developers uses a normal **Spotify** account and you can sign up for free and then set up **Apps** in the **Dashboard** to create the **Client Id** to use the service and as **Edit Settings** such as **Redirect URIs** or add up to 25 **Spotify** accounts for development, or when ready request an **Extension** to go public. You can find out more about **Spotify for Developers** and the **Web API** including documentation, examples, online console and more at developer.spotify.com.

Summary

Blazor is a powerful platform that allows you to create experiences in your **Browser** like **Blazorfy** using **Spotify** so you can see what you can do with it and leverage the power of the **.NET** platform in your applications, whether you've never written a single line of code until today, or had never heard or used **Blazor**, **.NET** or **C#** or you just wanted to try something new hopefully you've learned something today and you can go back over the **Workshop** and look up many of the concepts, but the best way is to try something small and grow from there, you can go from **Hello World** to **Blazorfy** in a couple of hours, where would spending more time take you? You'll just have to find out!