Developing with Power BI Embedding using .NET 5

In this lab, you will create a new .NET 5 project for an MVC web application and then you will go through the steps required to implement Power BI embedding. You will use the new Microsoft authentication library named *Microsoft.Identity.Web* to provide an interactive login experience and to acquire access tokens which you will need to call the Power BI Service API. After that, you will write the server-side C# code and the client-side JavaScript code to embed a simple Power BI report on a custom Web page. In the later exercises of this lab, you will add project-level support for Node.js, TypeScript and webpack so that you can migrate the client-side code from JavaScript to TypeScript so your code receives the benefits of strong typing, IntelliSense and compile-time type checks.

These lab exercises require Microsoft 365 tenant in which you can create Power BI workspaces and create Azure AD applications. If you need a tenant for a development environment, you can follow the steps in Create Power BI Development Environment.pdf.

To complete this lab, your developer PC must be configured to allow the execution of PowerShell scripts with Windows PowerShell 5. Your developer workstation must also have the following software and developer tools installed.

1) **PowerShell cmdlet library for AzureAD** – [download]
2) **.NET 5 SDK** – [download]
3) **Visual Studio Code** – [download]
4) **Node.js** – [download]
5) **Visual Studio 2019** (optional) – [download]

Please refer to this setup document if you need more detail on how to configure your developer workstation to work on this tutorial.

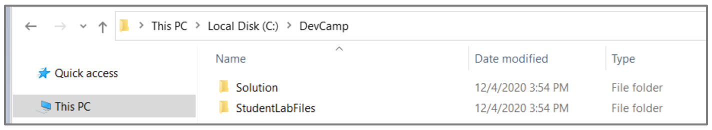
Exercise 1: Create a New Azure AD Application

In this exercise, you will begin by copying the student files into a local folder on your student workstation. After that, you will use the .NET 5 CLI to create a new .NET 5 project for an MVC web application with support for authentication.

1. Download the student lab files to a local folder on your developer workstation.
   a) Create a new top-level folder on your workstation named **DevCamp** at a location such as **c:\DevCamp**.
   b) Download the ZIP archive with the student lab files from GitHub by clicking the following link.

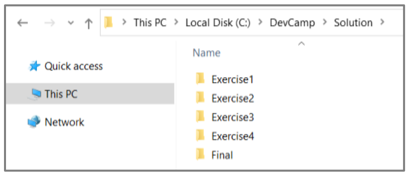
```
https://github.com/PowerBiDevCamp/DOTNET5-UserOwnsData-Tutorial/archive/main.zip
```

   c) From inside the ZIP file, extract the **Solutions** folder and **StudentLabFiles** folder into a local folder such as **c:\DevCamp\**.
   d) The **DevCamp** folder should now contain a Solutions folder and **StudentLabFiles** folder as shown in the following screenshot.



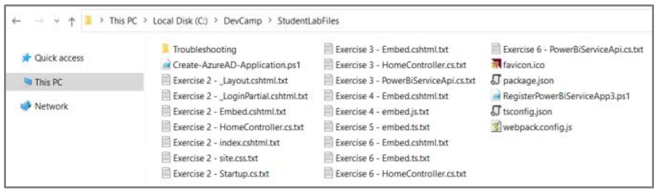
Name	Date modified	Type
Solution	12/4/2020 3:54 PM	File folder
StudentLabFiles	12/4/2020 3:54 PM	File folder

   e) The **Solutions** folder contains child folders with the completed .NET 5 project solution for each of the exercises.



Exercise1
Exercise2
Exercise3
Exercise4
Final

   f) The **StudentLabFiles** folder contains files that you will need as you move through the exercises in this lab.



Troubleshooting
Create-AzureAD-Application.ps1
Exercise 2 - _Layout.cshtml.txt
Exercise 2 - _LoginPartial.cshtml.txt
Exercise 2 - Embed.cshtml.txt
Exercise 2 - HomeController.cs.txt
Exercise 2 - index.cshtml.txt
Exercise 2 - site.css.txt
Exercise 2 - Startup.cs.txt
Exercise 3 - Embed.cshtml.txt
Exercise 3 - HomeController.cs.txt
Exercise 3 - PowerBiServiceApi.cs.txt
Exercise 4 - Embed.cshtml.txt
Exercise 4 - embed.js.txt
Exercise 5 - embed.ts.txt
Exercise 6 - Embed.cshtml.txt
Exercise 6 - Embed.ts.txt
Exercise 6 - HomeController.cs.txt
Exercise 6 - PowerBiServiceApi.cs.txt
favicon.ico
package.json
RegisterPowerBiServiceApp3.ps1
tsconfig.json
webpack.config.js

You will be required to write quite a bit code as you complete the exercises in this lab. Many of the files in the **StudentLabFiles** folder are provided to you as a convenience in case you'd like to copy-and-paste the code required during the various exercise of this lab..

2. Walk through the PowerShell code in **Create-AzureAD-Application.ps1** to understand what it does.
   a) Using Windows Explorer, look in the **StudentLabFiles** folder and locate the script named **Create-AzureAD-Application.ps1.**
   b) Open **Create-AzureAD-Application.ps1** in a text editor such as the Windows PowerShell ISE or Notepad.
   c) The script begins by calling **Connect-AzureAD** to establish a connection with Azure AD.

```
$authResult = Connect-AzureAD
```

   d) The script contains two variables to set the application display name and a reply URL of **https://localhost:5001/signin-oidc**.

```
$appDisplayName = "User-Owns-Data Sample App"
$replyUrl = "https://localhost:5001/signin-oidc"
```

When you register a reply URL with **localhost** with a port number such as **5001**, Azure AD will allow you to perform testing with reply URLs that use localhost and any other port number. For example, you can use a reply URL of **https://localhost:44300/signin-oidc**.

   e) The script also contains the code below which creates a new **PasswordCredential** object for an app secret.

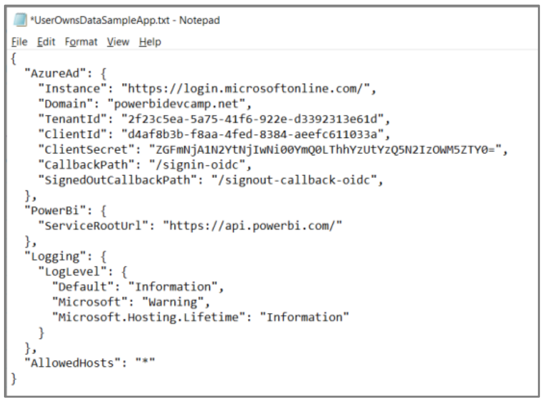
```
# create app secret
$newGuid = New-Guid
$appSecret = ([System.Convert]::ToBase64String([System.Text.Encoding]::UTF8.GetBytes(($newGuid))))+"="
$startDate = Get-Date
$passwordCredential = New-Object -TypeName Microsoft.Open.AzureAD.Model.PasswordCredential
$passwordCredential.StartDate = $startDate
$passwordCredential.EndDate = $startDate.AddYears(1)
$passwordCredential.KeyId = $newGuid
$passwordCredential.Value = $appSecret
```

   f) Down below, you can see the call to the **New-AzureADApplication** cmdlet which creates a new Azure AD application.

```
# create Azure AD Application
$aadApplication = New-AzureADApplication `
                        -DisplayName $appDisplayName `
                        -PublicClient $false `
                        -AvailableToOtherTenants $false `
                        -ReplyUrls @($replyUrl) `
                        -Homepage $replyUrl `
                        -PasswordCredentials $passwordCredential
```

Note you must execute this script using Windows PowerShell 5 and not PowerShell 7. This restriction is due to incompatibilities between PowerShell7 and the PowerShell module named **AzureAD**.

3. Execute the PowerShell script named **Create-Azure-ADApplication.ps1**
   a) Execute **Create-Azure-ADApplication.ps1**.using the Windows PowerShell ISE or the Windows PowerShell command line.
   b) When prompted for credentials, log in with an Azure AD user account in the same tenant where you are using Power BI.
   c) When the PowerShell script runs successfully, it will create and open a text file named **UserOwnsDataSampleApp.txt**.


*UserOwnsDataSampleApp.txt - Notepad
File Edit Format View Help

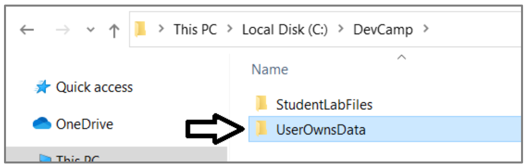
```
{
  "AzureAd": {
    "Instance": "https://login.microsoftonline.com/",
    "Domain": "powerbidevcamp.net",
    "TenantId": "2f23c5ea-5a75-41f6-922e-d3392313e61d",
    "ClientId": "d4af8b3b-f8aa-4fed-8384-aeefc611033a",
    "ClientSecret": "ZGFmNjA1N2YtNjIwNi00YmQ0LThhYzUtYzQ5N2IzOWM5ZTY0=",
    "CallbackPath": "/signin-oidc",
    "SignedOutCallbackPath": "/signout-callback-oidc",
  },
  "PowerBi": {
    "ServiceRootUrl": "https://api.powerbi.com/"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*"
}
```

You should leave the text file **UserOwnsDataSampleApp.txt** open for now. This file contains JSON configuration data that you will copy and paste into the **appsettings.json**.file of the new .NET 5 project you will create the next exercise.

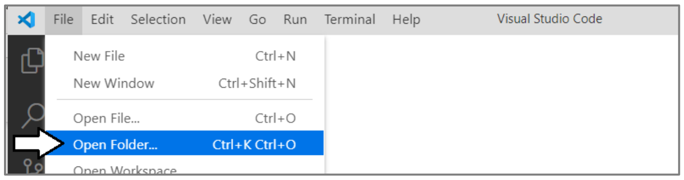
Exercise 2: Create a .NET 5 Project for a Secure Web Application

In this exercise, you will use the .NET CLI to create a new .NET 5 project for an MVC web application with support for authentication using the new Microsoft authentication library named Microsoft.Identity.Web..

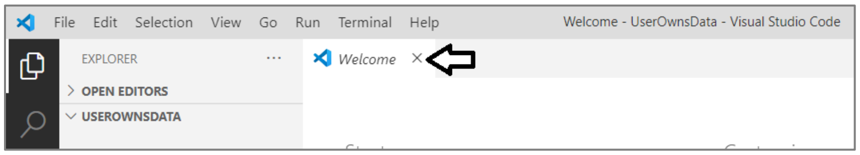
1. Create a new .NET 5 project for an ASP.NET MVC web application.
   a) Using Windows Explorer, create a child folder inside the **C:\DevCamp** folder named **UserOwnsData**.



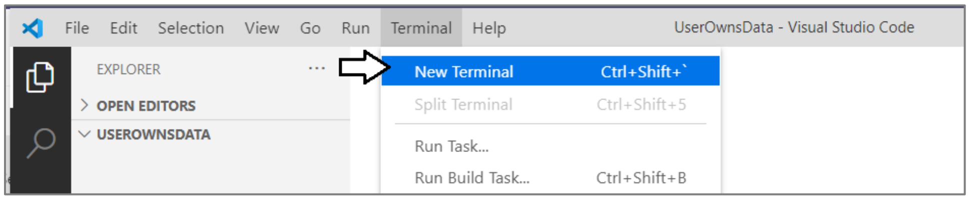
   b) Launch Visual Studio Code.
   c) Use the **Open Folder…** command in Visual Studio Code to open the **UserOwnsData** folder.



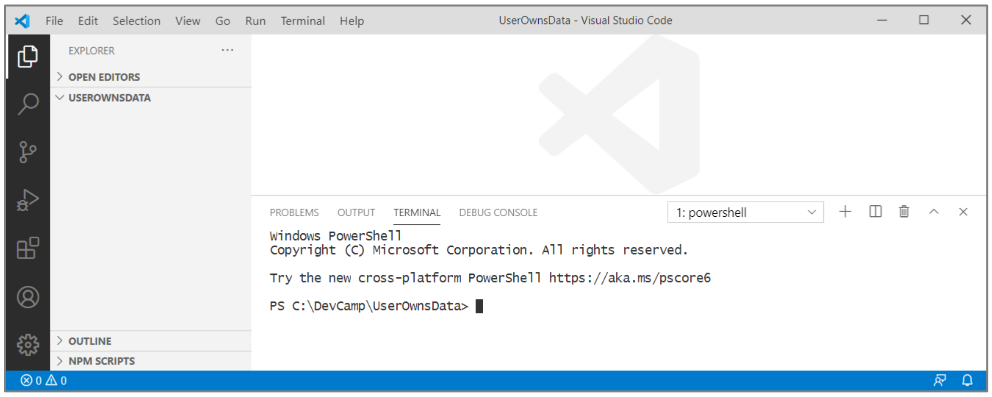
   d) Once you have open the **UserOwnsData** folder, close the **Welcome** page.



2. Use the Terminal console to verify the current version of .NET
   a) Use the **Terminal > New Terminal** command or the [**Ctrl+Shift+**`] keyboard shortcut to open the Terminal console.


File
Edit
Selection
View
Go
Run
Terminal
Help
UserOwnsData - Visual Studio Code
EXPLORER
OPEN EDITORS
USEROWNSDATA
New Terminal Ctrl+Shift+`
Split Terminal Ctrl+Shift+5
Run Task...
Run Build Task... Ctrl+Shift+B

   b) You should now see a Terminal console with a cursor where you can type and execute command-line instructions.

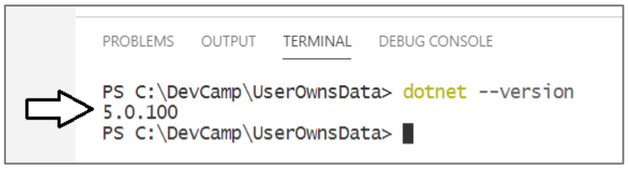


```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\DevCamp\UserOwnsData> 
```

c) Type the following **dotnet** command-line instruction into the console and press **Enter** to execute it.

```
dotnet --version
```

d) When you run the command, the **dotnet** CLI should respond by display the .NET version number.



```
PS C:\DevCamp\UserOwnsData> dotnet --version
5.0.100
PS C:\DevCamp\UserOwnsData>
```

Make sure you .NET version number is **5.0.100** at a minimum. If you are working with .NET CORE version 3.1 or early, the project templates for creating new web applications are much different and these lab instructions will not work as expected. If you do not have .NET 5 installed, you must install the .NET 5 SDK before you can move past this point.

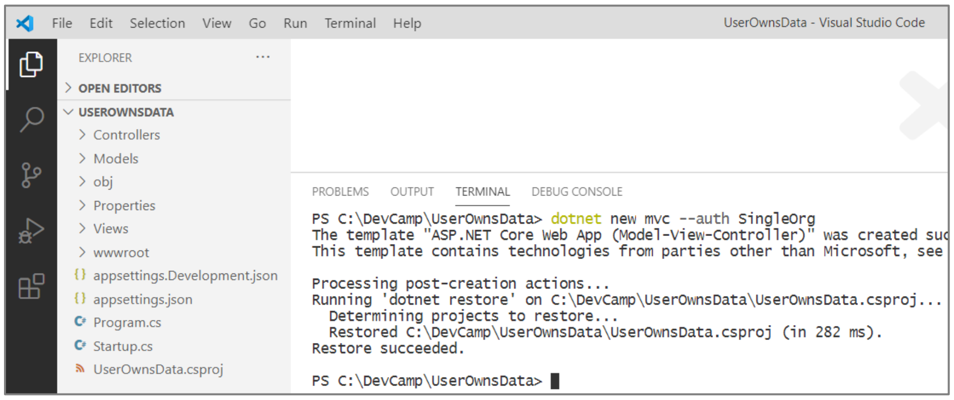
3. Run .NET CLI commands to create a new ASP.NET MVC project.

a) In the Terminal console, type and execute the following command to generate a new .NET console application.

```
dotnet new mvc --auth SingleOrg
```

The **--auth SingleOrg** parameter instructs the .NET 5 CLI to generate the new web application with extra code with authentication support using Microsoft's new authentication library named Microsoft.Identity.Web.

b) After running the **dotnet new** command, you should see that quite a few new folders and files have been added to the project.

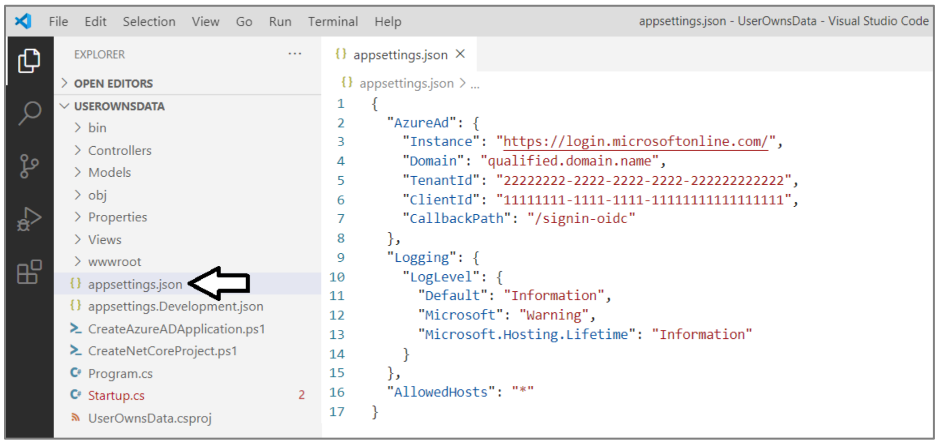


```
PS C:\DevCamp\UserOwnsData> dotnet new mvc --auth SingleOrg
The template "ASP.NET Core Web App (Model-View-Controller)" was created suc
This template contains technologies from parties other than Microsoft, see

Processing post-creation actions...
Running 'dotnet restore' on C:\DevCamp\UserOwnsData\UserOwnsData.csproj...
  Determining projects to restore...
  Restored C:\DevCamp\UserOwnsData\UserOwnsData.csproj (in 282 ms).
Restore succeeded.

PS C:\DevCamp\UserOwnsData>
```

1. Copy the JSON in **UserOwnsDataSampleApp.txt** into the **appsettings.json** file in your project.

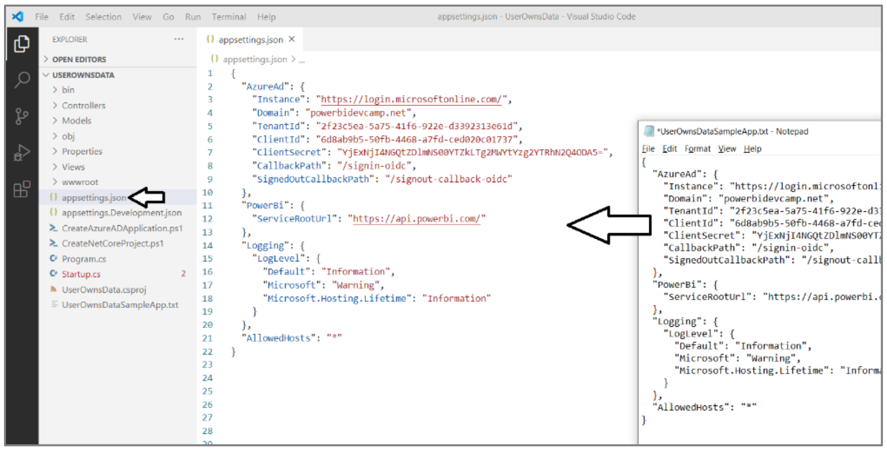
a) Return to the **UserOwnsData** project in Visual Studio Code and open the **appsettings.json** file.

b) The **appsettings.json** file should initially appear like the screenshot below.



```
{
  "AzureAd": {
    "Instance": "https://login.microsoftonline.com/",
    "Domain": "qualified.domain.name",
    "TenantId": "22222222-2222-2222-2222-222222222222",
    "ClientId": "11111111-1111-1111-11111111111111111",
    "CallbackPath": "/signin-oidc"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*"
}
```

c) Delete the contents of **appsettings.json** and replace it by copying and pasting the contents of **UserOwnsDataSampleApp.txt**


appsettings.json - UserOwnsData - Visual Studio Code
EXPLORER
OPEN EDITORS
USEROWNSDATA
bin
Controllers
Models
obj
Properties
Views
wwwroot
appsettings.json
appsettings.Development.json
CreateAzureADApplication.ps1
CreateNetCoreProject.ps1
Program.cs
Startup.cs
UserOwnsData.csproj
UserOwnsDataSampleApp.txt
*UserOwnsDataSampleApp.txt - Notepad

```json
{
  "AzureAd": {
    "Instance": "https://login.microsoftonline.com/",
    "Domain": "powerbidevcamp.net",
    "TenantId": "2f23c5ea-5a75-41f6-922e-d3392313e61d",
    "ClientId": "6d8ab9b5-50fb-4468-a7fd-ced020c01737",
    "ClientSecret": "YjExNjI4NGQtZDlmNS00YTZkLTg2MWYtYzg2YTRhN2Q4ODA5=",
    "CallbackPath": "/signin-oidc",
    "SignedOutCallbackPath": "/signout-callback-oidc"
  },
  "PowerBi": {
    "ServiceRootUrl": "https://api.powerbi.com/"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*"
}
```

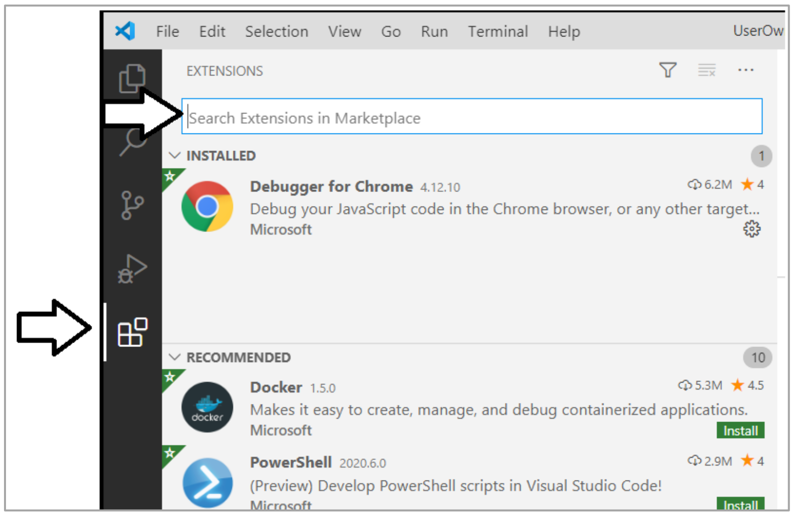
Note the **PowerBi:ServiceRootUrl** parameter has been added as a custom configuration value to track the base URL to the Power BI Service API. When programming against the Power BI Service in the Microsoft public cloud, the URL is https://api.powerbi.com/. However, the base URL for the Power BI Service API will be different in other clouds such as the government cloud. Therefore, this value will be stored as a project configuration value so it is easy to change if required. More info

2. Configure Visual Studio Code with the Omnisharp extensions needed for C# development with .NET 5.

   a) Click on the button at the bottom of the left navigation menu to display the **EXTENSION** pane.

   b) You should be able to see what extensions are currently installed.

   c) You should also be able to search to find new extensions you'd like to install.

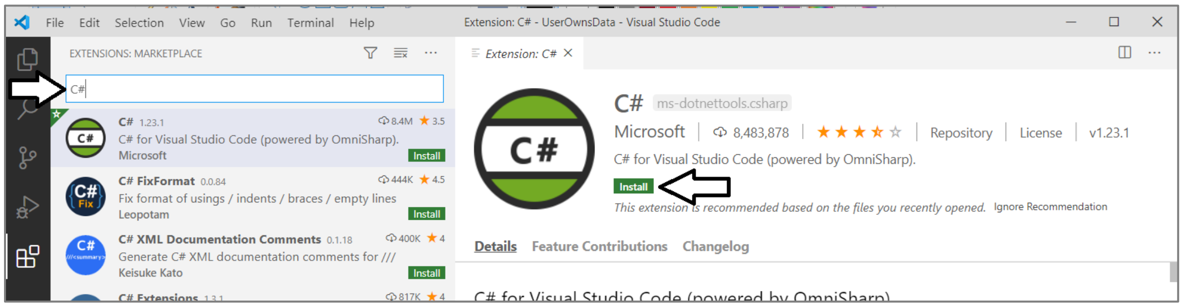


Debug your JavaScript code in the Chrome browser, or any other target...

Makes it easy to create, manage, and debug containerized applications.

(Preview) Develop PowerShell scripts in Visual Studio Code!

   d) Find and install the **C#** extension from Microsoft if it is not installed. This extension is also known as the Omnisharp extension



C# for Visual Studio Code (powered by OmniSharp).

Fix format of usings / indents / braces / empty lines

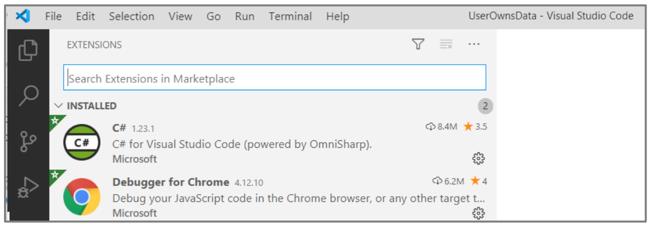
Generate C# XML documentation comments for ///

This extension is recommended based on the files you recently opened.

C# for Visual Studio Code (powered by OmniSharp)

e) Find and install the **Debugger for Chrome** extension from Microsoft if it is not already installed.

f) You should be able to confirm that the **C#** extension and the **Debugger for Chrome** extensions are now installed.


File Edit Selection View Go Run Terminal Help
UserOwnsData - Visual Studio Code
EXTENSIONS
Search Extensions in Marketplace
INSTALLED
2
C# 1.23.1
8.4M ★3.5
C# for Visual Studio Code (powered by OmniSharp).
Microsoft
Debugger for Chrome 4.12.10
6.2M ★4
Debug your JavaScript code in the Chrome browser, or any other target t...
Microsoft

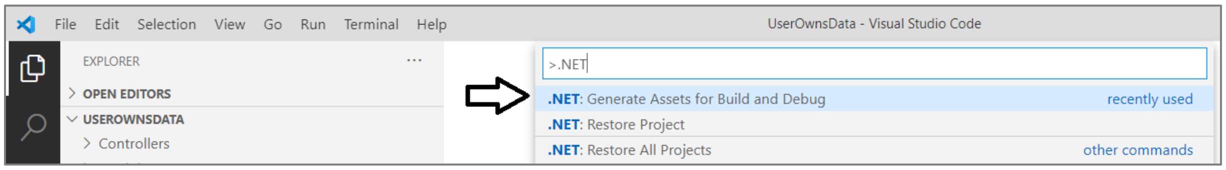
It is OK if you have other Visual Studio Code extensions installed as well. It's just important that you install these two extensions in addition to whatever other extensions you may have installed.

3. Generate the project assets required for building your project and running it in the .NET debugger.

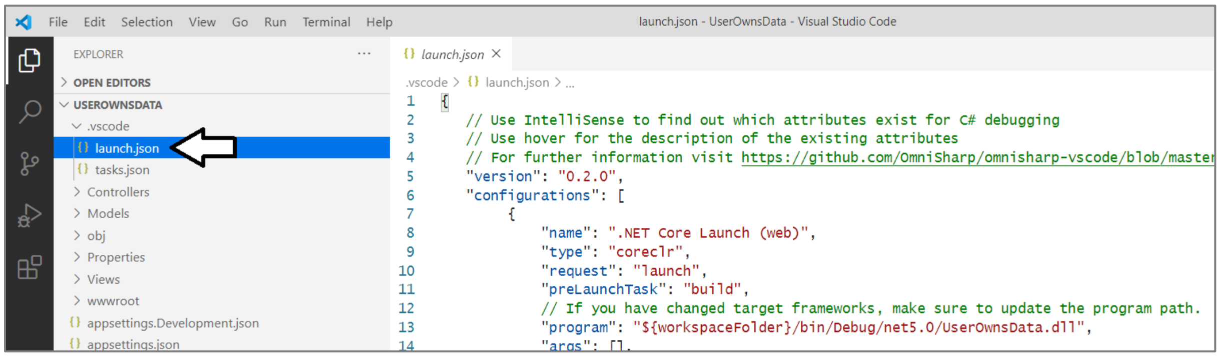
a) Open the Visual Studio Code Command Palette by using the **Ctrl**+**Shift**+**P** keyboard combination.

b) Type .NET into the Command Palette search box.

c) Select the command titled **.NET: Generate Assets for Build and Debug**.


File Edit Selection View Go Run Terminal Help
UserOwnsData - Visual Studio Code
EXPLORER
OPEN EDITORS
USEROWNSDATA
Controllers
>.NET
.NET: Generate Assets for Build and Debug
recently used
.NET: Restore Project
.NET: Restore All Projects
other commands

d) When the command runs successfully, in generates the files **launch.json** and **tasks.json** a new folder named **.vscode**.

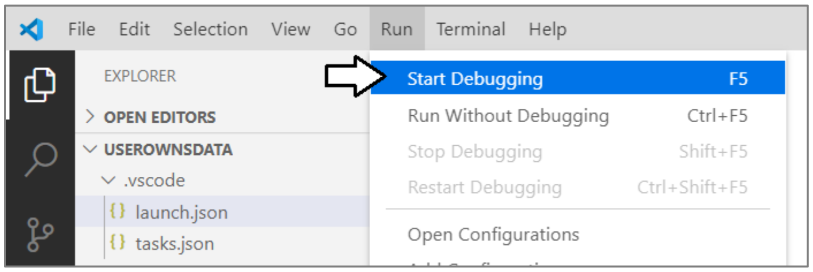

File Edit Selection View Go Run Terminal Help
launch.json - UserOwnsData - Visual Studio Code
EXPLORER
OPEN EDITORS
USEROWNSDATA
.vscode
launch.json
tasks.json
Controllers
Models
obj
Properties
Views
wwwroot
appsettings.Development.json
appsettings.json
launch.json
.vscode > launch.json > ...

```
{
    // Use IntelliSense to find out which attributes exist for C# debugging
    // Use hover for the description of the existing attributes
    // For further information visit https://github.com/OmniSharp/omnisharp-vscode/blob/master
    "version": "0.2.0",
    "configurations": [
        {
            "name": ".NET Core Launch (web)",
            "type": "coreclr",
            "request": "launch",
            "preLaunchTask": "build",
            // If you have changed target frameworks, make sure to update the program path.
            "program": "${workspaceFolder}/bin/Debug/net5.0/UserOwnsData.dll",
            "args": [],
```

It's not uncommon for the configuration of the C# extension (i.e. Omnisharp) in Visual Studio Code to cause errors when running the command to generate the assets for build and debug. If this is the case, it can be tricky to fix this problem. If the previous step to generate build and debug assets did not successfully create the **launch.json** file and the **tasks.json** file in the **.vscode** folder, you can copy these two essential files from **Troubleshooting/.vscode** folder located inside the **StudentFiles** folder.

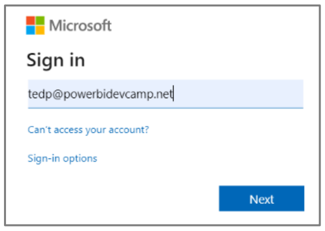
4. Run and test the **UserOwnsData** web application using Visual Studio Code and the .NET 5 debugger.

a) Start the .NET 5 debugger by selecting **Run > Start Debugging** or by pressing the **{F5}** keyboard shortcut.



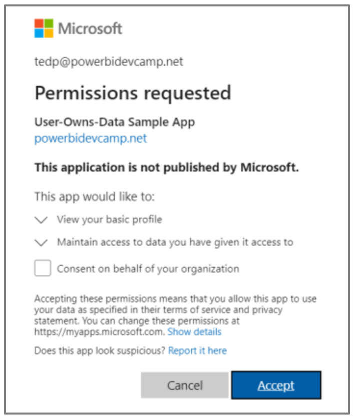
The UserOwnsData web application is currently configured authenticate the user before the user is able to view the home page. Therefore, you will be prompted to log in as soon as you launch the application in the .NET debugger.

b) When prompted to Sign in, log in using your organizational account.



c) Once you have signed in, you will be prompted by the **Permissions requested** dialog to grant consent to the application.

d) Click the **Accept** button to continue.



Microsoft

tedp@powerbidevcamp.net

Permissions requested

User-Owns-Data Sample App
powerbidevcamp.net

This application is not published by Microsoft.

This app would like to:
- View your basic profile
- Maintain access to data you have given it access to

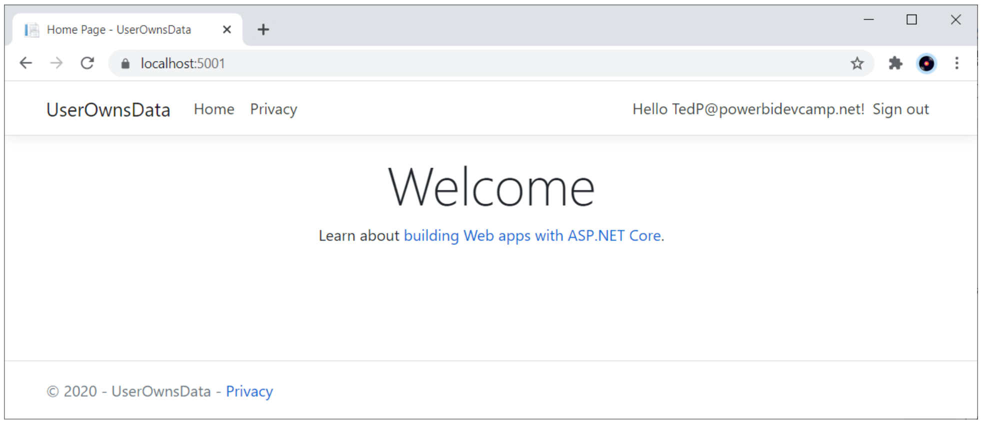
Consent on behalf of your organization

Accepting these permissions means that you allow this app to use your data as specified in their terms of service and privacy statement. You can change these permissions at https://myapps.microsoft.com. Show details

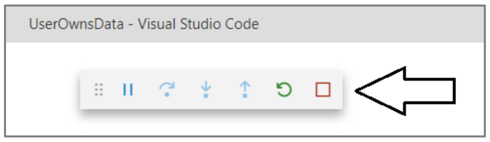
Does this app look suspicious? Report it here

Cancel Accept

e) You should now see the home page for the **UserOwnsData** web application which should match the following screenshot.

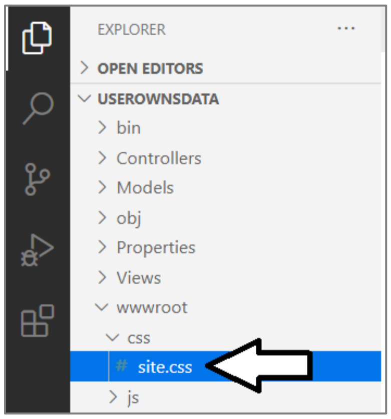

Home Page - UserOwnsData
localhost:5001
UserOwnsData
Home
Privacy
Hello TedP@powerbidevcamp.net!
Sign out
Welcome
Learn about building Web apps with ASP.NET Core.
© 2020 - UserOwnsData - Privacy

f) You're done testing. Close the browser, return to Visual Studio Code and stop the debug session using the debug toolbar.

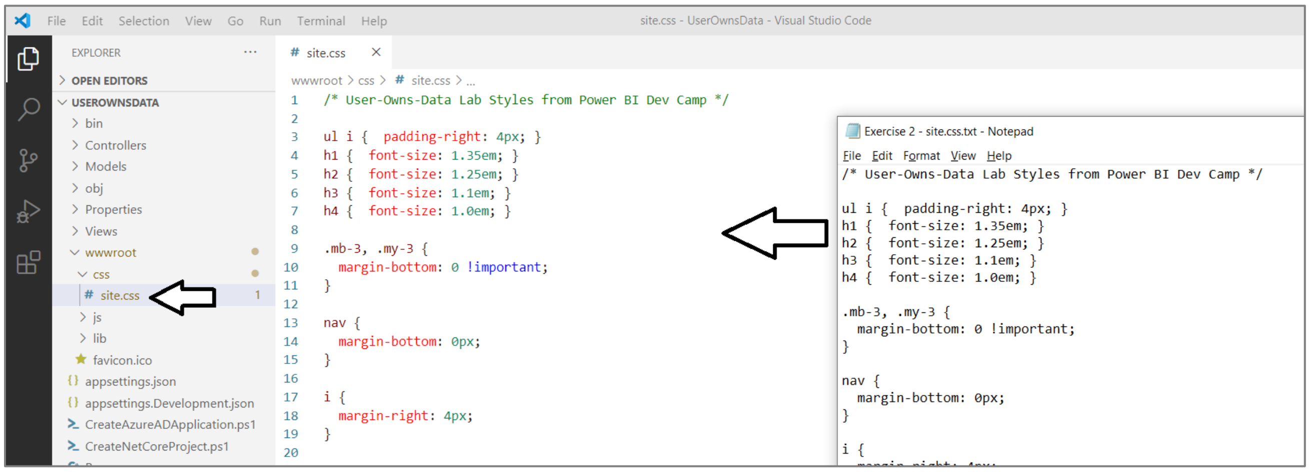


You now got to the point where the **UserOwnsData** web application is up and running and it can successfully authenticate the user. Over the next few steps you will add some custom HTML and CSS styles to make the application a bit more stylish. You will also configure the home page to allow for anonymous access in order to create a better login experience for the application's users.

5. Copy and paste a set of pre-written CSS styles into the **UserOwnsData** project's **site.css** file.
   a) Expand the **wwwroot** folder and then expand the **css** folder inside to examine the contents of the **wwwroot/css** folder.
   b) Open the CSS file named **site.css** and delete any existing content inside.



   c) Using the Windows Explorer, look inside the **StudentLabFiles** folder and locate the file named **Exercise 2 - site.css.txt**.
   d) Open **Exercise 2 - site.css.txt** up in a text editor and copy all of its contents into the Windows clipboard.
   e) Return to Visual Studio Code and paste the contents of the Windows clipboard into **sites.css**.

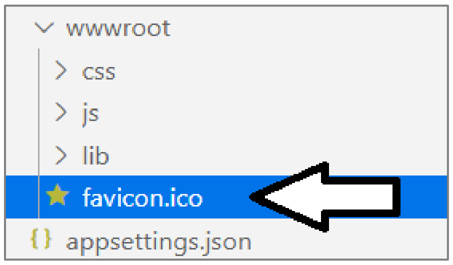

File Edit Selection View Go Run Terminal Help
site.css - UserOwnsData - Visual Studio Code
EXPLORER
OPEN EDITORS
USEROWNSDATA
bin
Controllers
Models
obj
Properties
Views
wwwroot
css
site.css
js
lib
favicon.ico
appsettings.json
appsettings.Development.json
CreateAzureADApplication.ps1
CreateNetCoreProject.ps1
site.css
wwwroot > css > # site.css > ...
Exercise 2 - site.css.txt - Notepad
File Edit Format View Help

```
/* User-Owns-Data Lab Styles from Power BI Dev Camp */

ul i {  padding-right: 4px; }
h1 {  font-size: 1.35em; }
h2 {  font-size: 1.25em; }
h3 {  font-size: 1.1em; }
h4 {  font-size: 1.0em; }

.mb-3, .my-3 {
  margin-bottom: 0 !important;
}

nav {
  margin-bottom: 0px;
}

i {
  margin-right: 4px;
}

```

```
/* User-Owns-Data Lab Styles from Power BI Dev Camp */

ul i {  padding-right: 4px; }
h1 {  font-size: 1.35em; }
h2 {  font-size: 1.25em; }
h3 {  font-size: 1.1em; }
h4 {  font-size: 1.0em; }

.mb-3, .my-3 {
  margin-bottom: 0 !important;
}

nav {
  margin-bottom: 0px;
}

i {
```

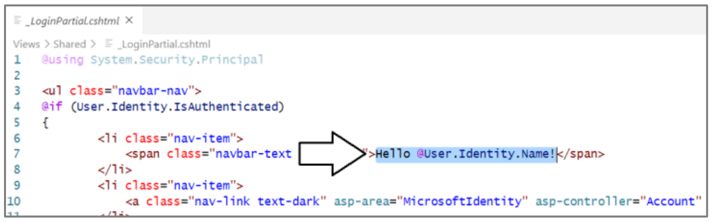
   f) Save your changes and close **site.css**.

6. Copy a custom **favicon.ico** file to the **wwwroot** folder.
   a) Using the Windows Explorer, look inside the **StudentLabFiles** folder and locate the file named **favicon.ico**.
   b) Copy the **favicon.ico** file into the **wwwroot** folder of your project.



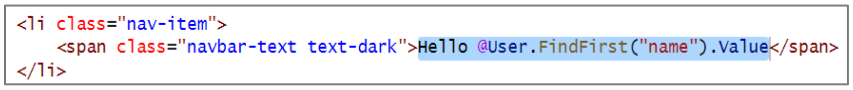
Any file you add the **wwwroot** folder will appear at the root folder of the website created by the **UserOwnsData** project. By adding the **favicon.ico** file, this web application will now display a custom **favicon.ico** in the browser page tab.

7. Modify the partial razor view file named **_LoginPartial.cshtml** to display the user display name instead of the user principal name.
   a) Expand the **Views > Shared** folder and locate the partial view named **_LoginPartial.cshtml**.
   b) Open **_LoginPartial.cshtml** in an editor window.
   c) In the existing code, locate the code **Hello @User.Identity.Name!** which displays a message with the user principal name.



```
@using System.Security.Principal

<ul class="navbar-nav">
@if (User.Identity.IsAuthenticated)
{
        <li class="nav-item">
            <span class="navbar-text ">Hello @User.Identity.Name!</span>
        </li>
        <li class="nav-item">
            <a class="nav-link text-dark" asp-area="MicrosoftIdentity" asp-controller="Account"
```

   d) Replace **Hello @User.Identity.Name!** with **Hello @User.FindFirst("name").Value** as shown in the following screenshot.



```
<li class="nav-item">
    <span class="navbar-text text-dark">Hello @User.FindFirst("name").Value</span>
</li>
```

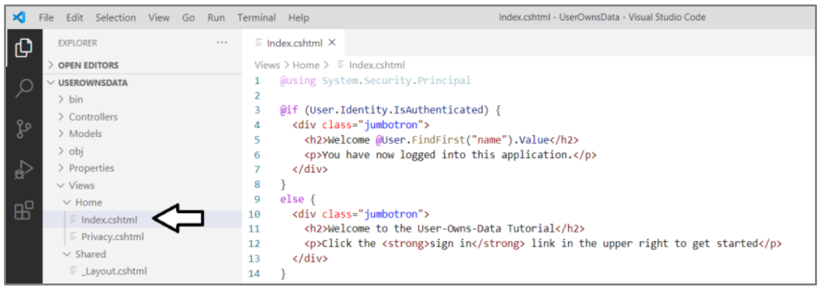
With this update, the application will display the user's display name in the welcome message instead of using the user principal name. In other words, the user greeting will now display a message like **Hello Betty White** instead of **Hello BettyW@Contoso.com!**.

   e) Save your changes and close **_LoginPartial.cshtml**.

8. Modify **Index.cshtml** to display different HTML output depending on whether the user has logged in or not.
   a) Expand the **Views > Home** folder and locate the view file named **Index.cshtml**.
   b) Open **Index.cshtml** in an editor window.
   c) Delete the contents of **Index.cshtml** and replace it with the code shown in the following code listing.

```
@using System.Security.Principal

@if (User.Identity.IsAuthenticated) {
  <div class="jumbotron">
    <h2>Welcome @User.FindFirst("name").Value</h2>
    <p>You have now logged into this application.</p>
  </div>
}
else {
  <div class="jumbotron">
    <h2>Welcome to the User-Owns-Data Tutorial</h2>
    <p>Click the <strong>sign in</strong> link in the upper right to get started</p>
  </div>
}
```

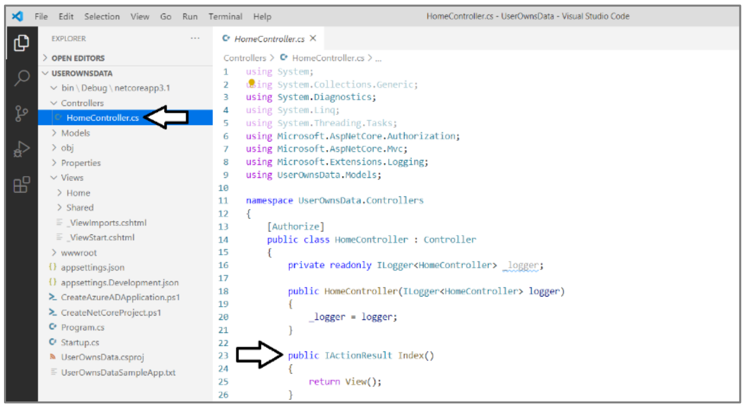
   d) Once you have copied the code from above, save your changes and close **Index.cshtml**.



```
@using System.Security.Principal

@if (User.Identity.IsAuthenticated) {
  <div class="jumbotron">
    <h2>Welcome @User.FindFirst("name").Value</h2>
    <p>You have now logged into this application.</p>
  </div>
}
else {
  <div class="jumbotron">
    <h2>Welcome to the User-Owns-Data Tutorial</h2>
    <p>Click the <strong>sign in</strong> link in the upper right to get started</p>
  </div>
}
```

When you create a new .NET 5 project which supports authentication, the underlying project template creates a home page that requires authentication. To support a more natural login experience for the user, it often makes sense to configure your application so that an anonymous user can access the home page. In the next step you will modify the **Home** controller in order to make the home page accessible to anonymous users.

9. Modify the **Index** action method in **HomeController.cs** to support anonymous access.
    a) Inside the **Controllers** folder, locate **HomeControllers.cs** and open this file in an editor window.
    b) Locate the **Index** method inside the **HomeController** class.


File Edit Selection View Go Run Terminal Help
HomeController.cs - UserOwnsData - Visual Studio Code
EXPLORER
OPEN EDITORS
USEROWNSDATA
bin \ Debug \ netcoreapp3.1
Controllers
HomeController.cs
Models
obj
Properties
Views
Home
Shared
_ViewImports.cshtml
_ViewStart.cshtml
wwwroot
appsettings.json
appsettings.Development.json
CreateAzureADApplication.ps1
CreateNetCoreProject.ps1
Program.cs
Startup.cs
UserOwnsData.csproj
UserOwnsDataSampleApp.txt
HomeController.cs
Controllers > HomeController.cs > ...

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Logging;
using UserOwnsData.Models;

namespace UserOwnsData.Controllers
{
    [Authorize]
    public class HomeController : Controller
    {
        private readonly ILogger<HomeController> _logger;

        public HomeController(ILogger<HomeController> logger)
        {
            _logger = logger;
        }

        public IActionResult Index()
        {
            return View();
        }
```

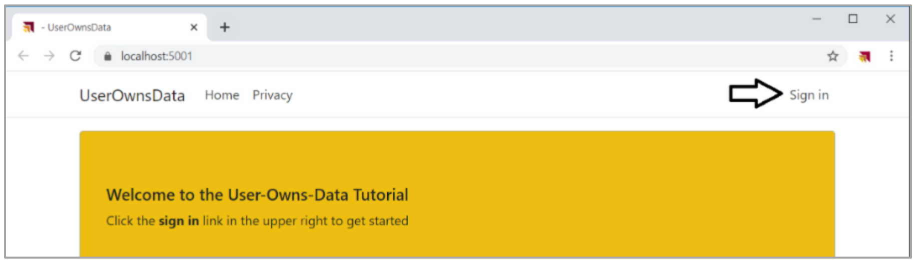
    c) Add the **[AllowAnonymous]** attribute to the **Index** method as shown in the following code listing.

```
[AllowAnonymous]
public IActionResult Index()
{
    return View();
}
```

    d) Save your changes and close **HomeController.cs**.

It's time again to test this web application in the .NET debugger so you can see the effects of your changes. In the next step, you will start the debugger so you can start up the web application and test the user authentication experience in the browser.

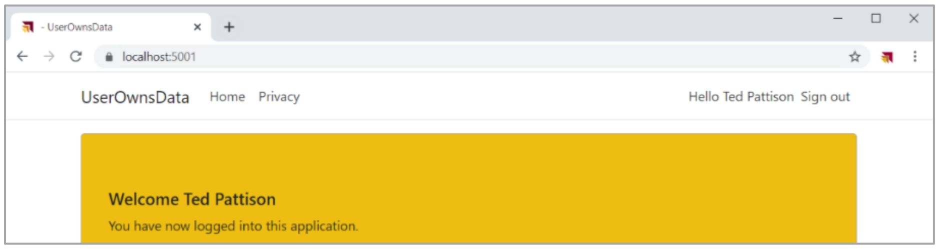
10. Test the **UserOwnsData** project by running it in the .NET debugging environment.
    a) Start the .NET debugger by selecting **Run > Start Debugging** or by pressing the **{F5}** keyboard shortcut.
    b) Once the debugging session has initialized, the browser should display the home page using anonymous access.
    c) Click the **Sign in** link to test the user experience when authenticating with Azure AD.



Welcome to the User-Owns-Data Tutorial

Click the **sign in** link in the upper right to get started

    d) Once you've signed in, you should be able to see the user display name in the welcome message in the upper right corner.

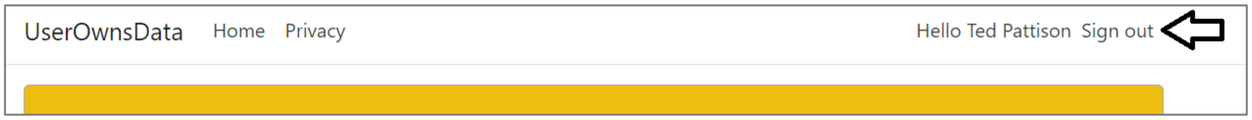


Welcome Ted Pattison

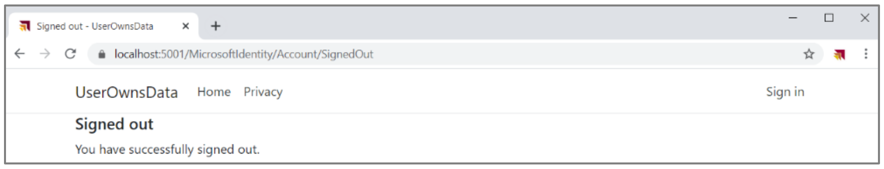
You have now logged into this application.

If the web page does not appear with a yellow background as shown in the screenshot above, it's possible your browser has cached the original version of the **site.css** file. If this is the case, you must clear the browser cache so it loads the latest version of **site.css**.

11. Test the user experience for logging out.
    a) Click the **Sign out** link to begin the logout experience.



    b) After logging out, you'll be directed to the **Microsoft.Identity.Web** logout page at **/MicrosoftIdentity/Account/SignedOut**.


Signed out - UserOwnsData
localhost:5001/MicrosoftIdentity/Account/SignedOut
UserOwnsData
Home
Privacy
Sign in

Signed out

You have successfully signed out.

    c) You're done testing. Close the browser, return to Visual Studio Code and stop the debug session using the debug toolbar.

In the next step, you will add a new controller action and view named **Embed**. However, instead of creating a new controller action and view, you will simply the rename the controller action and view named **Privacy** that were automatically added by the project template.

12. Create a new controller action named **Embed**.
    a) Locate the **HomeController.cs** file in the **Controllers** folder and open it in an editor window.
    b) Look inside the **HomeController** class and locate the method named **Privacy**.

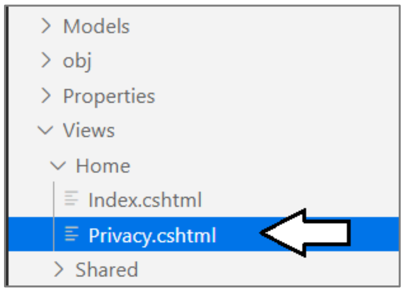
```
[AllowAnonymous]
public IActionResult Index() {
    return View();
}

public IActionResult Privacy() {
    return View();
}
```

    c) Rename of the **Privacy** method to **Embed**. No changes to the method body are required.

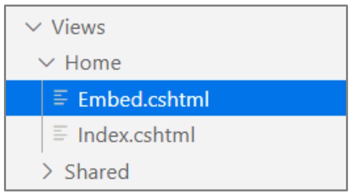
```
[AllowAnonymous]
public IActionResult Index() {
    return View();
}

public IActionResult Embed() {
    return View();
}
```

Note that, unlike the **Index** method, the **Embed** method does not have the **AllowAnonymous** attribute. That means only authenticated users will be able to navigate to this page. One really nice aspect of the ASP.NET MVC architecture is that it will automatically trigger an interactive login whenever an anonymous user attempts to navigate to a secured page such as **Embed**.

13. Create a new MVC view for the **Home** controller named **Embed.cshtml**.
    a) Look inside the **Views > Home** folder and locate the razor view file named **Privacy.cshtml**.



b) Rename the **Privacy.cshtml** razor file to **Embed.cshtml**..



c) Open **Embed.cshtml** in a code editor.

d) Delete the existing contents of **Embed.cshtml** and replace it with the follow line of HTML code.

```
<h2>TODO: Embed Report Here</h2>
```

e) Save your changes and close **Embed.cshtml**.

In a standard .NET 5 web application that uses MVC, there is a shared page layout defined in a file named **_Layouts.cshtml** which is located in the **Views > Shared** folder. In the next step you will modify the shared layout in the **_Layouts.cshtml** file so that you can add a link to the **Embed** page into the top navigation menu.

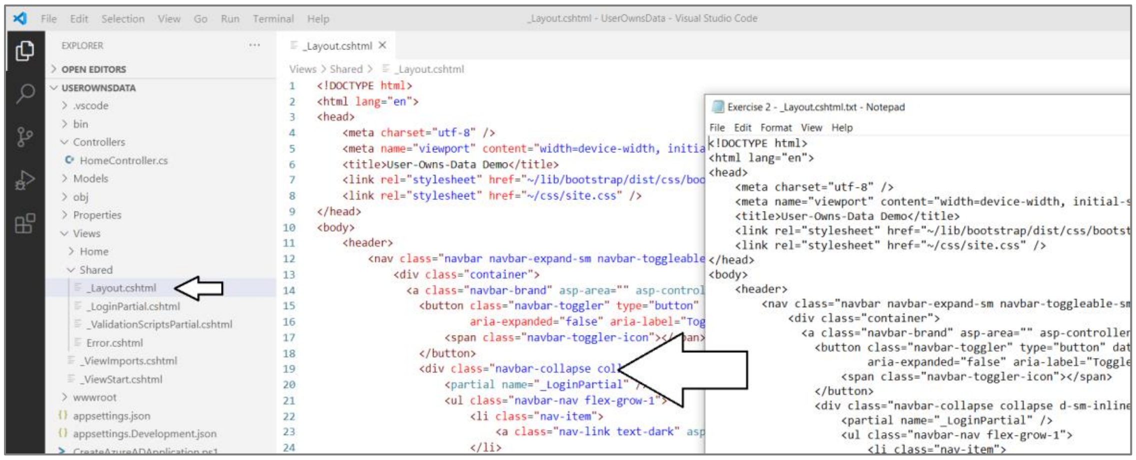
14. Modify the shared layout in **_Layout.cshtml** to include a link to the **Embed** page.

a) Inside the **Views > Shared** folder, locate **_Layouts.cshtml** and open this shared view file in an editor window.

b) Using Windows Explorer, look inside the **StudentLabFiles** folder and locate the file named **_Layout.cshtml**.

c) Open **Exercise 2 - _Layout.cshtml.txt** in the **StudentLabFiles** folder copy its contents to the Windows clipboard.

d) Return to Visual Studio Code and paste the contents of the Windows clipboard into the **_Layouts.cshtml** file.


File Edit Selection View Go Run Terminal Help
_Layout.cshtml - UserOwnsData - Visual Studio Code
EXPLORER
OPEN EDITORS
USEROWNSDATA
.vscode
bin
Controllers
HomeController.cs
Models
obj
Properties
Views
Home
Shared
_Layout.cshtml
_LoginPartial.cshtml
_ValidationScriptsPartial.cshtml
Error.cshtml
_ViewImports.cshtml
_ViewStart.cshtml
wwwroot
appsettings.json
appsettings.Development.json
_Layout.cshtml
Views > Shared > _Layout.cshtml
Exercise 2 - _Layout.cshtml.txt - Notepad
File Edit Format View Help

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initia
    <title>User-Owns-Data Demo</title>
    <link rel="stylesheet" href="~/lib/bootstrap/dist/css/boo
    <link rel="stylesheet" href="~/css/site.css" />
</head>
<body>
    <header>
        <nav class="navbar navbar-expand-sm navbar-toggleable
            <div class="container">
                <a class="navbar-brand" asp-area="" asp-control
                  <button class="navbar-toggler" type="button"
                          aria-expanded="false" aria-label="Tog
                      <span class="navbar-toggler-icon"></span>
                  </button>
                  <div class="navbar-collapse col
                      <partial name="_LoginPartial" />
                      <ul class="navbar-nav flex-grow-1">
                          <li class="nav-item">
                              <a class="nav-link text-dark" asp
                          </li>
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-s
    <title>User-Owns-Data Demo</title>
    <link rel="stylesheet" href="~/lib/bootstrap/dist/css/bootst
    <link rel="stylesheet" href="~/css/site.css" />
</head>
<body>
    <header>
        <nav class="navbar navbar-expand-sm navbar-toggleable-sm
            <div class="container">
                <a class="navbar-brand" asp-area="" asp-controller
                  <button class="navbar-toggler" type="button" dat
                          aria-expanded="false" aria-label="Toggle
                      <span class="navbar-toggler-icon"></span>
                  </button>
                  <div class="navbar-collapse collapse d-sm-inline
                      <partial name="_LoginPartial" />
                      <ul class="navbar-nav flex-grow-1">
                          <li class="nav-item">
```

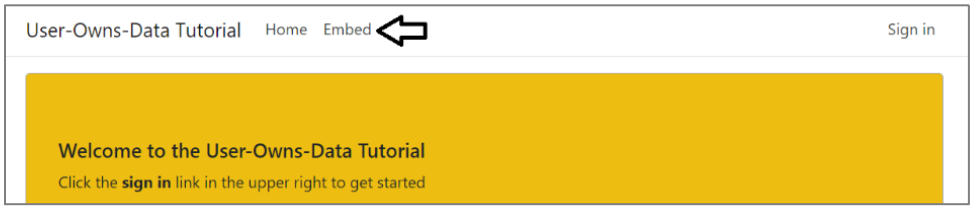
e) Save your changes and close **_Layouts.cshtml**

15. Run the web application in the Visual Studio Code debugger to test the new **Embed** page.

a) Start the Visual Studio Code debugger by selecting **Run > Start Debugging** or by pressing the **{F5}** keyboard shortcut.

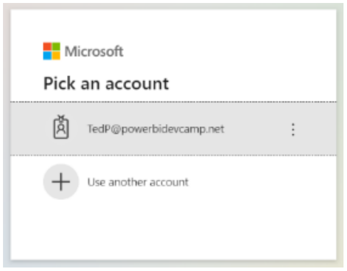
b) The **UserOwnsData** web application should display the home page as shown to an anonymous user.

c) Click on the **Embed** link in the top nav menu to navigate to the **Embed** page.

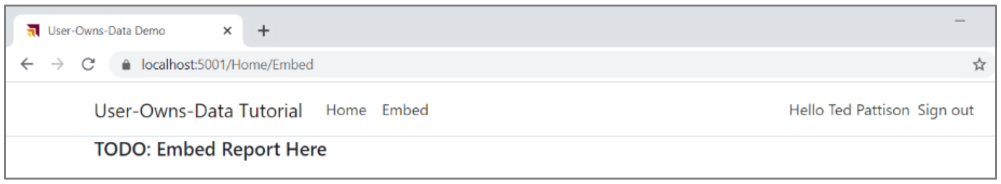

User-Owns-Data Tutorial
Home
Embed
Sign in
Welcome to the User-Owns-Data Tutorial
Click the sign in link in the upper right to get started

When you attempt to navigate to the **Embed** page as an anonymous user, you'll be automatically prompted to log in.

d) Log in using your user name and password.



e) Once you have logged in, you should be automatically redirected to the **Embed** page.



TODO: Embed Report Here

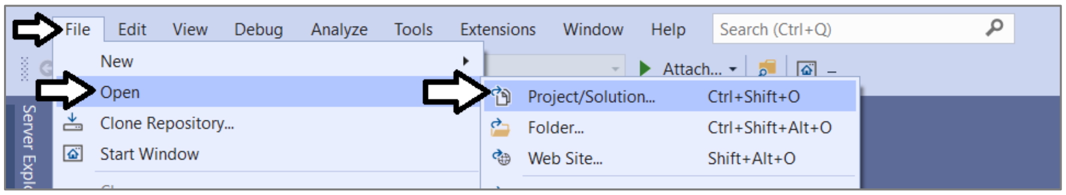
f) You're done testing. Close the browser, return to Visual Studio Code and stop the debug session using the debug toolbar.

The next step is an *optional step* for those campers that prefer developing with Visual Studio 2019 instead of Visual Studio Code. If you are happy developing with Visual Studio Code and are not interested in developing .NET 5 projects using Visual Studio 2019, you can skip the next step and move ahead to *Exercise 3: Call the Power BI Service API*.

16. Open and test the **UserOwnsData** project using Visual Studio 2019.

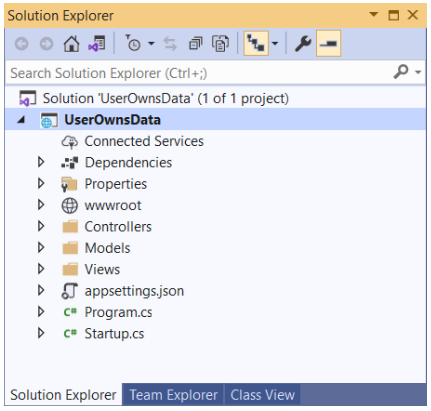
a) Launch Visual Studio 2019 – You can use any edition including the Enterprise edition, Pro edition or Community edition.

b) From the **File** menu, select the **Open > Project/Solution…** command.



c) In the **Open Project/Solution** dialog, select the **UserOwnsData.csproj** file in the **UserOwnsData** folder and click **Open**.

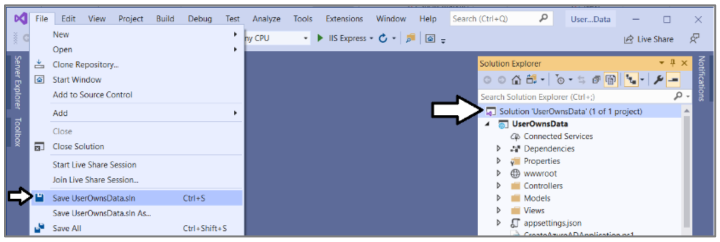
d) The **UserOwnsData** project should now be open in Visual Studio 2019 as shown in the following screenshot.



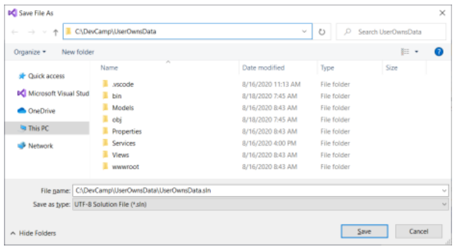
There is one big difference between developing with Visual Studio Code and Visual Studio 2019. Visual Studio Code only requires project files (*.csproj). However, Visual Studio 2019 requires that you work with both project files and solution files (*.sln). In the next step you will save a new project file for the **UserOwnsData** solution to make it easier to develop this project with Visual Studio 2019.

e) In the **Solution Explorer** on the right, select the top node in the tree with the caption **Solution "UserOwnsData"**.

f) From the **File** menu, select the **Save UserOwnsData.sln** menu command.



g) Save the solution file **UserOwnsData.sln** in the **UserOwnsData** project folder



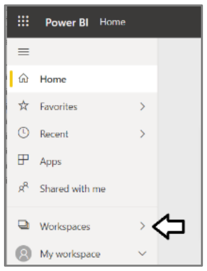
Remember that the **UserOwnsData.sln** file is only used by Visual Studio 2019 and it not used at all in Visual Studio Code.

The following lab exercises will go back to developing with Visual Studio Code. However, you can use whichever IDE you like better. Also, you can and open and test all the lab solutions using either Visual Studio Code or Visual Studio 2019.

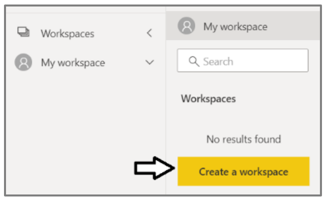
Exercise 3: Call the Power BI Service API

In this exercise, you will begin your work by creating a new Power BI workspace and importing a PBIX file to that you have a report for testing purposes. After that, you will add support to the **UserOwnsData** web application to acquire access tokens from Azure AD and to call the Power BI Service API. By the end of this exercise, your code will be able to call to the Power BI Service API to retrieve data about a report required for embedding.

1. Using the browser, log into the Power BI Service in the Microsoft 365 tenant serving as your Power BI development environment.
   a) Navigate the Power BI portal at https://app.powerbi.com and if prompted, log in using your credentials.
2. Create a new app workspace named **Dev Camp Demos**.
   a) Click the **Workspace** flyout menu in the left navigation.

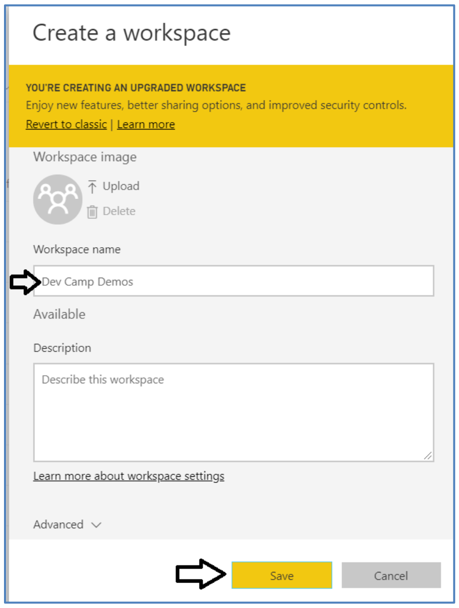


b) Click the **Create app workspace** button to display the **Create an app workspace** dialog.



c) In the **Create an app workspace** pane, enter a workspace name such as **Dev Camp Demos**.

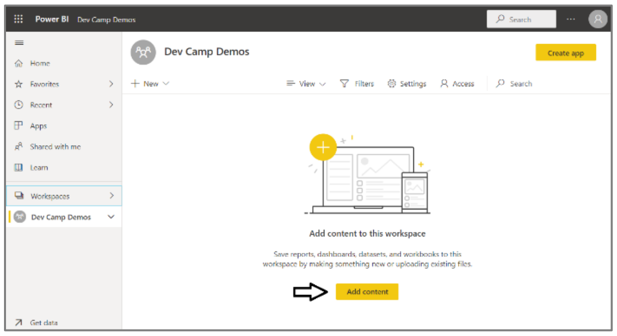
d) Click the **Save** button to create the new app workspace.


Create a workspace

YOU'RE CREATING AN UPGRADED WORKSPACE
Enjoy new features, better sharing options, and improved security controls.
Revert to classic | Learn more

Workspace name: Dev Camp Demos
Available

e) When you click **Save**, the Power BI service should create the new app workspace and then switch your current Power BI session to be running within the context of the new **Dev Camp Demos** workspace.

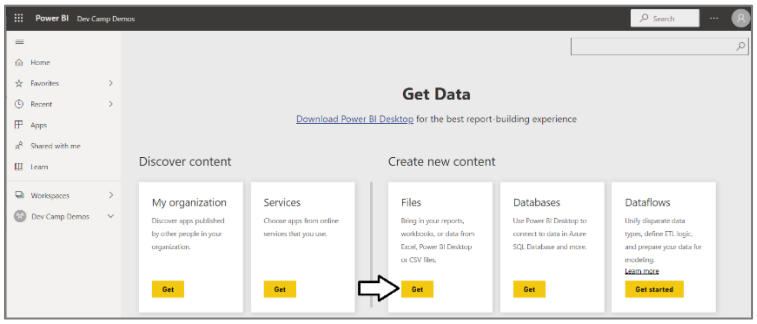


Now that you have created the new app workspace, the next step is to upload a PBIX project file created with Power BI Desktop. You are free to use your own PBIX file as long as the PBIX file does not have row-level security (RLS) enabled. If you don't have your own PBIX file, you can download the sample PBIX file named **COVID-19 US.pbix** and use that instead.

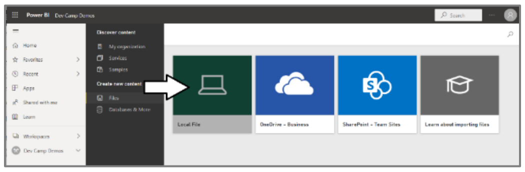
3. Upload a PBIX file to create a new report and dataset.

   a) Click **Add content** to navigate to the **Get Data** page.

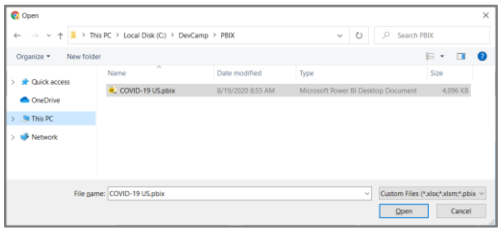
   b) Click the **Get** button in the **Files** section.


Power BI Dev Camp Demos
Search
Home
Favorites
Recent
Apps
Shared with me
Learn
Workspaces
Dev Camp Demos
Get Data
Download Power BI Desktop for the best report-building experience
Discover content
Create new content
My organization
Discover apps published by other people in your organization.
Get
Services
Choose apps from online services that you use.
Get
Files
Bring in your reports, workbooks, or data from Excel, Power BI Desktop or CSV files.
Get
Databases
Use Power BI Desktop to connect to data in Azure SQL Database and more.
Get
Dataflows
Unify disparate data types, define ETL logic, and prepare your data for modeling.
Learn more
Get started

c) Click on **Local File** in order to select a PBIX file that you have on your local hard drive.

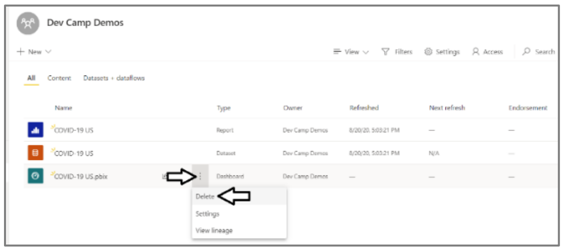


d) Select the PBIX file and click the **Open** button to upload it to the Power BI Service.


Open
This PC › Local Disk (C:) › DevCamp › PBIX
COVID-19 US.pbix
File name: COVID-19 US.pbix
Custom Files (*.xlsx;*.xlsm;*.pbix)
Open
Cancel

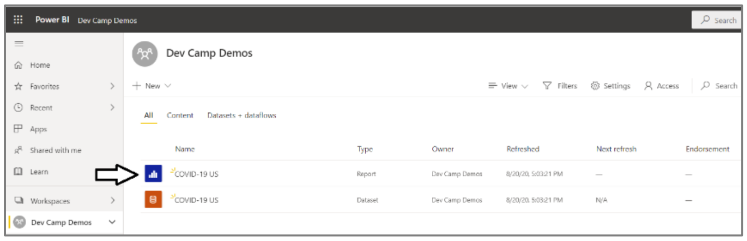
e) The Power BI Service should have created a report and a dataset from the PBIX file you uploaded.

f) If the Power BI Service created a dashboard as well, delete this dashboard as you will not need it.

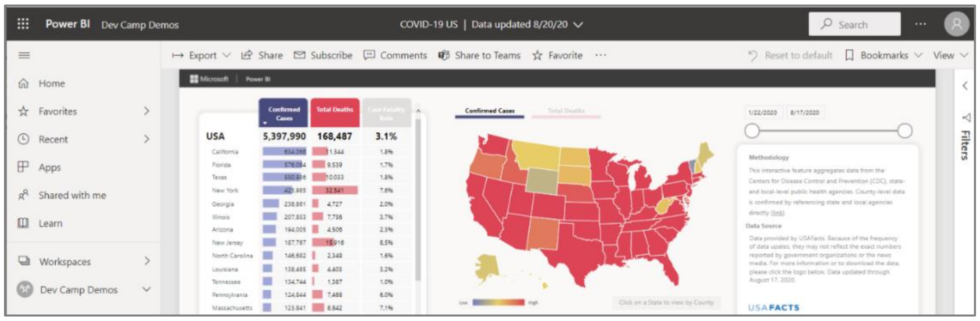


4. Open the report to see what it looks like when displayed in the Power BI Service.

a) Click on the report to open it.

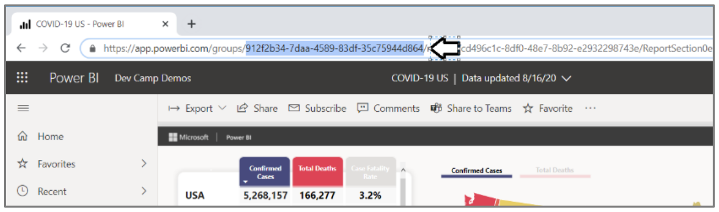


b) You should now be able to see the report.


Power BI Dev Camp Demos
COVID-19 US | Data updated 8/20/20
Confirmed Cases
Total Deaths
USA 5,397,990 168,487 3.1%
USAFACTS

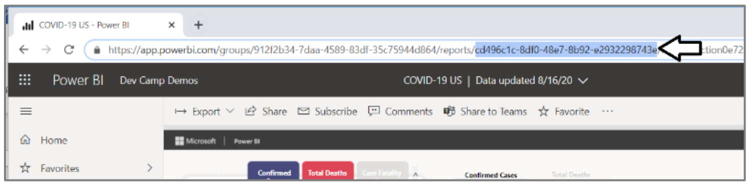
In the next step, you will find and record the GUID-based IDs for the report and its hosting workspace. You will then use these IDs later in this exercises when you first write the code to embed a report in the **UserOwnsData** web application.

5. Get the Workspace ID and the Report ID from the report URL.
   a) Locate and copy the app workspace ID from the report URL by copying the GUID that comes after **/groups/**.



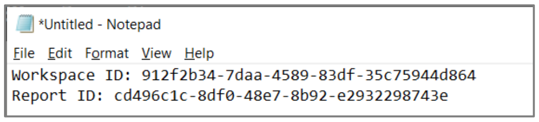
```
https://app.powerbi.com/groups/912f2b34-7daa-4589-83df-35c75944d864/r...cd496c1c-8df0-48e7-8b92-e2932298743e/ReportSection0e
```

   b) Open up a new text file in a program such as Notepad and paste in the value of the workspace ID.
   c) Locate and copy the report ID from the URL by copying the GUID that comes after **/reports/**.



```
https://app.powerbi.com/groups/912f2b34-7daa-4589-83df-35c75944d864/reports/cd496c1c-8df0-48e7-8b92-e2932298743e...ction0e72
```

   d) Copy the report ID into the text file Notepad.

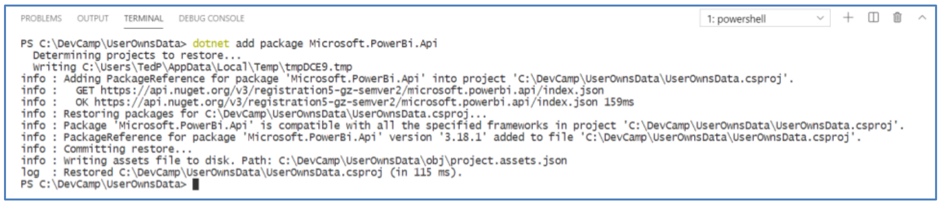


```
Workspace ID: 912f2b34-7daa-4589-83df-35c75944d864
Report ID: cd496c1c-8df0-48e7-8b92-e2932298743e
```

Leave the text file open for now. In a step later in this exercise, you will copy and paste these IDs into your C# code.

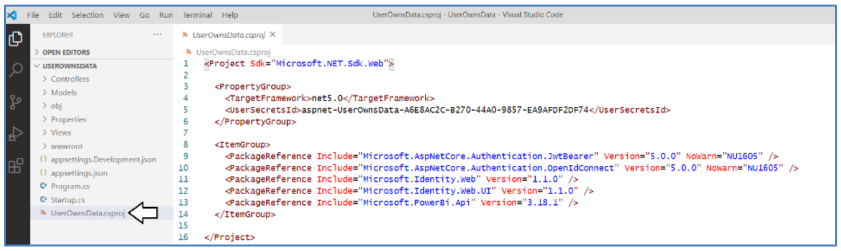
6. Add the NuGet package for the **Power BI .NET SDK**.
   a) Move back to the Terminal so you can execute another dotnet CLI command.
   b) Type and execute the following **dotnet add package** command to add the NuGet package for the **Power BI .NET SDK**.

```
dotnet add package Microsoft.PowerBi.Api
```



```
PS C:\DevCamp\UserOwnsData> dotnet add package Microsoft.PowerBi.Api
  Determining projects to restore...
  Writing C:\Users\TedP\AppData\Local\Temp\tmpDCE9.tmp
info : Adding PackageReference for package 'Microsoft.PowerBi.Api' into project 'C:\DevCamp\UserOwnsData\UserOwnsData.csproj'.
info :   GET https://api.nuget.org/v3/registration5-gz-semver2/microsoft.powerbi.api/index.json
info :   OK https://api.nuget.org/v3/registration5-gz-semver2/microsoft.powerbi.api/index.json 159ms
info : Restoring packages for C:\DevCamp\UserOwnsData\UserOwnsData.csproj...
info : Package 'Microsoft.PowerBi.Api' is compatible with all the specified frameworks in project 'C:\DevCamp\UserOwnsData\UserOwnsData.csproj'.
info : PackageReference for package 'Microsoft.PowerBi.Api' version '3.18.1' added to file 'C:\DevCamp\UserOwnsData\UserOwnsData.csproj'.
info : Committing restore...
info : Writing assets file to disk. Path: C:\DevCamp\UserOwnsData\obj\project.assets.json
log  : Restored C:\DevCamp\UserOwnsData\UserOwnsData.csproj (in 115 ms).
PS C:\DevCamp\UserOwnsData>
```

   c) Open the **UserOwnsData.csproj** file. You should now see this file contains a package reference to **Microsoft.PowerBi.Api**.

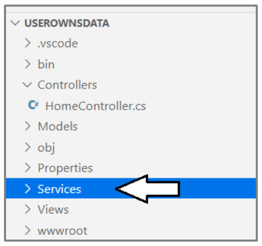


```
<Project Sdk="Microsoft.NET.Sdk.Web">

  <PropertyGroup>
    <TargetFramework>net5.0</TargetFramework>
    <UserSecretsId>aspnet-UserOwnsData-A6E8AC2C-B270-44A0-9857-EA9AFDF2DF74</UserSecretsId>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.AspNetCore.Authentication.JwtBearer" Version="5.0.0" NoWarn="NU1605" />
    <PackageReference Include="Microsoft.AspNetCore.Authentication.OpenIdConnect" Version="5.0.0" NoWarn="NU1605" />
    <PackageReference Include="Microsoft.Identity.Web" Version="1.1.0" />
    <PackageReference Include="Microsoft.Identity.Web.UI" Version="1.1.0" />
    <PackageReference Include="Microsoft.PowerBi.Api" Version="3.18.1" />
  </ItemGroup>

</Project>
```

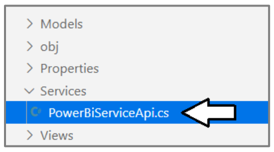
   d) Close the the **UserOwnsData.csproj** file without saving any changes.

Your project now includes the NuGet package for the Power BI .NET SDK so you can begin to program against the classes from this package in the **Microsoft.PowerBI.Api** namespace.

7. Create a new C# class named **PowerBiServiceApi** in which you will add code for calling the Power BI Service API.
   a) Return to the **UserOwnsData** project in Visual Studio Code.
   b) Create a new top-level folder in the **UserOwnsData** project named **Services**.



   c) Inside the **Services** folder, create a new C# source file named **PowerBiServiceApi.cs**.



   d) Copy and paste the following code into **PowerBiServiceApi.cs** to provide a starting point.

```
using System;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.Extensions.Configuration;
using Microsoft.Identity.Web;
using Microsoft.Rest;
using Microsoft.PowerBI.Api;
using Newtonsoft.Json;

namespace UserOwnsData.Services {

  public class EmbeddedReportViewModel {
    //TODO: implement this class
  }

  public class PowerBiServiceApi {
    //TODO: implement this class
  }

}
```

   e) Implement the **EmbeddedReportViewModel** class using the following code.

```
public class EmbeddedReportViewModel {
  public string Id;
  public string Name;
  public string EmbedUrl;
  public string Token;
}
```

The **EmbeddedReportViewModel** class is designed as a view model which defines the structure the embedding data required for a Power BI report. You will use this class later in this lab to pass embedding data for a report from an MVC controller to an MVC view.

   f) Begin your implementation by adding two private fields named **tokenAcquisition** and **urlPowerBiServiceApiRoot**.

```
public class PowerBiServiceApi {

  private ITokenAcquisition tokenAcquisition { get; }
  private string urlPowerBiServiceApiRoot { get; }

}
```

g) Add the following constructor to initialize the two private fields named **tokenAcquisition** and **urlPowerBiServiceApiRoot**.

```
public class PowerBiServiceApi {

  private ITokenAcquisition tokenAcquisition { get; }
  private string urlPowerBiServiceApiRoot { get; }

  public PowerBiServiceApi(IConfiguration configuration, ITokenAcquisition tokenAcquisition) {
    this.urlPowerBiServiceApiRoot = configuration["PowerBi:ServiceRootUrl"];
    this.tokenAcquisition = tokenAcquisition;
  }

}
```

This code uses the .NET dependency injection (DI) pattern. When your class needs to use a service, you can simply add a constructor parameter based on the type for that service and the .NET runtime takes care of passing the service instance at run time. In this case, the constructor is injecting an instance of the .NET configuration service using the **IConfiguration** parameter which is used to retrieve the **PowerBi:ServiceRootUrl** configuration value from **appsettings.json**. The **ITokenAcquisition** parameter which is named **tokenAcquisition** holds a reference to the Microsoft authentication service provided by the **Microsoft.Identity.Web** library and will be used to acquire access tokens from Azure AD.

h) Place your cursor at the bottom of the **PowerBiServiceApi** class and add another new line so you can add more members.

i) At the bottom off the **PowerBiServiceApi** class, add the following static read-only field named **RequiredScopes**.

```
public static readonly string[] RequiredScopes = new string[] {
  "https://analysis.windows.net/powerbi/api/Group.Read.All",
  "https://analysis.windows.net/powerbi/api/Report.ReadWrite.All",
  "https://analysis.windows.net/powerbi/api/Dataset.ReadWrite.All",
  "https://analysis.windows.net/powerbi/api/Content.Create",
  "https://analysis.windows.net/powerbi/api/Workspace.ReadWrite.All"
};
```

The **RequiredScopes** field is a string array with a set of delegated permissions supported by the Power BI Service API. Your application will pass these permissions when it calls to Azure AD to acquire an access token.

j) Move down in the **PowerBiServiceApi** class below the **RequiredScopes** field and add the **GetAccessToken** method.

```
public string GetAccessToken() {
  return this.tokenAcquisition.GetAccessTokenForUserAsync(RequiredScopes).Result;
}
```

k) Move down below the **GetAccessToken** method and add the **GetPowerBiClient** method.

```
public PowerBIClient GetPowerBiClient()     {
  var tokenCredentials = new TokenCredentials(GetAccessToken(), "Bearer");
  return new PowerBIClient(new Uri(urlPowerBiServiceApiRoot), tokenCredentials);
}
```

l) Move down below the **GetPowerBiClient** method and add the **GetReport** method.

```
public async Task<EmbeddedReportViewModel> GetReport(Guid WorkspaceId, Guid ReportId) {

  PowerBIClient pbiClient = GetPowerBiClient();

  // call to Power BI Service API to get embedding data
  var report = await pbiClient.Reports.GetReportInGroupAsync(WorkspaceId, ReportId);

  // return report embedding data to caller
  return new EmbeddedReportViewModel {
    Id = report.Id.ToString(),
    EmbedUrl = report.EmbedUrl,
    Name = report.Name,
    Token = GetAccessToken()
  };
}
```

m) Save and close **PowerBIServiceApi.cs**.

Note that **Exercise 3 - PowerBiServiceApi.cs.txt** in the **StudentLabFiles** folder contains the final code for **PowerBiServiceApi.cs**.

8. Modify the code in **Startup.cs** to properly register the services required for user authentication and access token acquisition.
   a) Open the **Startup.cs** file in an editor window.
   b) Underneath the existing **using** statements, add the following **using** statement;

```
using UserOwnsData.Services;
```

   c) Look inside the **ConfigureServices** method and locate the following line of code.

```
public void ConfigureServices(IServiceCollection services) {

  services.AddMicrosoftIdentityWebAppAuthentication(Configuration);
```

   d) Replace the call to **services.AddMicrosoftIdentityWebAppAuthentication** with the following code.

```
services
  .AddMicrosoftIdentityWebAppAuthentication(Configuration)
  .EnableTokenAcquisitionToCallDownstreamApi(PowerBiServiceApi.RequiredScopes)
  .AddInMemoryTokenCaches();
```

   e) Move below the call to **AddInMemoryTokenCaches** and add the following code.

```
services.AddScoped(typeof(PowerBiServiceApi));
```

   f) At this point, the **ConfigureService** method in **Startup.cs** should match the following code listing.

```
public void ConfigureServices(IServiceCollection services) {

  services
    .AddMicrosoftIdentityWebAppAuthentication(Configuration)
    .EnableTokenAcquisitionToCallDownstreamApi(PowerBiServiceApi.RequiredScopes)
    .AddInMemoryTokenCaches();

  services.AddScoped(typeof(PowerBiServiceApi));

  var mvcBuilder = services.AddControllersWithViews(options => {
    var policy = new AuthorizationPolicyBuilder()
        .RequireAuthenticatedUser()
        .Build();
    options.Filters.Add(new AuthorizeFilter(policy));
  });

  mvcBuilder.AddMicrosoftIdentityUI();

  services.AddRazorPages();
}
```

The code in **ConfigureServices** accomplishes several important things. The call to **AddMicrosoftWebAppCallsWebApi** configures the Microsoft authentication library to acquire access tokens. Next, the call to **AddInMemoryTokenCaches** configures a token cache that the Microsoft authentication library will use to cache access tokens and refresh tokens behind the scenes. Finally, the call to **services.AddScoped(typeof(PowerBiServiceApi))** configures the **PowerBiServiceApi** class as a service class that can be added to other classes using the dependency injection (DI) pattern.

9. Modify the **HomeController** class to program against the **PowerBiServiceApi** class.
   a) Inside the **Controllers** folder, locate **HomeController.cs** and open it in an editor window.
   b) Underneath the existing **using** statements, add a **using** statement to import the **UserOwnsData.Services** namespace.

```
using UserOwnsData.Services;
```

   c) At the top of the **HomeController** class locate the **_logger** field and the constructor as shown in the following code listing.

```
[Authorize]
public class HomeController : Controller {

  private readonly ILogger<HomeController> _logger;

  public HomeController(ILogger<HomeController> logger) {
    _logger = logger;
  }
```

d) Remove the **_logger** field and the existing constructor and replace them with the following code.

```
[Authorize]
public class HomeController : Controller {

  private PowerBiServiceApi powerBiServiceApi;

  public HomeController(PowerBiServiceApi powerBiServiceApi) {
    this.powerBiServiceApi = powerBiServiceApi;
  }
```

This is another example of using dependency injection. Since you registered the **PowerBiServiceApi** class as a service by calling **services.AddScoped** in the **ConfigureServices** method, you can simply add a **PowerBiServiceApi** parameter to the constructor and the .NET 5 runtime will take care of creating a **PowerBiServiceApi** instance and passing it to the constructor.

e) Locate the **Embed** method implementation in the **HomeController** class and replace it with the following code.

```
public async Task<IActionResult> Embed() {

  // replace these two GUIDs with the workspace ID and report ID you recorded earlier
  Guid workspaceId = new Guid("11111111-1111-1111-1111-111111111111");
  Guid reportId = new Guid("22222222-2222-2222-2222-222222222222");

  var viewModel = await powerBiServiceApi.GetReport(workspaceId, reportId);

  return View(viewModel);
}
```

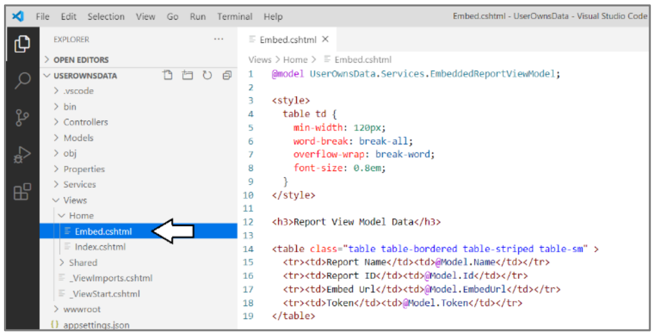
10. Modify the HTML and razor code in the view file named **Embed.cshtml**.

a) Locate the **Embed.cshtml** razor file inside the **Views > Home** folder and open this file in an editor window.

b) Delete the contents of **Embed.cshtml** and replace it with the following code which creates a table to display report data.

```
@model UserOwnsData.Services.EmbeddedReportViewModel;

<style>
  table td {
    min-width: 120px;
    word-break: break-all;
    overflow-wrap: break-word;
    font-size: 0.8em;
  }
</style>

<h3>Report View Model Data</h3>

<table class="table table-bordered table-striped table-sm" >
  <tr><td>Report Name</td><td>@Model.Name</td></tr>
  <tr><td>Report ID</td><td>@Model.Id</td></tr>
  <tr><td>Embed Url</td><td>@Model.EmbedUrl</td></tr>
  <tr><td>Token</td><td>@Model.Token</td></tr>
</table>
```

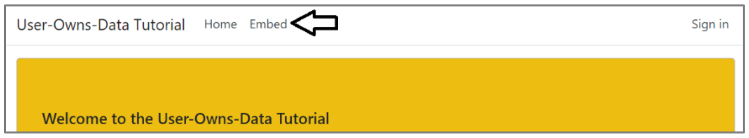
c) The code in **Embed.cshtml** should now match the following screenshot..



```
@model UserOwnsData.Services.EmbeddedReportViewModel;

<style>
  table td {
    min-width: 120px;
    word-break: break-all;
    overflow-wrap: break-word;
    font-size: 0.8em;
  }
</style>

<h3>Report View Model Data</h3>

<table class="table table-bordered table-striped table-sm" >
  <tr><td>Report Name</td><td>@Model.Name</td></tr>
  <tr><td>Report ID</td><td>@Model.Id</td></tr>
  <tr><td>Embed Url</td><td>@Model.EmbedUrl</td></tr>
  <tr><td>Token</td><td>@Model.Token</td></tr>
</table>
```

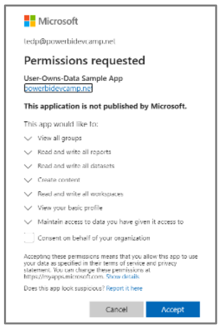
d) Save your changes and close **Embed.cshtml**.

11. Run the web application in the Visual Studio Code debugger to test the new **Embed** page.
    a) Start the Visual Studio Code debugger by selecting **Run > Start Debugging** or by pressing the **{F5}** keyboard shortcut.
    b) The **UserOwnsData** web application should display the home page as shown to an anonymous user.
    c) Click on the **Embed** link in the top nav menu to navigate to the **Embed** page.



Welcome to the User-Owns-Data Tutorial

    d) If you are prompted to enter your credentials, enter your user name and password and log in.
    e) After you have authenticated for the first time, you should be prompted with a **Permissions Requested** dialog.
    f) Click the **Accept** button to consent to the application using the requested delegated permissions on your behalf.



tedp@powerbidevcamp.net

Permissions requested

User-Owns-Data Sample App
powerbidevcamp.net

This application is not published by Microsoft.

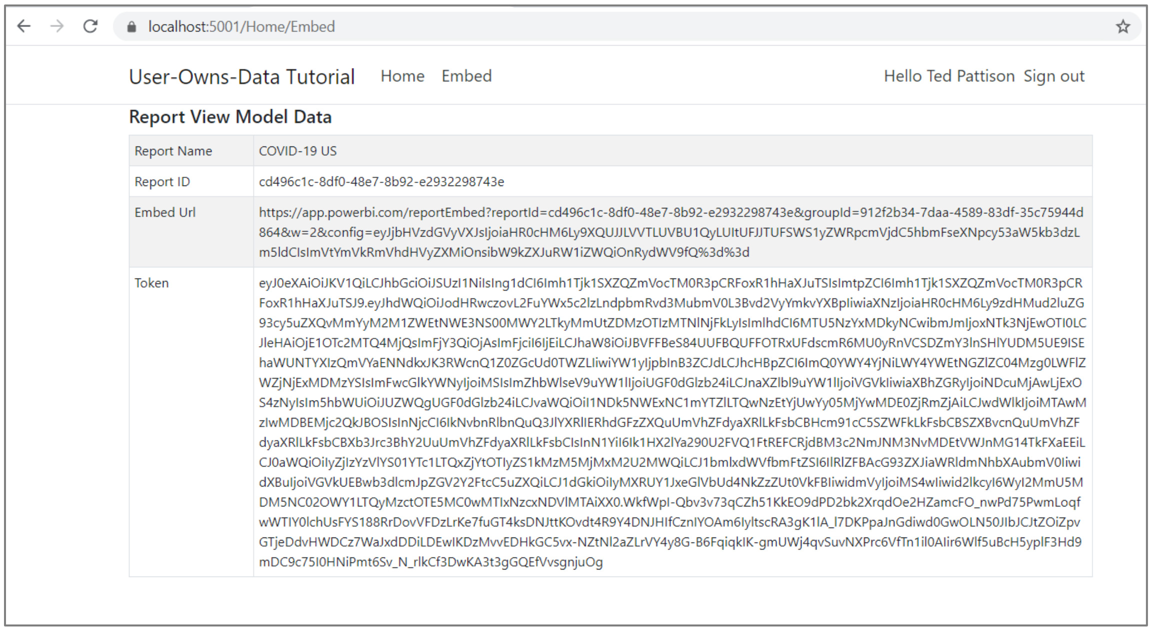
This app would like to:
- View all groups
- Read and write all reports
- Read and write all datasets
- Create content
- Read and write all workspaces
- View your basic profile
- Maintain access to data you have given it access to
- Consent on behalf of your organization

Accepting these permissions means that you allow this app to use your data as specified in their terms of service and privacy statement. You can change these permissions at https://myapps.microsoft.com. Show details

Does this app look suspicious? Report it here

The **Permissions requested** dialog is only shown to each user during the first successful login. Once a user clicks **Accept**, Azure AD remembers that the user has consented to the required permissions and does not need to prompt the user about permission requests.

    g) Once you navigate to the **Embed** page, it should display a table containing the embedding data for your Power BI report.



Report View Model Data

Report Name	COVID-19 US
Report ID	cd496c1c-8df0-48e7-8b92-e2932298743e
Embed Url	https://app.powerbi.com/reportEmbed?reportId=cd496c1c-8df0-48e7-8b92-e2932298743e&groupId=912f2b34-7daa-4589-83df-35c75944d864&w=2&config=eyJjbHVzdGVyVXJsIjoiaHR0cHM6Ly9XQUJJLVVTLUVBU1QyLUItUFJJTUFSWS1yZWRpcmVjdC5hbmFseXNpcy53aW5kb3dzLm5ldCIsImVtYmVkRmVhdHVyZXMiOnsibW9kZXJuRW1iZWQiOnRydWV9fQ%3d%3d
Token	eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsIng1dCI6Imh1Tjk1SXZQZmVocTM0R3pCRFoxR1hHaXJuTSIsImtpZCI6Imh1Tjk1SXZQZmVocTM0R3pCRFoxR1hHaXJuTSJ9.eyJhdWQiOiJodHRwczovL2FuYWx5c2lzLndpbmRvd3MubmV0L3Bvd2VyYmkvYXBpIiwiaXNzIjoiaHR0cHM6Ly9zdHMud2luZG93cy5uZXQvMmYyM2M1ZWEtNWE3NS00MWY2LTkyMmUtZDMzOTIzMTNlNjFkLyIsImlhdCI6MTU5NzYxMDkyNCwibmJmIjoxNTk3NjEwOTI0LCJleHAiOjE1OTc2MTQ4MjQsImFjY3QiOjAsImFjciI6IjEiLCJhaW8iOiJBVFFBeS84UUFBQUFFOTRxUFdscmR6MU0yRnVCSDZmY3lnSHlYUDM5UE9ISEhaWUNTYXIzQmVYaENNdkxJK3RWcnQ1Z0ZGcUd0TWZLIiwiYW1yIjpbInB3ZCJdLCJhcHBpZCI6ImQ0YWY4YjNiLWY4YWEtNGZlZC04Mzg0LWFlZWZjNjExMDMzYSIsImFwcGlkYWNyIjoiMSIsImZhbWlseV9uYW1lIjoiUGF0dGlzb24iLCJnaXZlbl9uYW1lIjoiVGVkIiwiaXBhZGRyIjoiNDcuMjAwLjExOS4zNyIsIm5hbWUiOiJUZWQgUGF0dGlzb24iLCJvaWQiOiI1NDk5NWExNC1mYTZlLTQwNzEtYjUwYy05MjYwMDE0ZjRmZjAiLCJwdWlkIjoiMTAwMzIwMDBEMjc2QkJBOSIsInNjcCI6IkNvbnRlbnQuQ3JlYXRlIERhdGFzZXQuUmVhZFdyaXRlLkFsbCBHcm91cC5SZWFkLkFsbCBSZXBvcnQuUmVhZFdyaXRlLkFsbCBXb3Jrc3BhY2UuUmVhZFdyaXRlLkFsbCIsInN1YiI6Ik1HX2lYa290U2FVQ1FtREFCRjdBM3c2NmJNM3NvMDEtVWJnMG14TkFXaEEiLCJ0aWQiOiIyZjIzYzVlYS01YTc1LTQxZjYtOTIyZS1kMzM5MjMxM2U2MWQiLCJ1bmlxdWVfbmFtZSI6IlRlZFBAcG93ZXJiaWRldmNhbXAubmV0IiwidXBuIjoiVGVkUEBwb3dlcmJpZGV2Y2FtcC5uZXQiLCJ1dGkiOiIyMXRUY1JxeGlVbUd4NkZzZUt0VkFBIiwidmVyIjoiMS4wIiwid2lkcyI6WyI2MmU5MDM5NC02OWY1LTQyMzctOTE5MC0wMTIxNzcxNDVlMTAiXX0.WkfWpI-Qbv3v73qCZh51KkEO9dPD2bk2XrqdOe2HZamcFO_nwPd75PwmLoqfwWTIY0lchUsFYS188RrDovVFDzLrKe7fuGT4ksDNJttKOvdt4R9Y4DNJHIfCznIYOAm6IyltscRA3gK1lA_l7DKPpaJnGdiwd0GwOLN50JIbJCJtZOiZpvGTjeDdvHWDCz7WaJxdDDiLDEwIKDzMvvEDHkGC5vx-NZtNl2aZLrVY4y8G-B6FqiqkIK-gmUWj4qvSuvNXPrc6VfTn1il0AIir6Wlf5uBcH5yplF3Hd9mDC9c75I0HNiPmt6Sv_N_rlkCf3DwKA3t3gGQEfVvsgnjuOg

    h) You're done testing. Close the browser, return to Visual Studio Code and stop the debug session using the debug toolbar.

You are now half way in your development efforts to embed a Power BI report. You have written the server-side code to call the Power BI Service API and retrieve the data required to embed a report. In the next exercise, you will complete the Power BI embedding implementation by adding client-side JavaScript code which programs against the Power BI JavaScript API.

Exercise 4: Embedding a Report using powerbi.js

In this exercise, you will modify the view named **Embed.cshtml** to embed a Power BI report on a web page. Your work will involve adding a new a JavaScript file named **embed.js** in which you will write the minimal client-side code required to embed a report.

1. Modify the razor view file named **Embed.cshtml**.
   a) Inside the **Views > Home** folder, locate and open **Embed.cshtml** in an editor window.
   b) Replace the contents of **Embed.cshtml** with the following code.

```
@model UserOwnsData.Services.EmbeddedReportViewModel;

<div id="embed-container" style="height:800px;"></div>

@section Scripts {

}
```

Note that the div element with the ID of **embed-container** will be used as the embed container.

Over the next few steps, you will add three **script** tags into the **Scrips** section. The benefit of adding script tags into the **Scripts** section is that they will load after the JavaScript libraries such as jquery which are loaded from the shared view **_Layout.cshtml**.

   c) Place your cursor inside the **Scripts** section and paste in the following **script** tag to import **powerbi.min.js** from a CDN.

```
<script src="https://cdn.jsdelivr.net/npm/powerbi-client@2.13.3/dist/powerbi.min.js"></script>
```

powerbi.min.js is the JavaScript file that loads the client-side library named the **Power BI JavaScript API**.

   d) Underneath the **script** tag for the Power BI JavaScript API, add a second **script** tag using the following code.

```
<script>
  var viewModel = {
    reportId: "@Model.Id",
    embedUrl: "@Model.EmbedUrl",
    token: "@Model.Token"
  };
</script>
```

This **script** tag is creates a JavaScript object named **viewModel** which is accessible to the JavaScript code you'll write later in this lab.

   e) Underneath the other two **script** tags, add a third **script** tag to load a custom JavaScript file named **embed.js**.

```
<script src="~/js/embed.js"></script>
```

Note that the JavaScript file named **embed.js** does not exist yet. You will create the **embed.js** file in the next step.

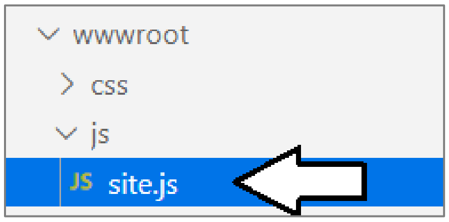
   f) When you are done, the contents you have in **Embed.cshtml** should match the following code listing.

```
@model UserOwnsData.Services.EmbeddedReportViewModel;

<div id="embed-container" style="height:800px;"></div>

@section Scripts {
  <script src="https://cdn.jsdelivr.net/npm/powerbi-client@2.13.3/dist/powerbi.min.js"></script>
  <script>
    var viewModel = {
      reportId: "@Model.Id",
      embedUrl: "@Model.EmbedUrl",
      token: "@Model.Token"
    };
  </script>
  <script src="~/js/embed.js"></script>
}
```

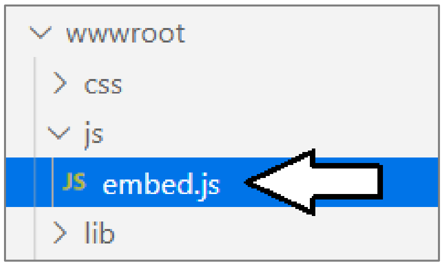
   g) Save your changes and close **Embed.cshtml**.

The final step is to add a new JavaScript file named **embed.js** with the code required to embed a report.

2. Add a new JavaScript file named **embed.js**.
   a) Locate the top-level folder named **wwwroot** and expand it.
   b) Locate the **js** folder inside the **wwwroot** folder and expand that.
   c) Currently, there should be one file inside the **wwwroot > js** folder named **site.js**.



   d) Rename **site.js** to **embed.js**.



3. Add the JavaScript code to **embed.js** to embed a report.
   a) Open **embed.js** in an editor window.
   b) Delete whatever content exists inside **embed.js**.
   c) Paste the following code into **embed.js** to provide a starting point.

```
$(function(){

  // 1 - get DOM object for div that is report container

  // 2 - get report embedding data from view model

  // 3 - embed report using the Power BI JavaScript API.

  // 4 - add logic to resize embed container on window resize event

});
```

You will now copy and paste four sections of JavaScript code into **embed.js** to complete the implementation. Note that you can copy and paste all the code at once by copying the contents of **Exercise 4 - embed.js.txt** in the **StudentLabFiles** folder.

   d) Add the following JavaScript code to create a variable named **reportContainer** which holds a reference to **embed-container**.

```
// 1 - get DOM object for div that is report container
var reportContainer = document.getElementById("embed-container");
```

   e) Add code to create 3 variables named **reportId**, **embedUrl** and **token** which are initialized from the global **viewModel** object.

```
// 2 - get report embedding data from view model
var reportId = window.viewModel.reportId;
var embedUrl = window.viewModel.embedUrl;
var token = window.viewModel.token
```

Now this JavaScript code has retrieved the three essential pieces of data from **window.viewModel** to embed a Power BI report.

f) Add the following code to embed a report by calling the **powerbi.embed** function provided by the Power BI JavaScript API.

```
// 3 - embed report using the Power BI JavaScript API.
var models = window['powerbi-client'].models;

var config = {
  type: 'report',
  id: reportId,
  embedUrl: embedUrl,
  accessToken: token,
  permissions: models.Permissions.All,
  tokenType: models.TokenType.Aad,
  viewMode: models.ViewMode.View,
  settings: {
    panes: {
      filters: { expanded: false, visible: true },
      pageNavigation: { visible: false }
    }
  }
};

// Embed the report and display it within the div container.
var report = powerbi.embed(reportContainer, config);
```

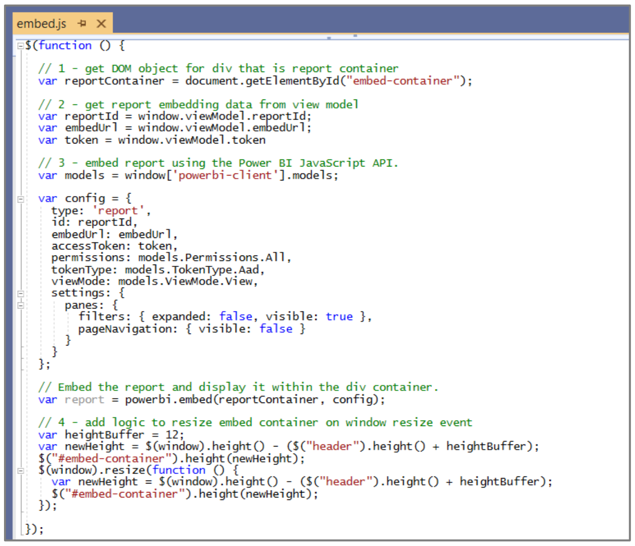
Note that the variable named **models** is initialized using a call to **window['powerbi-client'].models**. The **models** variable is used to set configuration values such as **models.Permissions.All**, **models.TokenType.Aad** and **models.ViewMode.View**.

A key point is that you need to create a configuration object in order to call the **powerbi.embed** function. You can learn a great deal more about creating the configuration object for Power BI embedding in this wiki for the Power BI JavaScript API.

g) Add the following JavaScript code to resize the **embed-container** div element whenever the window resize event fires.

```
// 4 - add logic to resize embed container on window resize event
var heightBuffer = 12;
var newHeight = $(window).height() - ($("header").height() + heightBuffer);
$("#embed-container").height(newHeight);
$(window).resize(function () {
  var newHeight = $(window).height() - ($("header").height() + heightBuffer);
  $("#embed-container").height(newHeight);
});
```

h) Your code in **embed.js** should match the following screenshot.

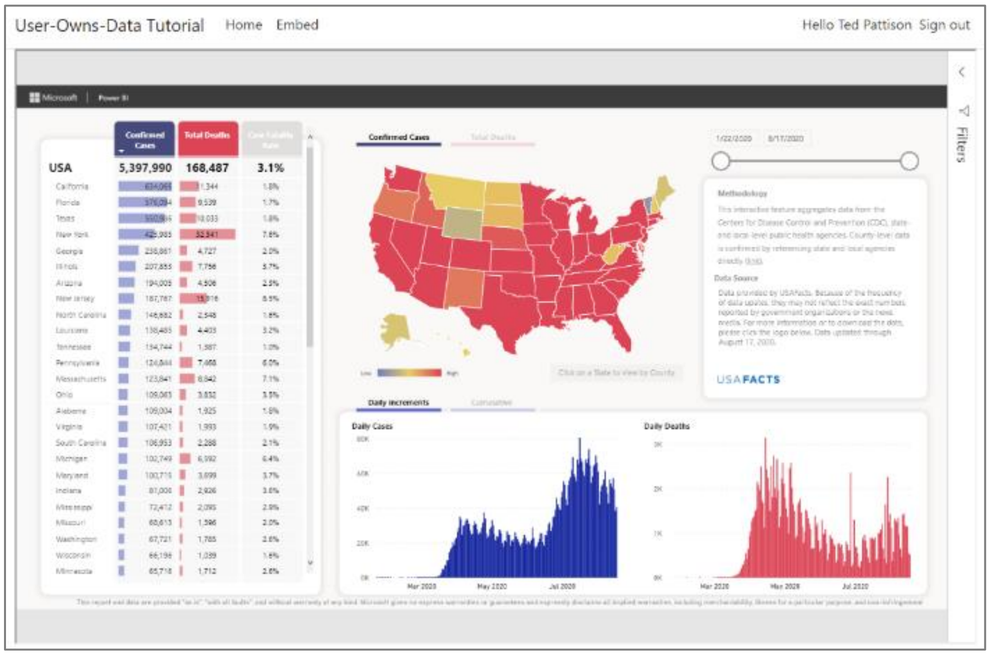


```
$(function () {

  // 1 - get DOM object for div that is report container
  var reportContainer = document.getElementById("embed-container");

  // 2 - get report embedding data from view model
  var reportId = window.viewModel.reportId;
  var embedUrl = window.viewModel.embedUrl;
  var token = window.viewModel.token

  // 3 - embed report using the Power BI JavaScript API.
  var models = window['powerbi-client'].models;

  var config = {
    type: 'report',
    id: reportId,
    embedUrl: embedUrl,
    accessToken: token,
    permissions: models.Permissions.All,
    tokenType: models.TokenType.Aad,
    viewMode: models.ViewMode.View,
    settings: {
      panes: {
        filters: { expanded: false, visible: true },
        pageNavigation: { visible: false }
      }
    }
  };

  // Embed the report and display it within the div container.
  var report = powerbi.embed(reportContainer, config);

  // 4 - add logic to resize embed container on window resize event
  var heightBuffer = 12;
  var newHeight = $(window).height() - ($("header").height() + heightBuffer);
  $("#embed-container").height(newHeight);
  $(window).resize(function () {
    var newHeight = $(window).height() - ($("header").height() + heightBuffer);
    $("#embed-container").height(newHeight);
  });

});
```

Remember you can copy and paste all the code at once by using the text in **Exercise 4 - embed.js.txt** in the **StudentLabFiles** folder.

i) Save your changes and close **embed.js**.

4. Run the web application in the Visual Studio Code debugger to test your work on the **Embed** page.
   a) Start the Visual Studio Code debugger by selecting **Run > Start Debugging** or by pressing the **{F5}** keyboard shortcut.
   b) The **UserOwnsData** web application should display the home page as shown to an anonymous user.
   c) Click on the **Embed** link in the top nav menu to navigate to the **Embed** page and login when prompted.
   d) You should now be able to navigate to the **Embed** page and see the Power BI report displayed on the page.



	Confirmed Cases	Total Deaths	Case Fatality Rate
USA	5,397,990	168,487	3.1%
California	634,068	11,344	1.8%
Florida	576,094	9,539	1.7%
Texas	[illegible]	10,033	1.8%
New York	[illegible]	32,941	7.8%
Georgia	238,861	4,727	2.0%
Illinois	207,855	7,756	3.7%
Arizona	194,005	4,506	2.3%
New Jersey	187,767	[illegible]	8.5%
North Carolina	148,882	2,548	1.8%
Louisiana	138,485	4,403	3.2%
Tennessee	134,744	1,387	1.0%
Pennsylvania	124,844	7,468	6.0%
Massachusetts	123,841	8,842	7.1%
Ohio	109,063	3,832	3.5%
Alabama	109,004	1,925	1.8%
Virginia	107,421	1,993	1.9%
South Carolina	106,953	2,288	2.1%
Michigan	102,749	6,592	6.4%
Maryland	100,715	3,699	3.7%
Indiana	81,006	2,926	3.6%
Mississippi	72,412	2,095	2.9%
Missouri	68,613	1,396	2.0%
Washington	67,721	1,785	2.6%
Wisconsin	66,198	1,039	1.6%
Minnesota	65,718	1,712	2.6%

Methodology

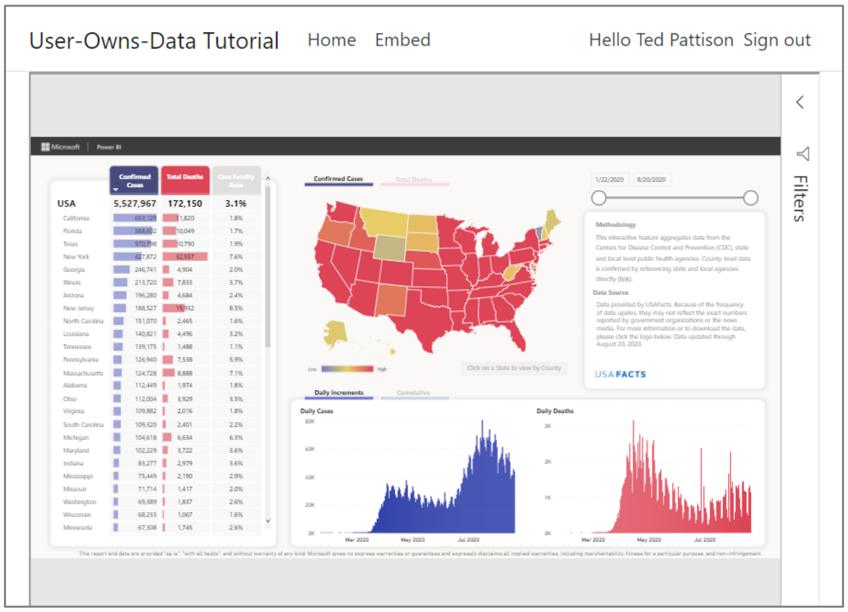
This interactive feature aggregates data from the Centers for Disease Control and Prevention (CDC), state- and local-level public health agencies. County-level data is confirmed by referencing state and local agencies directly (link).

Data Source

Data provided by USAFacts. Because of the frequency of data upates, they may not reflect the exact numbers reported by government organizations or the news media. For more information or to download the data, please click the logo below. Data updated through August 17, 2020.

This report and data are provided "as is", "with all faults", and without warranty of any kind. Microsoft gives no express warranties or guarantees and expressly disclaims all implied warranties, including merchantability, fitness for a particular purpose, and non-infringement.

   e) Try resizing the browser window. The embedded report should continually adapt to the size of the window.



	Confirmed Cases	Total Deaths	Case Fatality Rate
USA	5,527,967	172,150	3.1%
California	653,129	11,820	1.8%
Florida	588,602	10,049	1.7%
Texas	570,798	10,790	1.9%
New York	427,872	32,557	7.6%
Georgia	246,741	4,904	2.0%
Illinois	213,720	7,833	3.7%
Arizona	196,280	4,684	2.4%
New Jersey	188,527	15,932	8.5%
North Carolina	151,070	2,465	1.6%
Louisiana	140,821	4,496	3.2%
Tennessee	139,175	1,488	1.1%
Pennsylvania	126,940	7,538	5.9%
Massachusetts	124,728	8,888	7.1%
Alabama	112,449	1,974	1.8%
Ohio	112,004	3,929	3.5%
Virginia	109,882	2,016	1.8%
South Carolina	109,320	2,401	2.2%
Michigan	104,618	6,634	6.3%
Maryland	102,229	3,722	3.6%
Indiana	83,277	2,979	3.6%
Mississippi	75,449	2,190	2.9%
Missouri	71,714	1,417	2.0%
Washington	69,389	1,837	2.6%
Wisconsin	68,233	1,067	1.6%
Minnesota	67,308	1,745	2.6%

Methodology

This interactive feature aggregates data from the Centers for Disease Control and Prevention (CDC), state and local level public health agencies. County level data is confirmed by referencing state and local agencies directly (link).

Data Source

Data provided by USAFacts. Because of the frequency of data upates, they may not reflect the exact numbers reported by government organizations or the news media. For more information or to download the data, please click the logo below. Data updated through August 20, 2020.

This report and data are provided "as is", "with all faults", and without warranty of any kind. Microsoft gives no express warranties or guarantees and expressly disclaims all implied warranties, including merchantability, fitness for a particular purpose, and non-infringement.

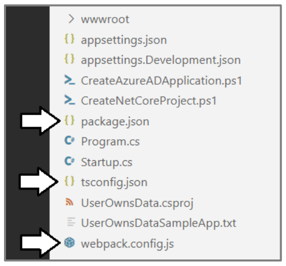
   f) You're done testing. Close the browser, return to Visual Studio Code and stop the debug session using the debug toolbar.

You have now reached an important milestone. You can now tell all your peers that you have embedded a Power BI report. However, there is more that you can do to improve the developer experience for writing client-side code against the Power BI JavaScript API. In the next exercise, you will add support to your project so that you can program client-side code using TypeScript instead of JavaScript. By moving to TypeScript you can benefit from strongly-typed programming, compile-time type checking and much better IntelliSense.

Exercise 5: Adding TypeScript Support to a .NET 5 Project

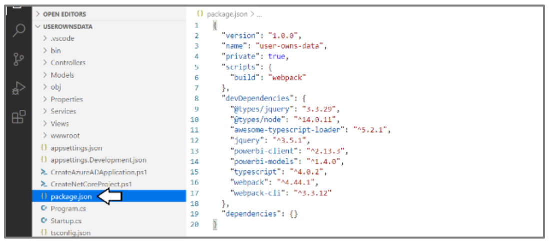
In this exercise, you will add support for developing your client-side code with TypeScript instead of JavaScript. It is assumed that you have already installed Node.js so that the Node Package Manager command-line tool (**npm)** is available at the commend line. You will begin by adding several Node.js configuration files to the root folder of the **UserOwnsData** project. After that you will restore a set of Node.js packages and use the webpack utility to compile TypeScript code into an output file named **embed.js**.

1. Copy three essential node.js development configuration files into the root folder of the **UserOwnsData** project.
   a) Locate these three files in the **StudentLabFiles** folder.
      i) **package.json** – the standard project file for all Node.js projects.
      ii) **tsconfig.json** – a configuration file used by the TypeScript compiler (TSC).
      iii) **webpack.config.js** – a configuration file used by the webpack utility.
   b) Copy **package.json**, **tsconfig.json** and **webpack.config.js** into the root folder of the **UserOwnsData** project.



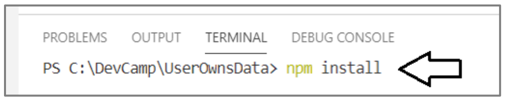
Visual Studio Code makes it difficult to add existing files to a project folder. You can use the Windows Explorer to copy these three files from the **StudentLabFiles** folder to the **UserOwnsData** project folder.

2. Restore the Node.js packages which are referenced in **package.json**.
   a) Open **package.json** and review the Node.js packages referenced in **devDependencies** section.



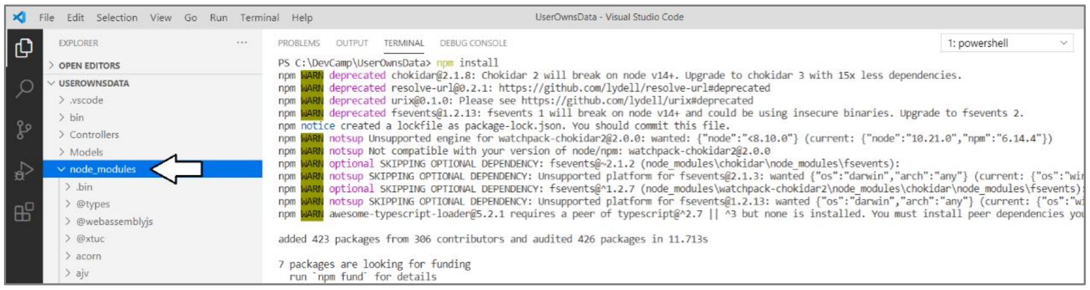
```
{
  "version": "1.0.0",
  "name": "user-owns-data",
  "private": true,
  "scripts": {
    "build": "webpack"
  },
  "devDependencies": {
    "@types/jquery": "3.3.29",
    "@types/node": "^14.0.11",
    "awesome-typescript-loader": "^5.2.1",
    "jquery": "^3.5.1",
    "powerbi-client": "^2.13.3",
    "powerbi-models": "^1.4.0",
    "typescript": "^4.0.2",
    "webpack": "^4.44.1",
    "webpack-cli": "^3.3.12"
  },
  "dependencies": {}
}
```

   b) Open the Visual Studio Code terminal by clicking the **View > Terminal** menu command or by using **Ctrl+`** keyboard shortcut.
   c) Run the **npm install** command to restore the list of Node.js packages.



```
PS C:\DevCamp\UserOwnsData> npm install
```

   d) When you run the **npm install** command, **npm** will download all the Node.js packages into the **node_modules** folder.



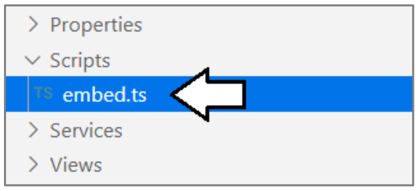
```
PS C:\DevCamp\UserOwnsData> npm install
npm WARN deprecated chokidar@2.1.8: Chokidar 2 will break on node v14+. Upgrade to chokidar 3 with 15x less dependencies.
npm WARN deprecated resolve-url@0.2.1: https://github.com/lydell/resolve-url#deprecated
npm WARN deprecated urix@0.1.0: Please see https://github.com/lydell/urix#deprecated
npm WARN deprecated fsevents@1.2.13: fsevents 1 will break on node v14+ and could be using insecure binaries. Upgrade to fsevents 2.
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN notsup Unsupported engine for watchpack-chokidar2@2.0.0: wanted: {"node":"<8.10.0"} (current: {"node":"10.21.0","npm":"6.14.4"})
npm WARN notsup Not compatible with your version of node/npm: watchpack-chokidar2@2.0.0
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@~2.1.2 (node_modules\chokidar\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@2.1.3: wanted {"os":"darwin","arch":"any"} (current: {"os":"wi
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@^1.2.7 (node_modules\watchpack-chokidar2\node_modules\chokidar\node_modules\fsevents):
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@1.2.13: wanted {"os":"darwin","arch":"any"} (current: {"os":"w
npm WARN awesome-typescript-loader@5.2.1 requires a peer of typescript@^2.7 || ^3 but none is installed. You must install peer dependencies yo

added 423 packages from 306 contributors and audited 426 packages in 11.713s

7 packages are looking for funding
  run `npm fund` for details
```

3. Take a quick look at the **tsconfig.json** file.
   a) Open the **tsconfig.json** file in an editor window and examine the TypeScript compiler settings inside.
   b) When you are done, close **tsconfig.json** without saving any changes.
4. Take a quick look at the **webpack.config.js** file.
   a) Open the **webpack.config.js** file in an editor window and examine its content.

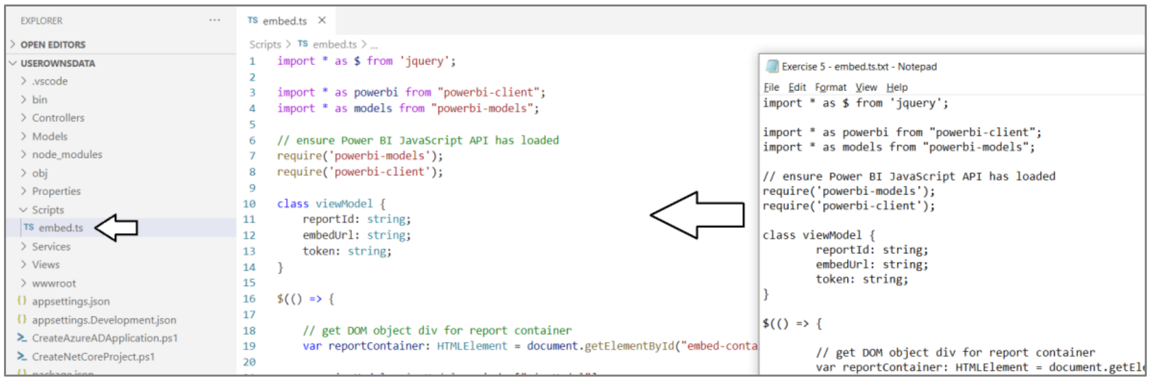
```
const path = require('path');

module.exports = {
  entry: './Scripts/embed.ts',
  output: {
    filename: 'embed.js',
    path: path.resolve(__dirname, 'wwwroot/js'),
  },
  resolve: {
    extensions: ['.js', '.ts']
  },
  module: {
    rules: [
      { test: /\.(ts)$/, loader: 'awesome-typescript-loader' }
    ],
  },
  mode: "development",
  devtool: 'source-map'
};
```

Note the **entry** property of **model.exports** object is set to **./Scripts/embed.ts**. The **path** and **filename** of the **output** object combine to a file path of **wwwroot/js/embed.js**. When the webpack utility runs, it will look for a file named **embed.ts** in the **Scripts** folder as its main entry point for the TypeScript compiler (tsc.exe) and produce an output file in named **embed.js** in the **wwwroot/js** folder.

   b) When you are done, close **webpack.config.js** without saving any changes.
5. Add a new TypeScript source file named **embed.ts**.
   a) In the **UserOwnsData** project folder, create a new top-level folder named **Scripts**.
   b) Create a new file inside the **Scripts** folder named **embed.ts**.



   c) In Windows Explorer, locate the **Exercise 5 - embed.ts.txt** file in the **StudentLabFiles** folder.
   d) Open **Exercise 5 - embed.ts.txt** in a text editor such as Notepad and copy all its contents to the Windows clipboard.
   e) Return to Visual Studio Code and paste the contents of the Windows clipboard into **Embed.ts.**

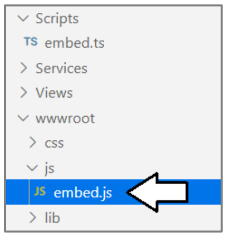


```
import * as $ from 'jquery';

import * as powerbi from "powerbi-client";
import * as models from "powerbi-models";

// ensure Power BI JavaScript API has loaded
require('powerbi-models');
require('powerbi-client');

class viewModel {
    reportId: string;
    embedUrl: string;
    token: string;
}

$(() => {

    // get DOM object div for report container
    var reportContainer: HTMLElement = document.getElementById("embed-conta
```

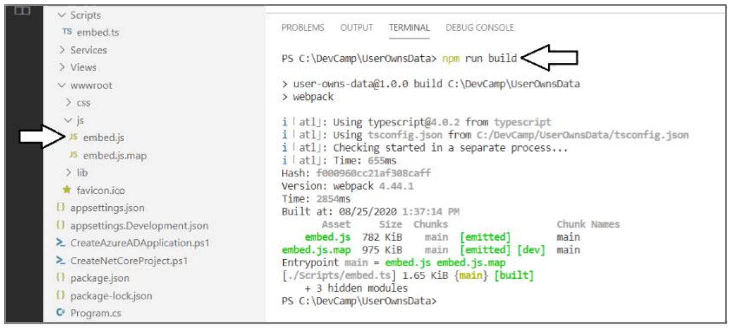
```
import * as $ from 'jquery';

import * as powerbi from "powerbi-client";
import * as models from "powerbi-models";

// ensure Power BI JavaScript API has loaded
require('powerbi-models');
require('powerbi-client');

class viewModel {
        reportId: string;
        embedUrl: string;
        token: string;
}

$(() => {

        // get DOM object div for report container
        var reportContainer: HTMLElement = document.getEl
```

   f) Save your changes and close **embed.ts**.

6. Use the webpack utility to compile **embed.ts** into **embed.js**.
   a) Locate the original **embed.js** file in the **wwwroot/js** folder and delete it.

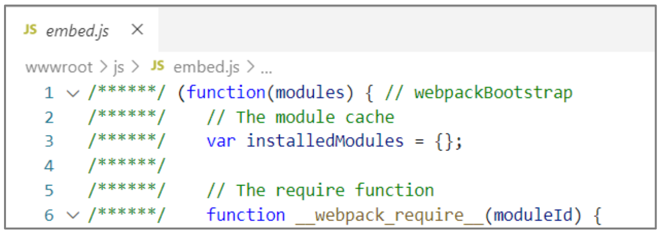


   b) Open the Visual Studio Code terminal by clicking the **View > Terminal** menu command or by using **Ctrl+`** keyboard shortcut.
   c) Run the **npm run build** command to run the webpack utility.
   d) When you run **npm run build**, webpack should automatically generate a new version of **embed.js** in the **wwwroot/js** folder.



```
PS C:\DevCamp\UserOwnsData> npm run build

> user-owns-data@1.0.0 build C:\DevCamp\UserOwnsData
> webpack

i ｢atl｣: Using typescript@4.0.2 from typescript
i ｢atl｣: Using tsconfig.json from C:/DevCamp/UserOwnsData/tsconfig.json
i ｢atl｣: Checking started in a separate process...
i ｢atl｣: Time: 655ms
Hash: f000960cc21af308caff
Version: webpack 4.44.1
Time: 2854ms
Built at: 08/25/2020 1:37:14 PM
       Asset      Size  Chunks                   Chunk Names
    embed.js   782 KiB    main  [emitted]        main
embed.js.map   975 KiB    main  [emitted] [dev]  main
Entrypoint main = embed.js embed.js.map
[./Scripts/embed.ts] 1.65 KiB {main} [built]
    + 3 hidden modules
PS C:\DevCamp\UserOwnsData>
```

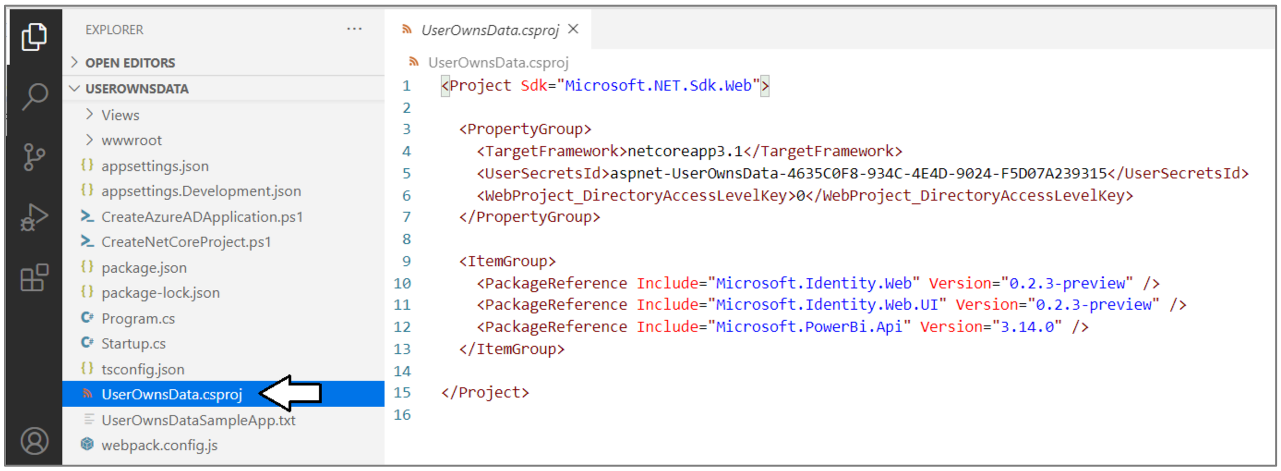
   e) Open the new version of **embed.js**. You should see it is a source file generated by the webpack utility.



```
/******/ (function(modules) { // webpackBootstrap
/******/     // The module cache
/******/     var installedModules = {};
/******/
/******/     // The require function
/******/     function __webpack_require__(moduleId) {
```

   f) Close **embed.js** without saving any changes.

7. Update **UserOwnsData.csproj** to add the **npm run build** command as part of the MSBuild build process.
   a) Open the project file named **UserOwnsData.csproj** in an editor window.



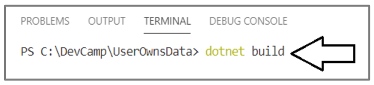
```
<Project Sdk="Microsoft.NET.Sdk.Web">

  <PropertyGroup>
    <TargetFramework>netcoreapp3.1</TargetFramework>
    <UserSecretsId>aspnet-UserOwnsData-4635C0F8-934C-4E4D-9024-F5D07A239315</UserSecretsId>
    <WebProject_DirectoryAccessLevelKey>0</WebProject_DirectoryAccessLevelKey>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.Identity.Web" Version="0.2.3-preview" />
    <PackageReference Include="Microsoft.Identity.Web.UI" Version="0.2.3-preview" />
    <PackageReference Include="Microsoft.PowerBi.Api" Version="3.14.0" />
  </ItemGroup>

</Project>
```

b) Add a new **Target** element named **PostBuild** to run the **npm run build** command as shown in the following code listing.

```
<Project Sdk="Microsoft.NET.Sdk.Web">

  <PropertyGroup>
    <TargetFramework>netcoreapp3.1</TargetFramework>
    <UserSecretsId>aspnet-UserOwnsData-87660A42-54AC-4CF9-8583-B31608FED004</UserSecretsId>
    <WebProject_DirectoryAccessLevelKey>0</WebProject_DirectoryAccessLevelKey>
  </PropertyGroup>

  <ItemGroup>
    <PackageReference Include="Microsoft.Identity.Web" Version="0.3.0-preview" />
    <PackageReference Include="Microsoft.Identity.Web.UI" Version="0.3.0-preview" />
    <PackageReference Include="Microsoft.PowerBi.Api" Version="3.14.0" />
  </ItemGroup>

  <Target Name="PostBuild" AfterTargets="PostBuildEvent">
    <Exec Command="npm run build" />
  </Target>

</Project>
```

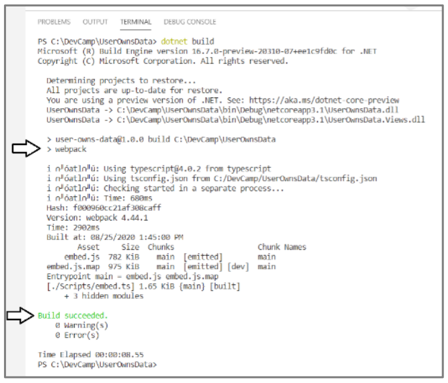
c) Save your changes and close **UserOwnsData.csproj**.

d) Return to the terminal and run the **dotnet build** command.



```
PS C:\DevCamp\UserOwnsData> dotnet build
```

e) When you run the **dotnet build** command, the output window should show you that the webpack command is running.



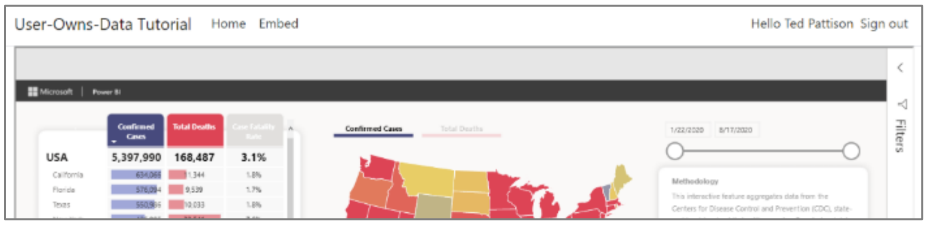
```
PS C:\DevCamp\UserOwnsData> dotnet build
Microsoft (R) Build Engine version 16.7.0-preview-20310-07+ee1c9fd0c for .NET
Copyright (C) Microsoft Corporation. All rights reserved.

  Determining projects to restore...
  All projects are up-to-date for restore.
  You are using a preview version of .NET. See: https://aka.ms/dotnet-core-preview
  UserOwnsData -> C:\DevCamp\UserOwnsData\bin\Debug\netcoreapp3.1\UserOwnsData.dll
  UserOwnsData -> C:\DevCamp\UserOwnsData\bin\Debug\netcoreapp3.1\UserOwnsData.Views.dll

  > user-owns-data@1.0.0 build C:\DevCamp\UserOwnsData
  > webpack

  i n╜óatln╜ü: Using typescript@4.0.2 from typescript
  i n╜óatln╜ü: Using tsconfig.json from C:/DevCamp/UserOwnsData/tsconfig.json
  i n╜óatln╜ü: Checking started in a separate process...
  i n╜óatln╜ü: Time: 680ms
  Hash: f000960cc21af308caff
  Version: webpack 4.44.1
  Time: 2902ms
  Built at: 08/25/2020 1:45:00 PM
         Asset     Size  Chunks                    Chunk Names
      embed.js  782 KiB    main  [emitted]        main
  embed.js.map  975 KiB    main  [emitted] [dev]  main
  Entrypoint main = embed.js embed.js.map
  [./Scripts/embed.ts] 1.65 KiB {main} [built]
      + 3 hidden modules

Build succeeded.
    0 Warning(s)
    0 Error(s)

Time Elapsed 00:00:08.55
PS C:\DevCamp\UserOwnsData>
```

Now whenever you start a debug session with the **{F5}** key, the TypeScript in **embed.ts** will be automatically compiled into **embed.js**.

8. Run the web application in the Visual Studio Code debugger to test your work on the **Embed** page.

a) Start the Visual Studio Code debugger by selecting **Run > Start Debugging** or by pressing the **{F5}** keyboard shortcut.

b) Click on the **Embed** link in the top nav menu to navigate to the **Embed** page and login when prompted to view the report.



c) You're done testing. Close the browser, return to Visual Studio Code and stop the debug session using the debug toolbar.

When you test the **UserOwnsData** web application, it should behave just as it did when you tested it in Exercise 4. The difference is that now the client-side behavior is now implemented with TypeScript instead of JavaScript.

Exercise 6: Creating a View Model for App Workspaces

Up to this point, you have implemented the **UserOwnsData** project to embed a single report by hard-coding the IDs of that report and its hosting workspace. In this exercise, you will remove the hard-coded IDs and extend the **Embed** page of the **UserOwnsData** project to dynamically discover what workspaces and reports are available to the current user.

1. Extend the **PowerBiServiceApi** class with a new method named **GetEmbeddedViewModel**.
   a) Locate the **PowerBiServiceApi.cs** in the **Service** folder and open it in an editor window.
   b) Add the following method named **GetEmbeddedViewModel** to the end of **PowerBiServiceApi** class.

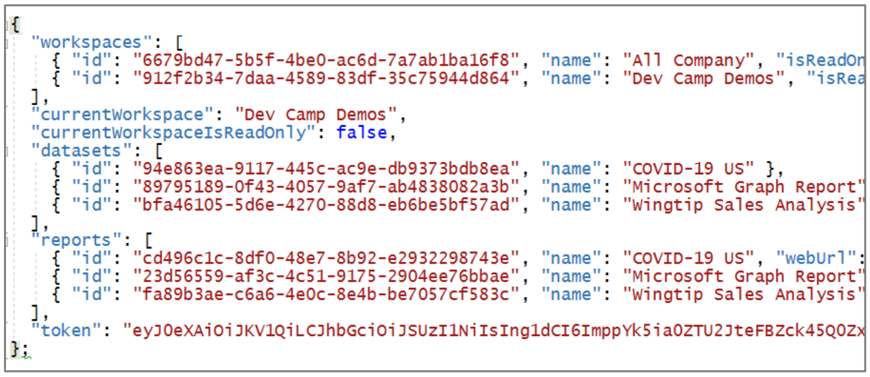
```
public async Task<string> GetEmbeddedViewModel(string appWorkspaceId = "") {

  var accessToken = this.tokenAcquisition.GetAccessTokenForUserAsync(RequiredScopes).Result;
  var tokenCredentials = new TokenCredentials (accessToken, "Bearer");
  PowerBIClient pbiClient = new PowerBIClient (new Uri (urlPowerBiServiceApiRoot), tokenCredentials);

  Object viewModel;
  if (string.IsNullOrEmpty (appWorkspaceId)) {
    viewModel = new {
      currentWorkspace = "My Workspace",
      workspaces = ( await pbiClient.Groups.GetGroupsAsync() ).Value,
      datasets = ( await pbiClient.Datasets.GetDatasetsAsync() ).Value,
      reports = ( await pbiClient.Reports.GetReportsAsync() ).Value,
      token = accessToken
    };
  } else {
    Guid workspaceId = new Guid (appWorkspaceId);
    var workspaces = (await pbiClient.Groups.GetGroupsAsync ()).Value;
    var currentWorkspace = workspaces.First ((workspace) => workspace.Id == workspaceId);
    viewModel = new {
      workspaces = workspaces,
        currentWorkspace = currentWorkspace.Name,
        currentWorkspaceIsReadOnly = currentWorkspace.IsReadOnly,
        datasets = (await pbiClient.Datasets.GetDatasetsInGroupAsync (workspaceId)).Value,
        reports = (await pbiClient.Reports.GetReportsInGroupAsync (workspaceId)).Value,
        token = accessToken
    };
  }

  return JsonConvert.SerializeObject(viewModel);
}
```

The **GetEmbeddedViewModel** method accepts an **appWorkspaceId** parameter and returns a string value with JSON-formatted data. If the **appWorkspaceId** parameter is blank, the **GetEmbeddedViewModel** method returns a view model for the current user's personal workspace. If the **appWorkspaceId** parameter contains a GUID, the **GetEmbeddedViewModel** method returns a view model for the app workspace associated with that GUID.

You can copy and paste this method from the **Exercise 6 - PowerBiServiceApi.cs.txt** file in the **StudentLabFiles** folder.

   a) Save your work and close **PowerBiServiceApi.cs**.

2. Enhance your conceptual understanding of the data involved by examining the JSON returned by **GetEmbeddedViewModel**.



```
{
  "workspaces": [
    { "id": "6679bd47-5b5f-4be0-ac6d-7a7ab1ba16f8", "name": "All Company", "isReadOn
    { "id": "912f2b34-7daa-4589-83df-35c75944d864", "name": "Dev Camp Demos", "isRea
  ],
  "currentWorkspace": "Dev Camp Demos",
  "currentWorkspaceIsReadOnly": false,
  "datasets": [
    { "id": "94e863ea-9117-445c-ac9e-db9373bdb8ea", "name": "COVID-19 US" },
    { "id": "89795189-0f43-4057-9af7-ab4838082a3b", "name": "Microsoft Graph Report"
    { "id": "bfa46105-5d6e-4270-88d8-eb6be5bf57ad", "name": "Wingtip Sales Analysis"
  ],
  "reports": [
    { "id": "cd496c1c-8df0-48e7-8b92-e2932298743e", "name": "COVID-19 US", "webUrl":
    { "id": "23d56559-af3c-4c51-9175-2904ee76bbae", "name": "Microsoft Graph Report"
    { "id": "fa89b3ae-c6a6-4e0c-8e4b-be7057cf583c", "name": "Wingtip Sales Analysis"
  ],
  "token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsIng1dCI6ImppYk5ia0ZTU2JteFBZck45Q0Zx
};
```

3. Modify **Embed** method in **HomeController** to call the **GetEmbeddedViewModel** method.
   a) Locate the **HomeController.cs** file and open it in an editor window.
   b) Locate the **Embed** method which should currently match this **Embed** method implementation.

```
public async Task<IActionResult> Embed() {
  Guid workspaceId = new Guid("912f2b34-7daa-4589-83df-35c75944d864");
  Guid reportId = new Guid("cd496c1c-8df0-48e7-8b92-e2932298743e");
  var viewModel = await powerBiServiceApi.GetReport(workspaceId, reportId);
  return View(viewModel);
}
```

   c) Delete the **Embed** method implementation and replace it the following code.

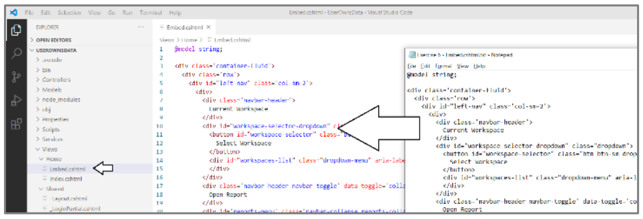
```
public async Task<IActionResult> Embed(string workspaceId) {
  var viewModel = await powerBiServiceApi.GetEmbeddedViewModel(workspaceId);
  // cast string value to object type in order to pass string value as MVC view model
  return View(viewModel as object);
}
```

   d) Save your work and close **HomeController.cs**.

There are a few things to note about the new implementation of the **Embed** controller action method. First, the method now takes a string parameter named **workspaceId**. When this controller method is passed a workspace ID in the **workspaceId** query string parameter, it passes that workspace ID along to the **PowerBiServiceApi** class when it calls the **GetEmbeddedViewModel** method.

The second thing to note about this example if that the string-based **viewModel** variable is cast to a type of **object** in the **return** statement using the syntax **View(viewModel as object)**. This is a required workaround because passing a string parameter to **View()** would fail because the string value would be treated as a view name instead of a view model being passed to the underlying view.

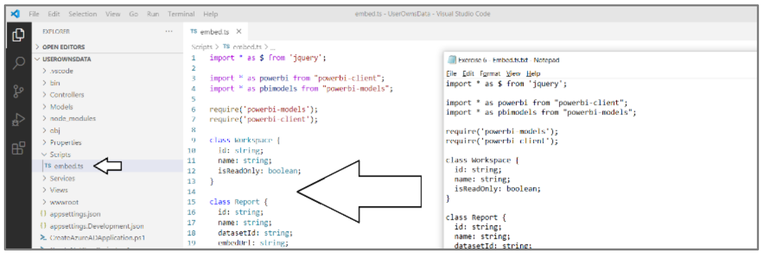
4. Replace the code in **Embed.cshtml** with a better implementation.
   a) Locate **Embed.cshtml** file in the **Views > Home** folder, open it in an editor window and delete all the content inside.
   b) In Windows Explorer, locate the **Exercise 6 - Embed.cshtml.txt** file in the **StudentLabFiles** folder.
   c) Open **Exercise 6 - Embed.cshtml.txt** in a text editor such as Notepad and copy all its contents to the Windows clipboard.
   d) Return to Visual Studio Code and paste the contents of to the Windows clipboard into **Embed.cshtml.**



```
@model string;

<div class='container-fluid'>
  <div class='row'>
    <div id="left-nav" class='col-sm-2'>
      <div>
        <div class='navbar-header'>
          Current Workspace
        </div>
        <div id="workspace-selector-dropdown" class="dropdown">
          <button id="workspace-selector" class="btn btn-sm drop
            Select Workspace
          </button>
          <div id="workspaces-list" class="dropdown-menu" aria-l
          </div>
        </div>
        <div class='navbar-header navbar-toggle' data-toggle='co
          Open Report
```

   e) Save your changes and close **Embed.cshtml.**

5. Replace the code in **Embed.ts** with a better implementation.
   a) Locate **Embed.ts** file in the **Scripts** folder, open it in an editor window and delete all the content inside.
   b) In Windows Explorer, locate the **Exercise 6 - Embed.ts.txt** file in the **StudentLabFiles** folder.
   c) Open **Exercise 6 - Embed.ts.txt** in a text editor such as Notepad and copy all its contents to the Windows clipboard.
   d) Return to Visual Studio Code and paste the content of **Exercise 6 - Embed.ts.txt** into **Embed.ts.**

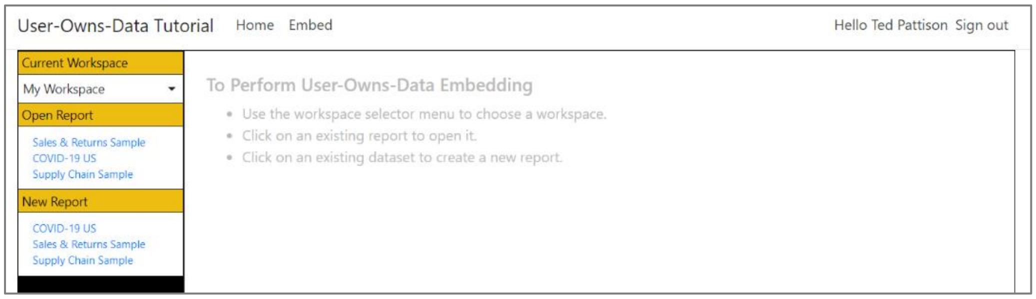

File Edit Selection View Go Run Terminal Help
embed.ts - UserOwnsData - Visual Studio Code
EXPLORER
OPEN EDITORS
USEROWNSDATA
.vscode
bin
Controllers
Models
node_modules
obj
Properties
Scripts
embed.ts
Services
Views
wwwroot
appsettings.json
appsettings.Development.json
CreateAzureADApplication.ps1
Exercise 6 - Embed.ts.txt - Notepad
File Edit Format View Help

```
import * as $ from 'jquery';

import * as powerbi from "powerbi-client";
import * as pbimodels from "powerbi-models";

require('powerbi-models');
require('powerbi-client');

class Workspace {
  id: string;
  name: string;
  isReadOnly: boolean;
}

class Report {
  id: string;
  name: string;
  datasetId: string;
  embedUrl: string;
```

   e) Save your changes and close **Embed.ts.**

Revieing the client-side code in **Embed.ts** has been left as an exercise for the reader, After you run and test the **UserOwnsData** application at the end of this exercise and you have experienced the user interface from the user perspective, you might want to examine the code in **Embed.ts** to see how this application implements the functionality to navigate between workspace and to discover the reports and datasets in each workspace.

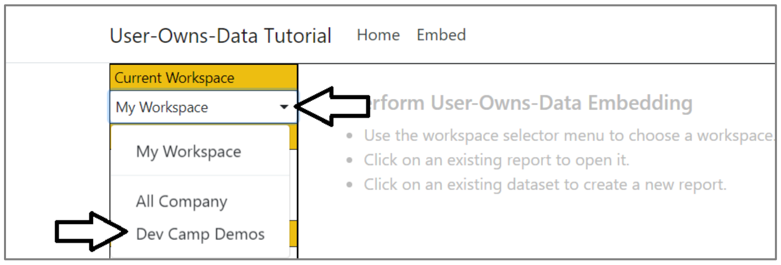
6. Run the web application in the .NET debugger to test your work on the **Embed** page.
   a) Start the Visual Studio Code debugger by selecting **Run > Start Debugging** or by pressing the **{F5}** keyboard shortcut.
   b) Click on the **Embed** link in the top nav menu to navigate to the **Embed** page and login when prompted.
   c) The **Embed** page should appear much differently than before as shown in the following screenshot.



To Perform User-Owns-Data Embedding
- Use the workspace selector menu to choose a workspace.
- Click on an existing report to open it.
- Click on an existing dataset to create a new report.

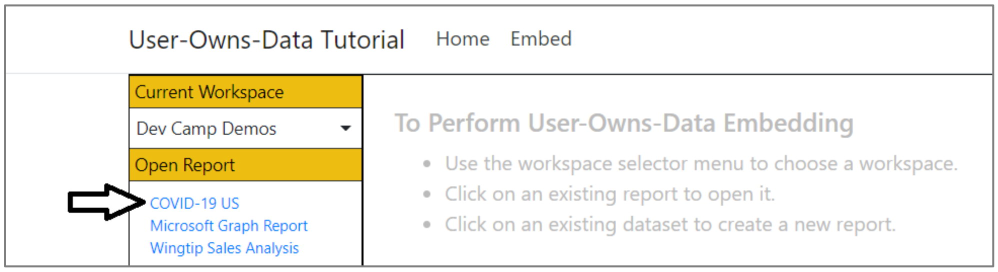
Note there is a dropdown list for the **Current Workspace** that you can use to navigate across workspaces.

   d) Navigate to the workspace you created earlier in this lab.



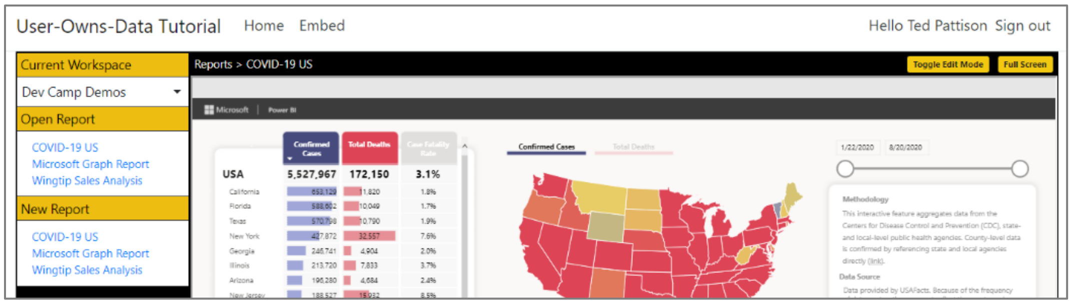
rform User-Owns-Data Embedding
- Use the workspace selector menu to choose a workspace.
- Click on an existing report to open it.
- Click on an existing dataset to create a new report.

   e) Click on a report in the **Open Report** section.

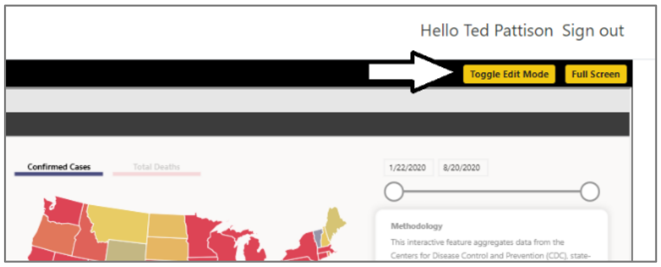


To Perform User-Owns-Data Embedding
- Use the workspace selector menu to choose a workspace.
- Click on an existing report to open it.
- Click on an existing dataset to create a new report.

   f) The report should open in read-only mode.

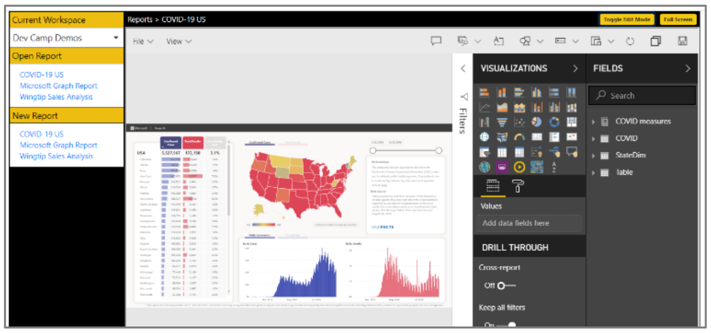


g) Click the **Toggle Edit Mode** button to move the report into edit mode.

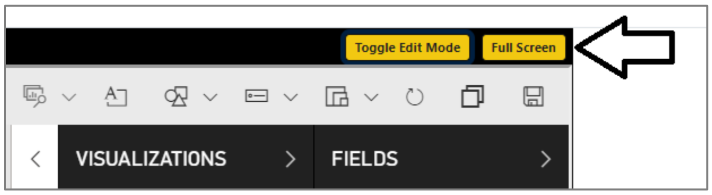


This interactive feature aggregates data from the Centers for Disease Control and Prevention (CDC), state-

h) Note that when the report goes into edit mode, there isn't much space to work on the report while editing.



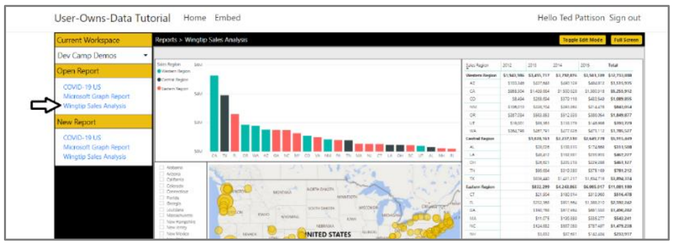
i) Click the Full Screen button to enter full screen mode



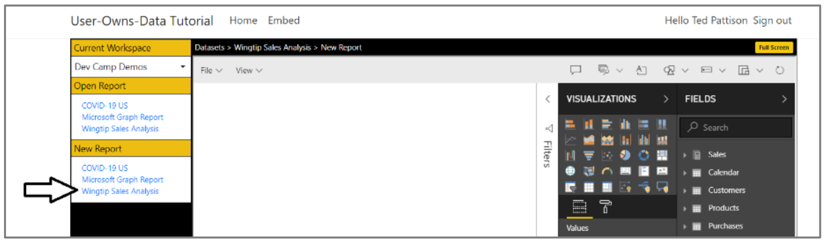
You can invoke the **File > Save** command in a report that is in edit mode to save your changes.

j) Press the **Esc** key in the keyboard to exit full screen mode.

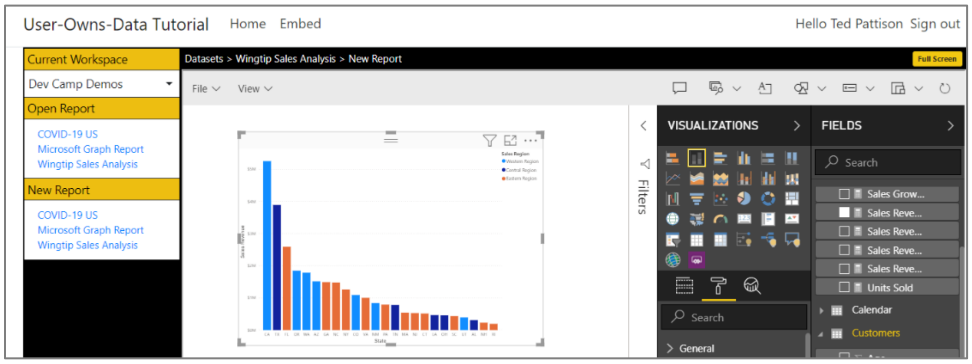
k) Click on a second report in the **Open Report** section to navigate between reports.



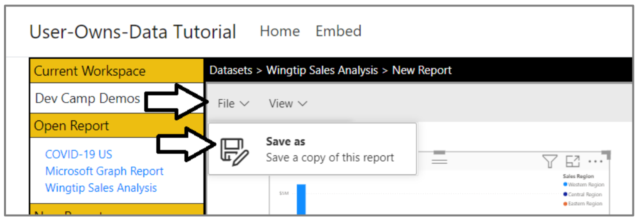
l) Create a new report by clicking on a dataset name in the **New Report** section.



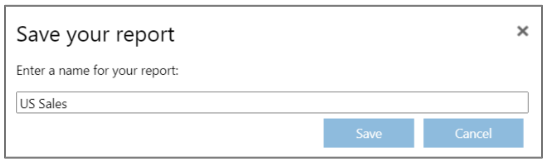
m) Add a simple visual of any type to the new report.



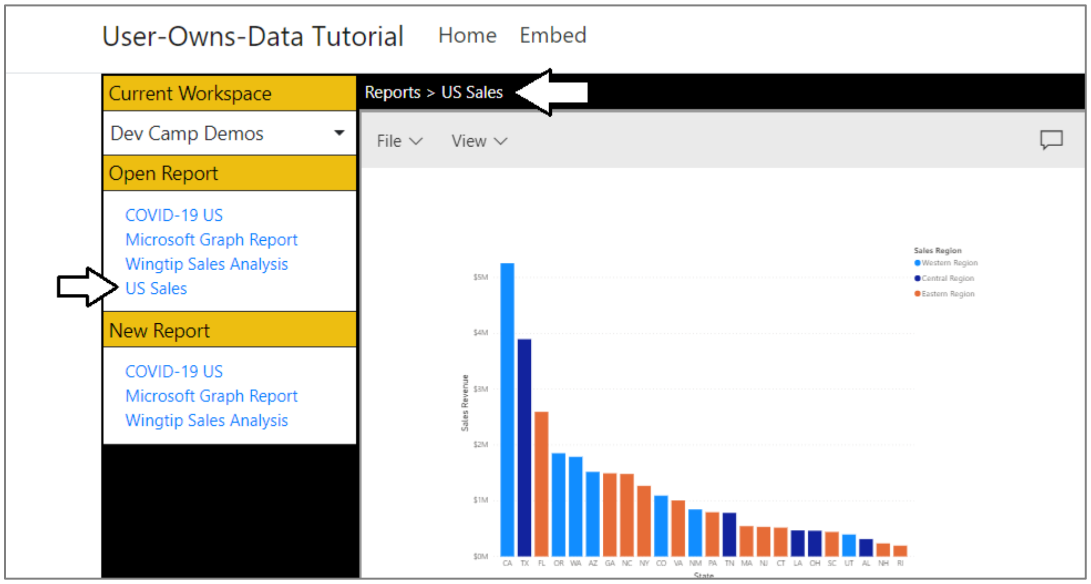
n) Save the new report using the **File > Save as** menu command.



o) Give your new report a name.



p) After you click save, the new report should show up in the Open Report section and be displayed in read-only mode.



q) When you're done testing, close the browser, return to Visual Studio Code and stop the debug session.

Congratulations. You have now completed this lab and have gained cutting-edge experience developing with Power BI embedding