## Developing for User-Owns-Data Embedding using .NET Core

In this lab, you will create a new .NET Core project and go through all 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. Then you will work through creating 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 exercise of the lab, you will support for npm, TypeScript and webpack so that you can move the client-side code from JavaScript to TypeScript where your code receives the benefits of strong typing, IntelliSense and compile-time type checks.

To complete this lab, your developer workstation must configure to allow the execution of PowerShell scripts. Your developer workstation must also have the following software and developer tools installed.

1) **PowerShell cmdlet library for AzureAD** – [[download](https://docs.microsoft.com/en-us/powershell/azure/active-directory/install-adv2?view=azureadps-2.0)]

2) **DOTNET Core SDK 3.1 or later** – [[download](https://dotnet.microsoft.com/download)]

3) **Node.js** – [[download](https://nodejs.org/en/download/)]

4) **Visual Studio Code** – [[download](https://code.visualstudio.com/Download)]

5) **Visual Studio 2019 (optional)** – [[download](https://visualstudio.microsoft.com/downloads/)]

Please refer to the setup document for this lab if you need more detail on how to configure your developer workstation.

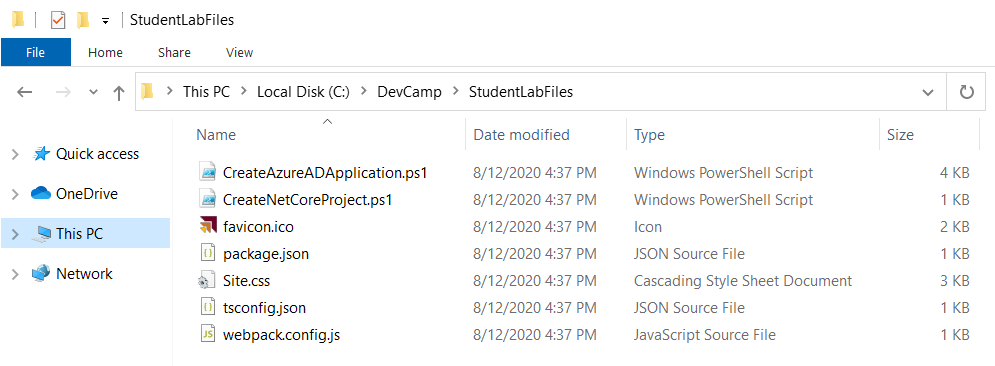
### Exercise 1: Create a New .NET Core MVC Web Application Project

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

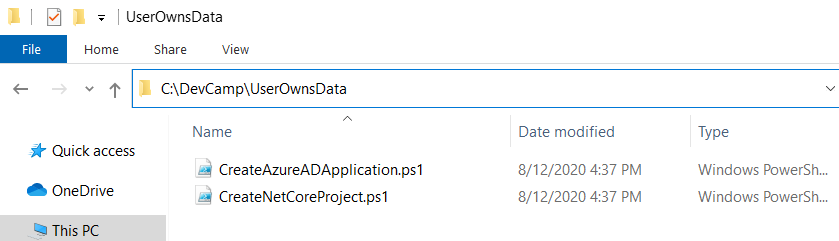
1. Download the student lab files to a local folder on your developer workstation.
   1. Create a new top-level folder on your workstation named **DevCamp** at a location such as **c:\DevCamp**.
   2. Download the ZIP archive with the student lab files from GitHub by clicking the following link.

<https://github.com/TedPattison/NetCore-UserOwnsData-Tutorial/raw/master/StudentLabFiles.zip>

* 1. Extract the **StudentLabFiles** folder from **StudentLabFiles.zip** into a to a local folder such as **c:\DevCamp\StudentLabFiles**.
  2. The **StudentLabFiles** folder should contain the set of files shown in the following screenshot.



1. Create a new .NET Core project using the .NET Core CLI and a PowerShell script.
   1. Create a new folder on your local drive named **UserOwnsData** at a location such as **c:\DevCamp\UserOwnsData**.
   2. In the **StudentLabFiles** folder, locate the scripts named **CreateNetCoreProject.ps1** and **CreateAzureADApplication.ps1**.
   3. Copy **CreateNetCoreProject.ps1** and **CreateAzureADApplication.ps1** into the **UserOwnsData** folder.



1. Review the PowerShell code in **CreateNetCoreProject.ps1**.
   1. Open **CreateNetCoreProject.ps1** in a text editor such asNotepad or the PowerShell Integrated Scripting Environment (ISE).
   2. Review the code in **CreateNetCoreProject.ps1** which creates a new .NET Core project and add a few .NuGet packages.

dotnet new mvc --auth SingleOrg --framework netcoreapp3.1

dotnet remove package Microsoft.AspNetCore.Authentication.AzureAD.UI

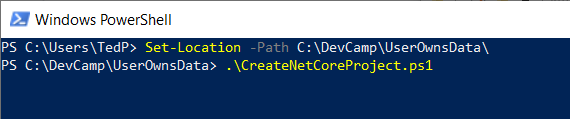
# update to latest available version of Microsoft.Identity.Web

dotnet add package Microsoft.Identity.Web -v 0.2.3-preview

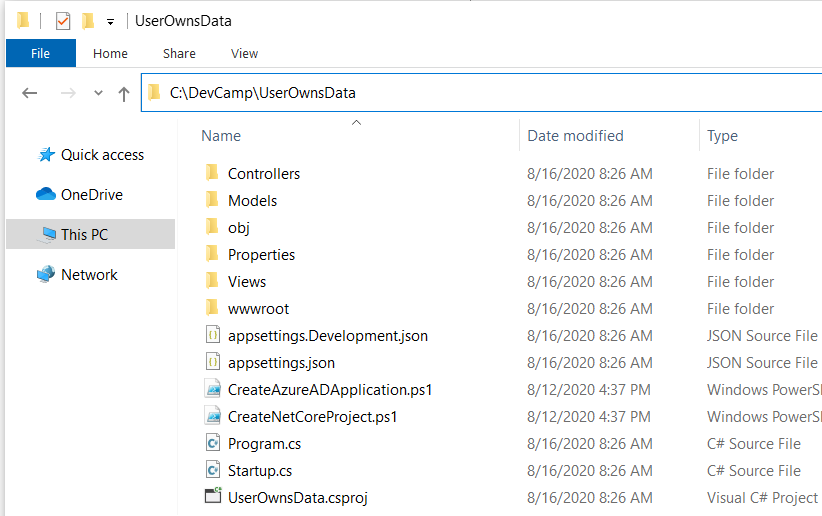
dotnet add package Microsoft.Identity.Web.UI -v 0.2.3-preview

dotnet add package Microsoft.PowerBi.Api

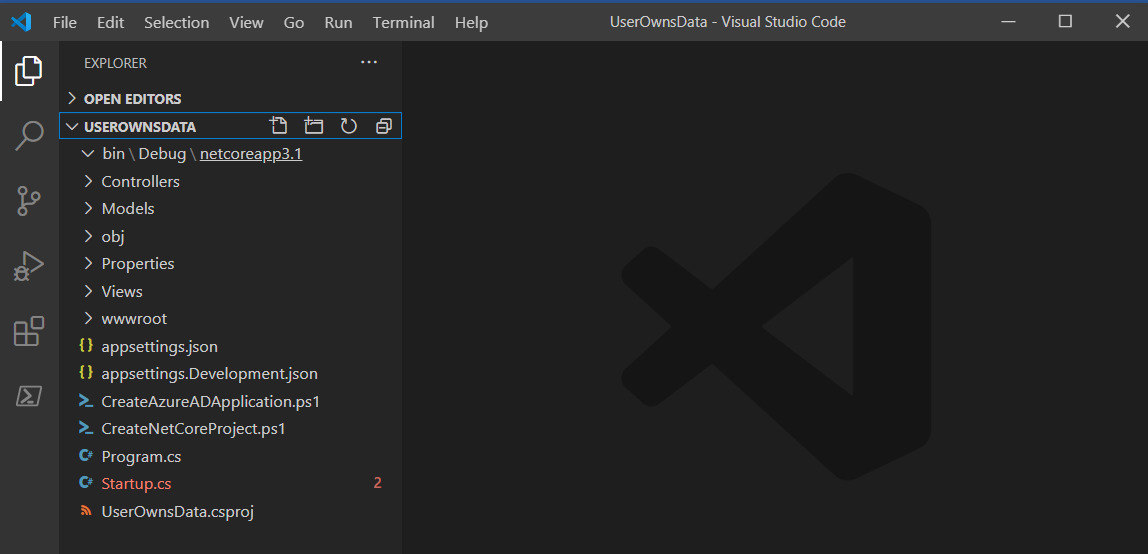
* 1. Open up a PowerShell console and set the location of the command prompt to the **UserOwnsData** folder.
  2. Execute the script **CreateNetCoreProject.ps1** and issuing the command **.\CreateNetCoreProject.ps1**.



* 1. Once the script has completed, you should see that the **UserOwnsData** folder has been populated with project files.



1. Open the **UserOwnsData** folder with Visual Studio Code
   1. Launch Visual Studio Code.
   2. Use the **Open Folder** command in Visual Studio Code to open the **UserOwnsData** folder.



You will not be able to build the project yet. That is because the PowerShell script removed the .NuGet package for Microsoft.AspNetCore.Authentication.AzureAD.UI and added two new packages for the Microsoft.Identity.Web library. You will have to modify the code in the project before it will build.

### Exercise 2: Implement User Login using Microsoft.Identity.Web

In this exercise, you start by running a PowerShell script to create a new confidential client application in Azure AD. After that, you will configure your project to implement an interactive user login experience with Azure AD by using the Microsoft.Identity.Web library.

1. Create a new Azure AD application by running the PowerShell script named **CreateAzureADApplication.ps1**.
   1. Open the PowerShell script named **CreateAzureADApplication.ps1** in a text editor such asNotepad or the PowerShell ISE.
   2. The script begin by **calling Connect-AzureAD** to establish a connection with Azure AD.

$authResult = Connect-AzureAD

* 1. The script contains two variables to set the application 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**.

* 1. 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

* 1. 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 `

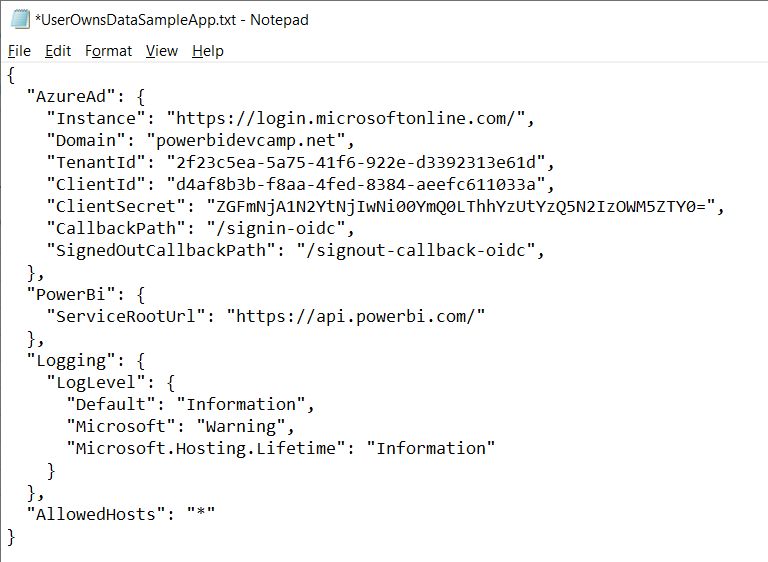
-AvailableToOtherTenants $false `

-ReplyUrls @($replyUrl) `

-Homepage $replyUrl `

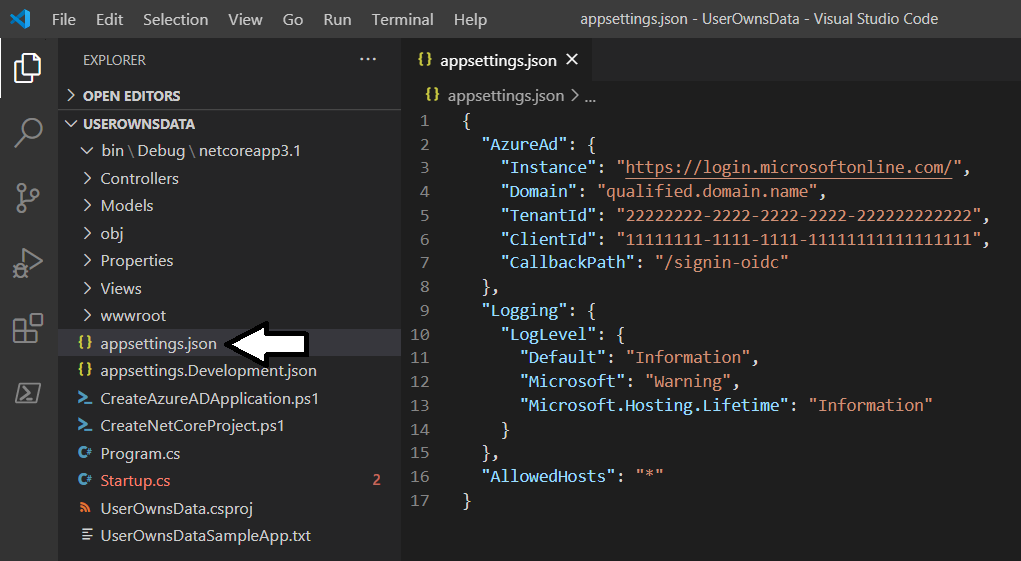
-PasswordCredentials $passwordCredential

* 1. Execute the PowerShell script named **CreateAzureADApplication.ps1**.
  2. When the PowerShell scriptruns successfully, it will create and open a text file named **UserOwnsDataSampleApp.txt**.

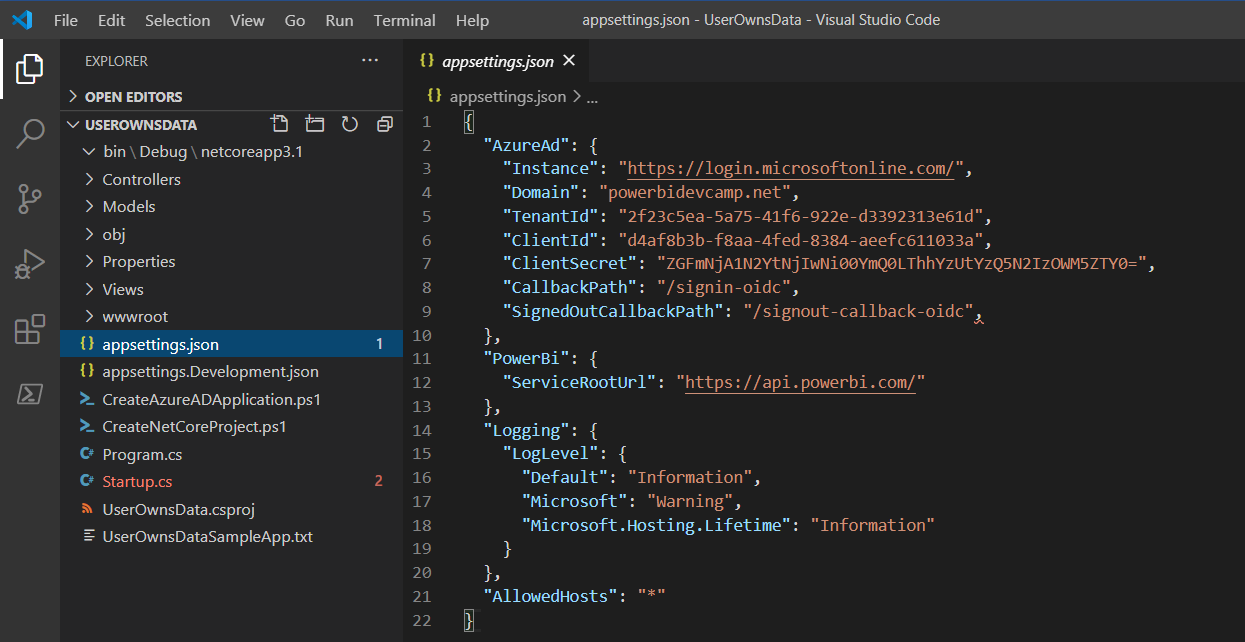


The text file **UserOwnsDataSampleApp.txt** contains JSON configuration that you will copy and paste into **appsettings.json**.

1. Copy the JSON in **UserOwnsDataSampleApp.txt** into the **appsettings.json** file in your project.
   1. Return to the **UserOwnsData** project in Visual Studio Code and open the **appsettings.json** file.
   2. The **appsettings.json** file should initially appear like the screenshot below.

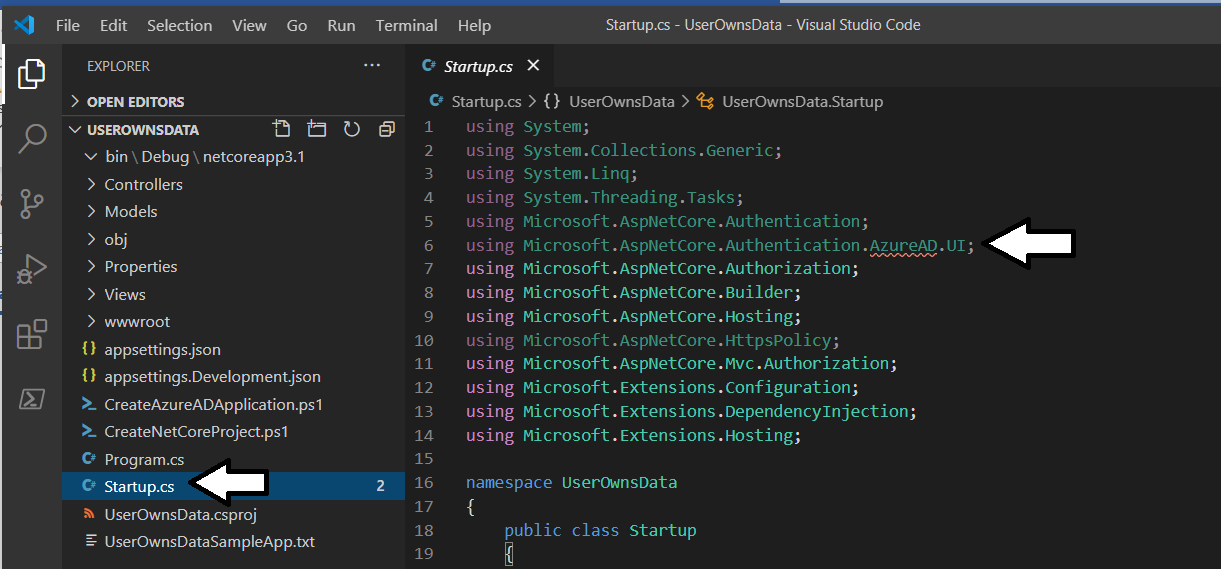


* 1. Delete the contents of **appsettings.json** and replace it by copying and pasting the contents of **UserOwnsDataSampleApp.txt**

****

Note the **PowerBi:ServiceRootUrl** parameter has been added as a custom configuration value to track the base URL to the Power BI Service. When you are programming against the Power BI Service in Microsoft public cloud, the URL is <https://api.powerbi.com/>. However, the root URL for the Power BI Service 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 whenever required..

1. Modify the code in **Startup.cs** to properly initialize the authentication service provided by **Microsoft.Identoty.Web**.
   1. Open the **Startup.cs** file in an editor window.
   2. Remove line that imports **Microsoft.AspnetCore.Authentication.AzureAD.UI** which is causing a build error.



* 1. Please your cursor below the existing **using** statements and add the following **using** statements.

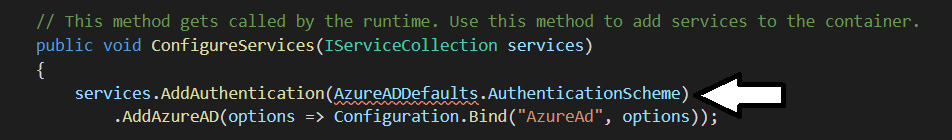
using Microsoft.Identity.Web;

using Microsoft.Identity.Web.UI;

using Microsoft.Identity.Web.TokenCacheProviders;

using Microsoft.Identity.Web.TokenCacheProviders.InMemory;

* 1. Move down into the **ConfigureServices** method and locate and delete the code which calls **services.AddAuthentication**.



* 1. Replace the call you removed with a call to **services.AddMicrosoftWebAppAuthentication**.as shown in the following listing.

// This method gets called by the runtime. Use this method to add services to the container.

public void ConfigureServices(IServiceCollection services) {

services.AddMicrosoftWebAppAuthentication(Configuration);

* 1. Move down in the **ConfigureServices** method and locate the code which calls **services.AddControllersWithViews**.

services.AddControllersWithViews(options =>

{

var policy = new AuthorizationPolicyBuilder()

.RequireAuthenticatedUser()

.Build();

options.Filters.Add(new AuthorizeFilter(policy));

});

* 1. Replace this code with the code shown in the following listing.

var mvcBuilder = services.AddControllersWithViews(options => {

var policy = new AuthorizationPolicyBuilder()

.RequireAuthenticatedUser()

.Build();

options.Filters.Add(new AuthorizeFilter(policy));

});

mvcBuilder.AddMicrosoftIdentityUI();

* 1. At this point, the **ConfigureServices** method in your project should match what is shown in the following code listing.

public void ConfigureServices(IServiceCollection services) {

services.AddMicrosoftWebAppAuthentication(Configuration);

var mvcBuilder = services.AddControllersWithViews(options => {

var policy = new AuthorizationPolicyBuilder()

.RequireAuthenticatedUser()

.Build();

options.Filters.Add(new AuthorizeFilter(policy));

});

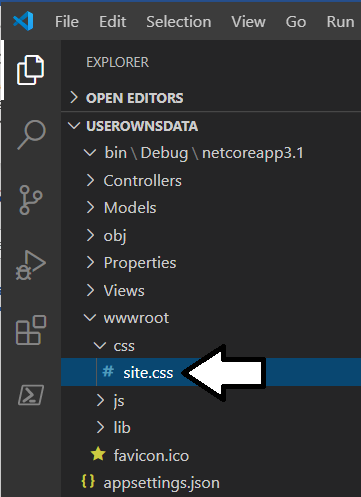
mvcBuilder.AddMicrosoftIdentityUI();

services.AddRazorPages();

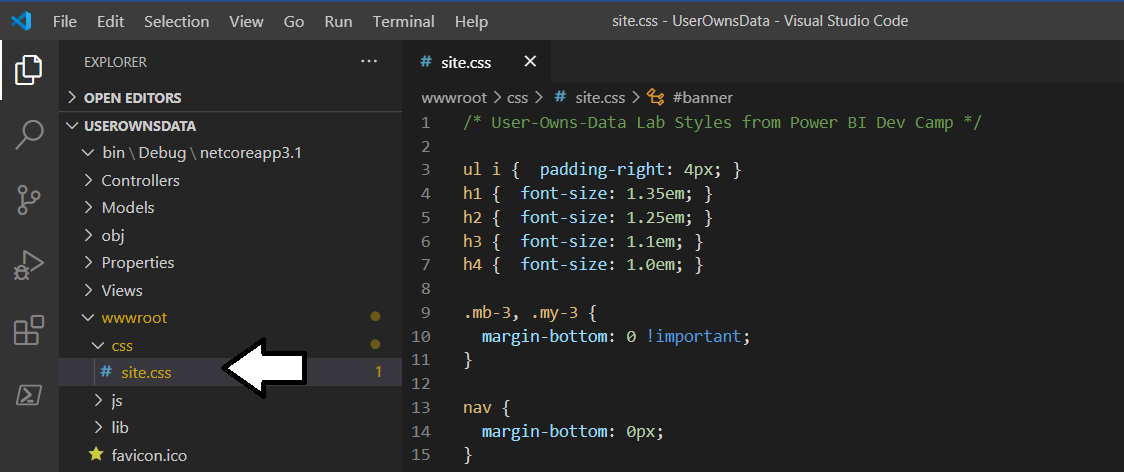
}

* 1. Save your changes and close **Startup.cs**.

1. Copy a pre-written set of CSS styles into the **UserOwnsData** project's **Site.css** file.
   1. Expand the **wwwroot** folder and then inside that expand the **css** folder to examine the contents of the **wwwroot/css** folder.
   2. Locate and open the CSS file that is automatically included in new .NET Core projects named **site.css**.

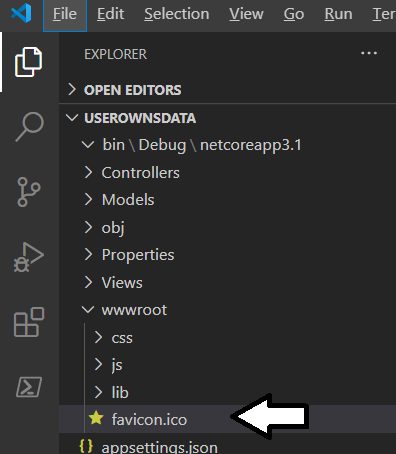


* 1. Using the Windows Explorer, look inside the **StudentLabFiles** folder and locate the file named **site.css**.
  2. Open the **site.css** file in the **StudentLabFiles** folder in a text editor and copy its entire contents into the Windows clipboard.
  3. Copy the CSS styles from the **site.css** file in the **StudentLabFiles** folder into the **site.css** file in your project.



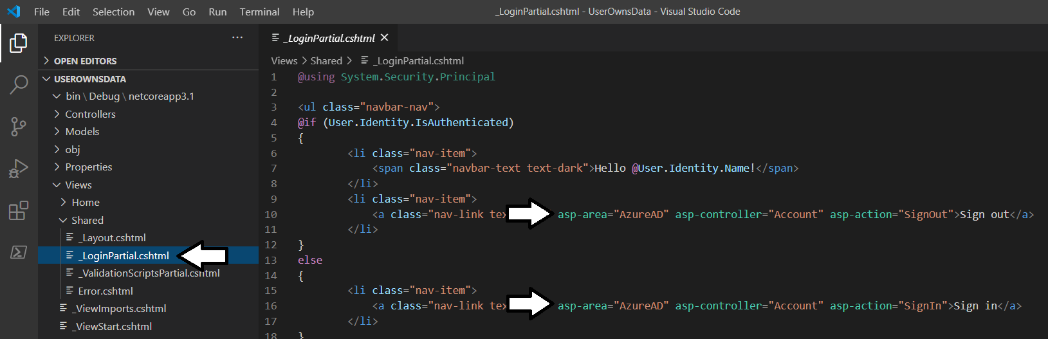
* 1. Save your changes and close **site.css**.

1. Copy a custom **favicon.ico** file to the **wwwroot** folder.
   1. Using the Windows Explorer, look inside the **StudentLabFiles** folder and locate the file named **favicon.ico**.
   2. 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.

1. Modify the partial razor view file named **\_LoginPartial.cshtml** to integrate with the **Microsoft.Identity.Web** authentication library..
   1. Expand the **Views > Shared** folder and locate the partial view named **\_LoginPartial.cshtml**.
   2. Open **\_LoginPartial.cshtml** in an editor window.
   3. In the existing code, you can see that there are two links which have an **asp-area** tag with a value of **AzureAD**.



The **asp-area** value of **AzureAD** is used by the older library named **Microsoft.AspNetCore.Authentication.AzureAD.UI**. Since we are moving to the **Microsoft.Identity.Web** authentication library, the value must be changed from **AzureAD** to **MicrosoftIdentity**.

* 1. Delete all the code **\_LoginPartial.cshtml** and replace it with the code shown in the following code listing.

@using System.Security.Principal

<ul class="navbar-nav">

@if (User.Identity.IsAuthenticated) {

<li class="nav-item">

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

</li>

<li class="nav-item">

<a class="nav-link text-dark" asp-area="MicrosoftIdentity" asp-controller="Account" asp-action="SignOut">

Sign out

</a>

</li>

}

else {

<li class="nav-item">

<a class="nav-link text-dark" asp-area="MicrosoftIdentity" asp-controller="Account" asp-action="SignIn">

Sign in

</a>

</li>

}

</ul>

* 1. Save your changes and close **\_LoginPartial.cshtml**.

1. Modify the HTML in **Index.cshtml** to display differently depending on whether the user has logged in or not.
   1. Expand the **Views > Home** folder and locate the view file named **Index.cshtml**.
   2. Open **Index.cshtml** in an editor window.
   3. 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>ed

}

else {

<div class="jumbotron">

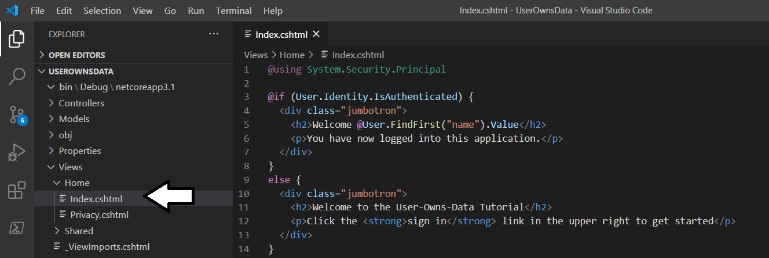
<h2>Welcome to the User-Owns-Data Tutorial</h2>

<p>Click the <strong>sign in</strong> link in the upper right to get started</p>

</div>

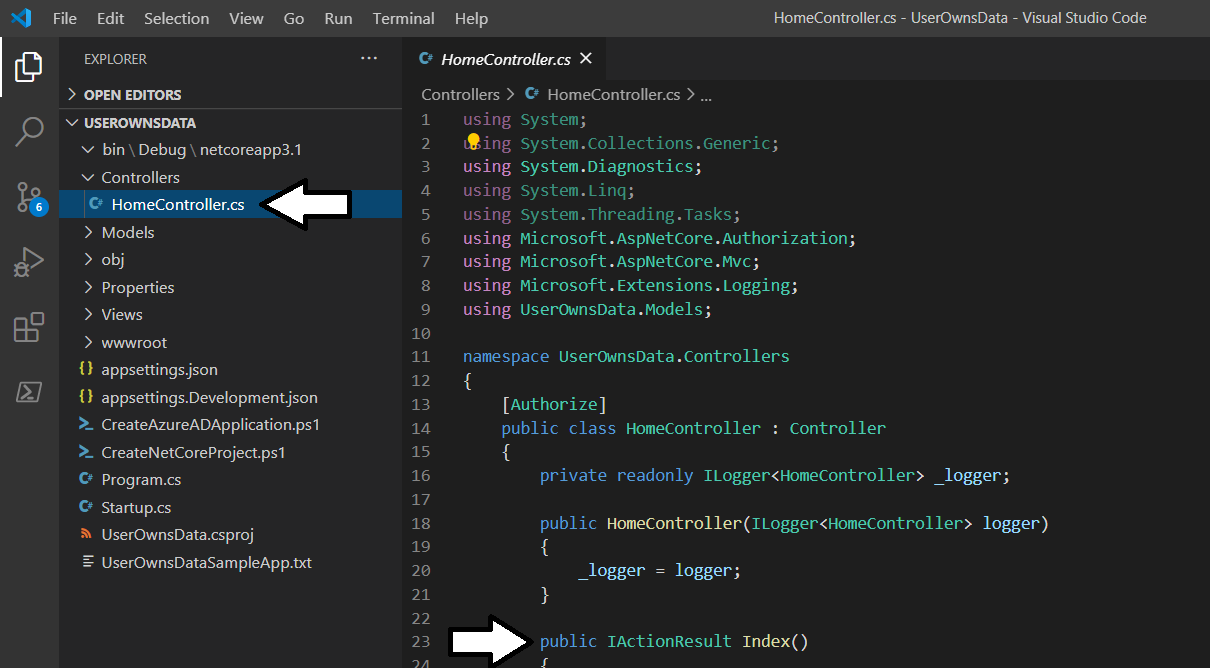
}

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



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

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



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

[AllowAnonymous]

public IActionResult Index()

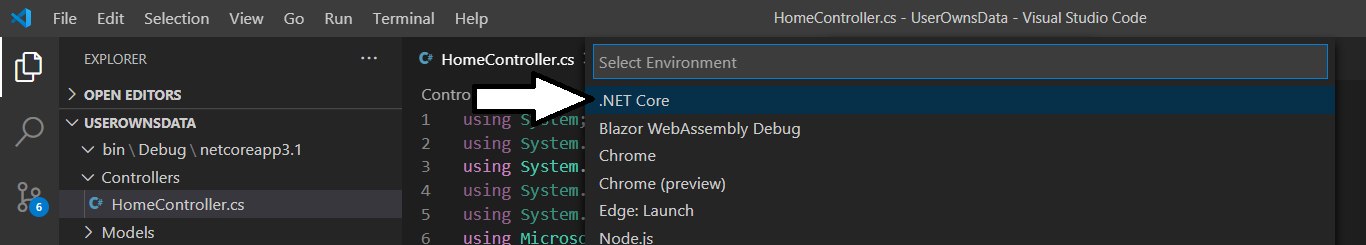
{

return View();

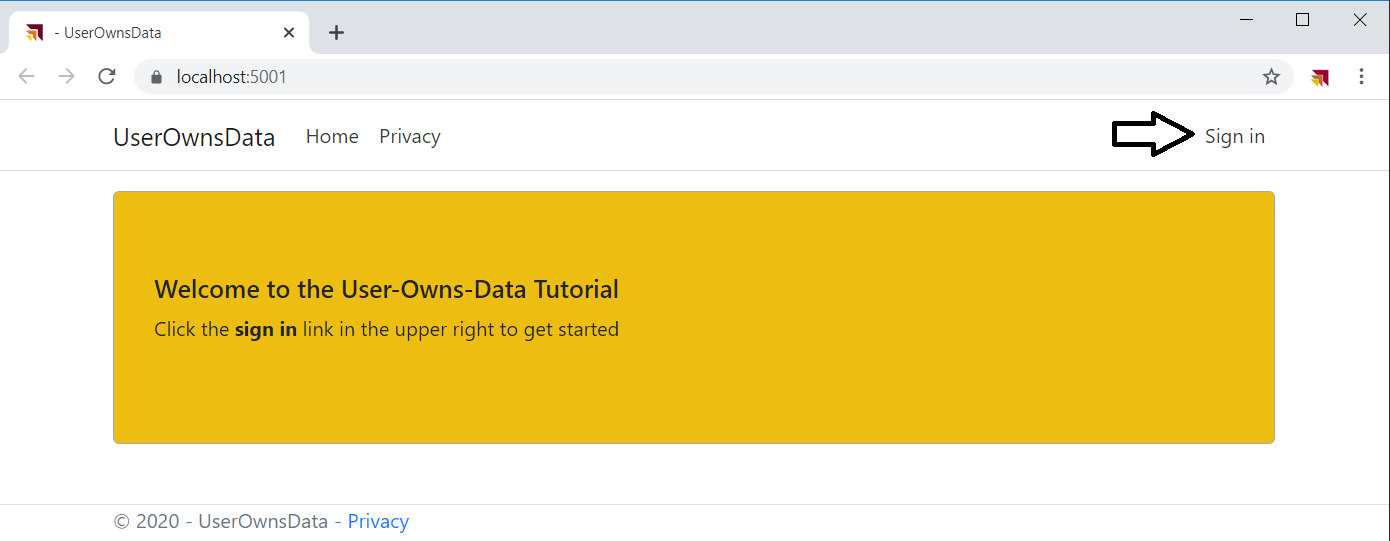
}

* 1. Save your changes and close **HomeController.cs**.

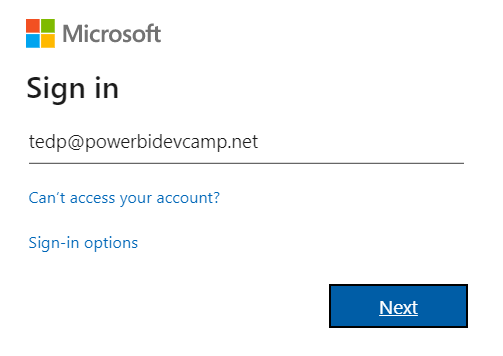
1. Test
   1. Press {F5}



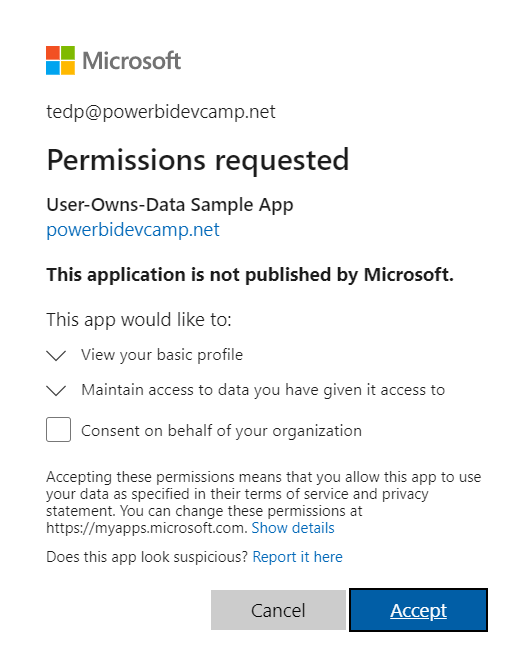
* 1. X



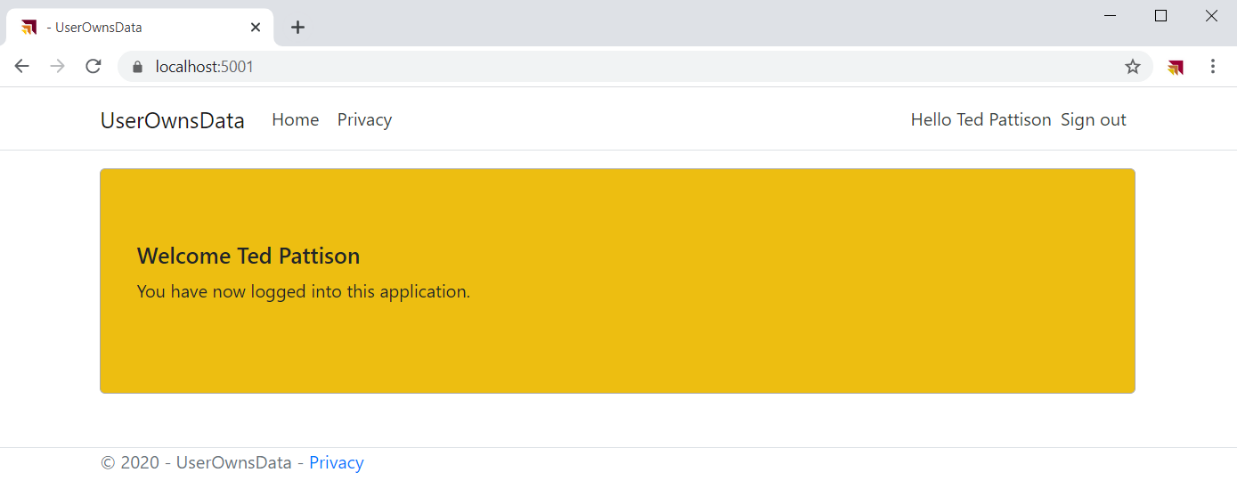
* 1. X



* 1. X



* 1. X

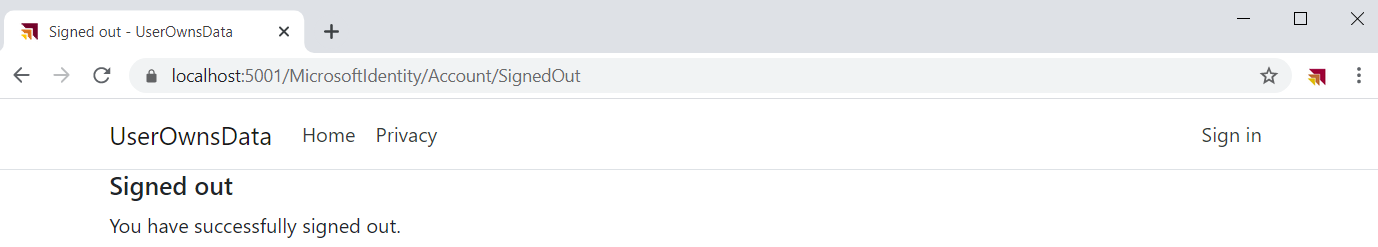


* 1. X

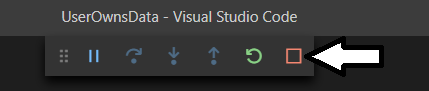
1. Logout
   1. Click the sign out link.



* 1. X



* 1. Close the browser and then return to Visual Studio Code and terminate the debug session and Shut down the



* 1. X

In the next steps, you will create new page. Rename Privicay to Embed. Update Home controller class and then update name of view named Priavacy.cshtml to Embed.cshtml.

1. Add Embed page
   1. Open HomeController.cs.
   2. Locate this code.

[AllowAnonymous]

public IActionResult Index() {

return View();

}

public IActionResult Privacy() {

return View();

}

* 1. Change the name of the Privacy method to Embed. No changes to the code are required.

[AllowAnonymous]

public IActionResult Index() {

return View();

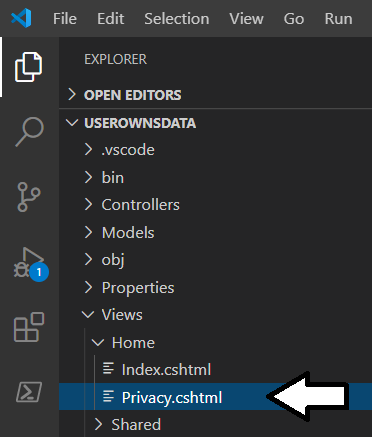
}

public IActionResult Embed() {

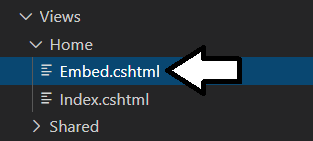
return View();

}

* 1. Locate view



* 1. X

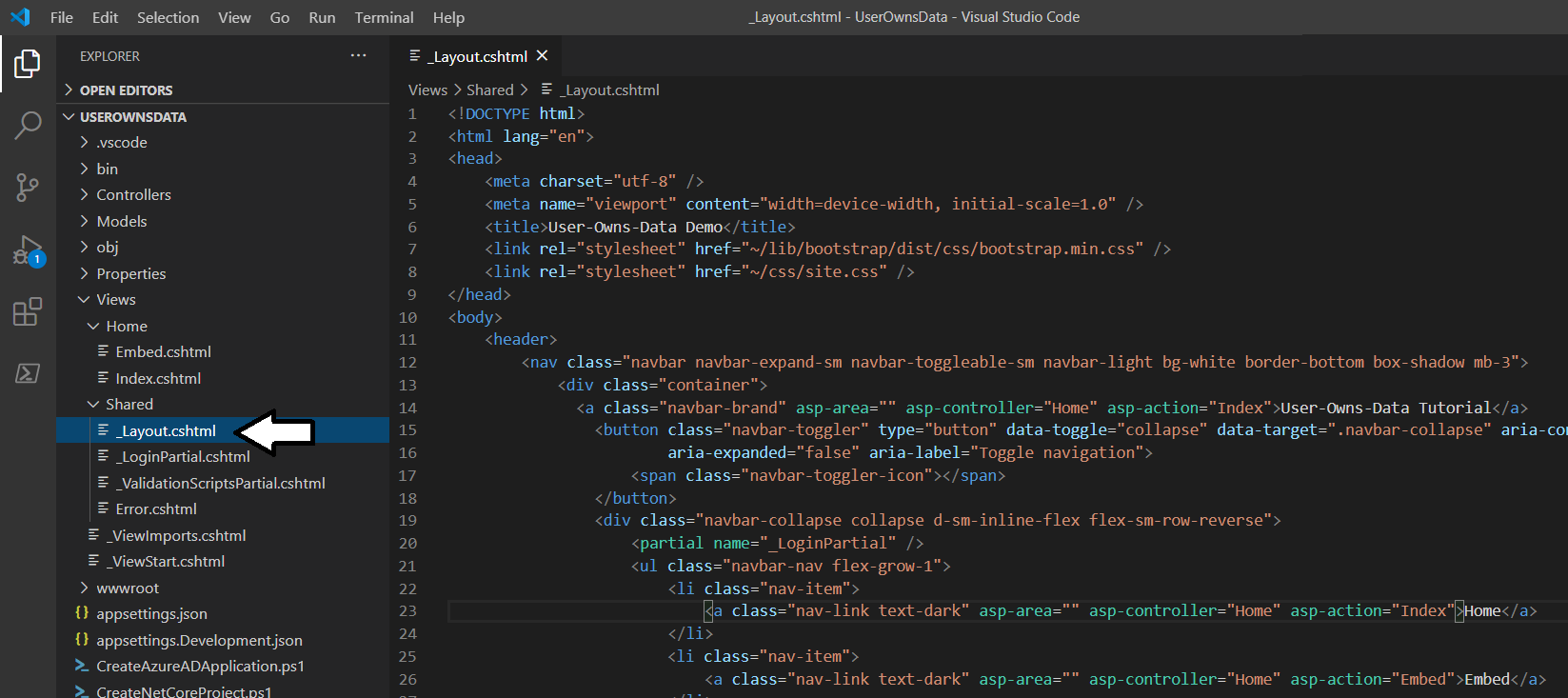


* 1. Add Modify Embed.cshtml

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

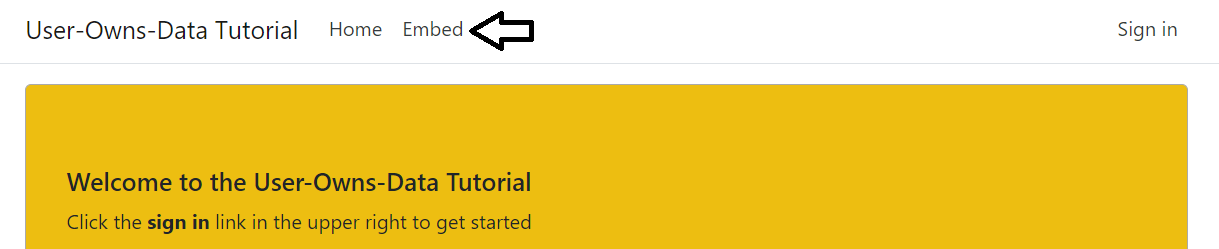
* 1. Save and close.

1. Modify Navigation in \_Layout.cshtml
   1. Copy the code in \_Layout.cshtml from the StudentLabFiles folder.

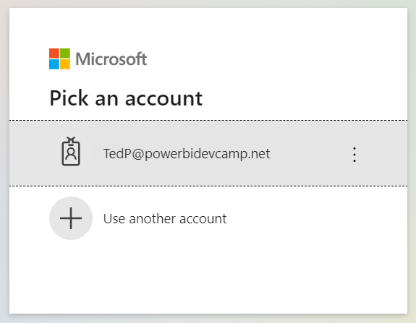


* 1. Save and close

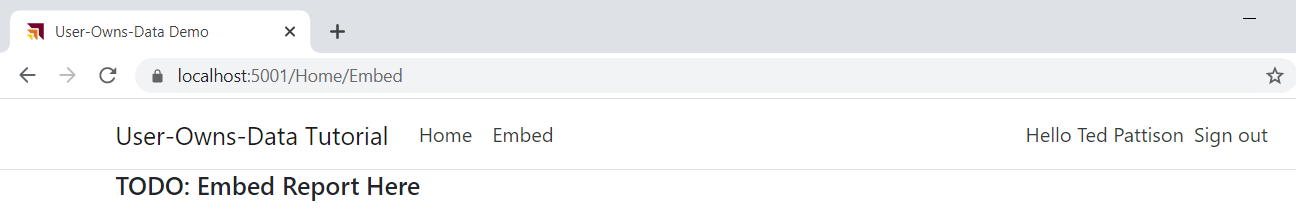
1. Test
   1. Ssssss



* 1. Ss



* 1. Xxx

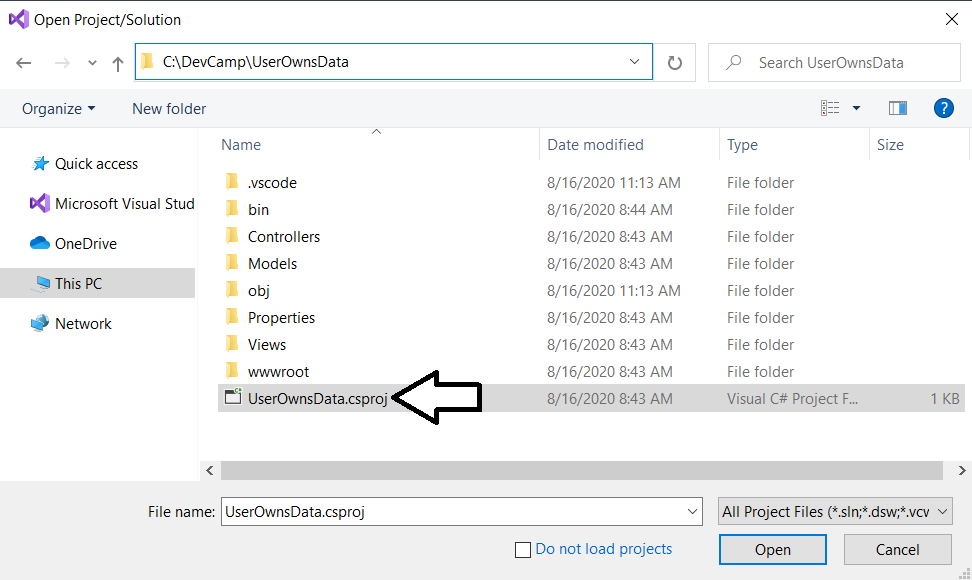


* 1. x

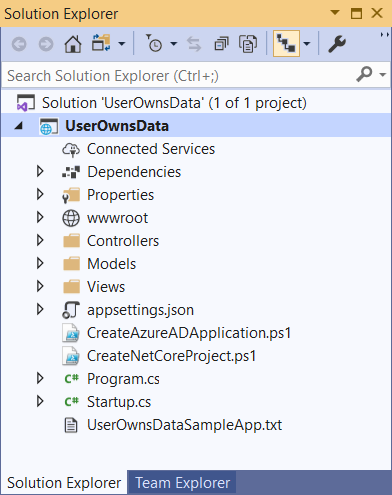
1. [OPTIONAL STEP] Open the and run the project using Visual Studio
   1. Sssss



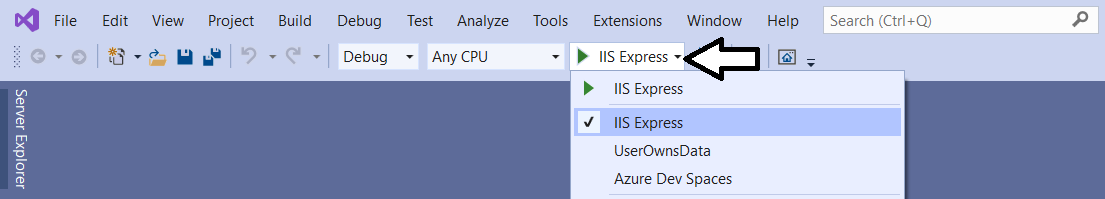
* 1. X



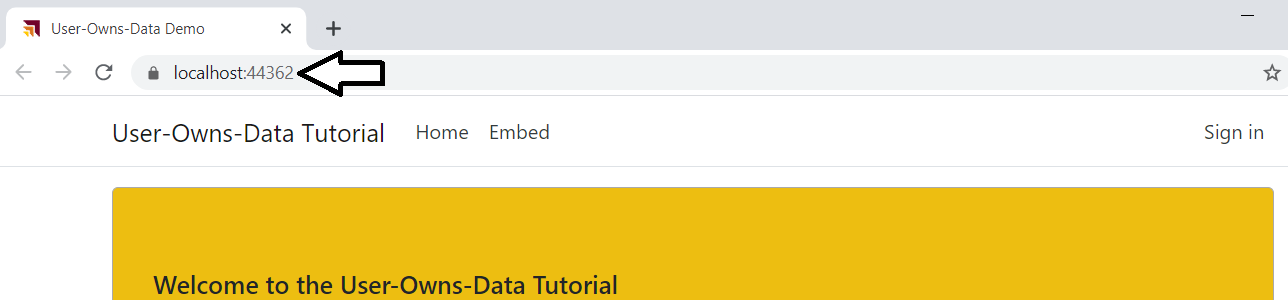
* 1. X



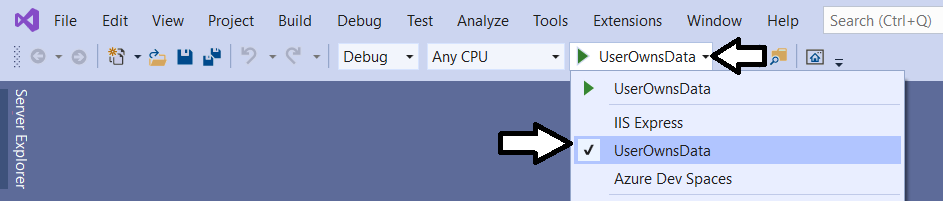
* 1. X



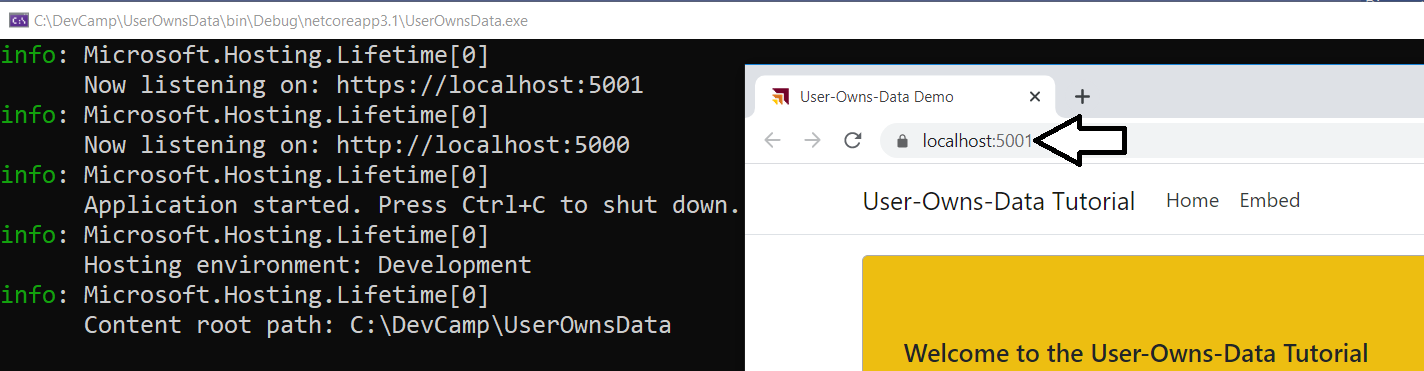
* 1. X



* 1. X



* 1. X



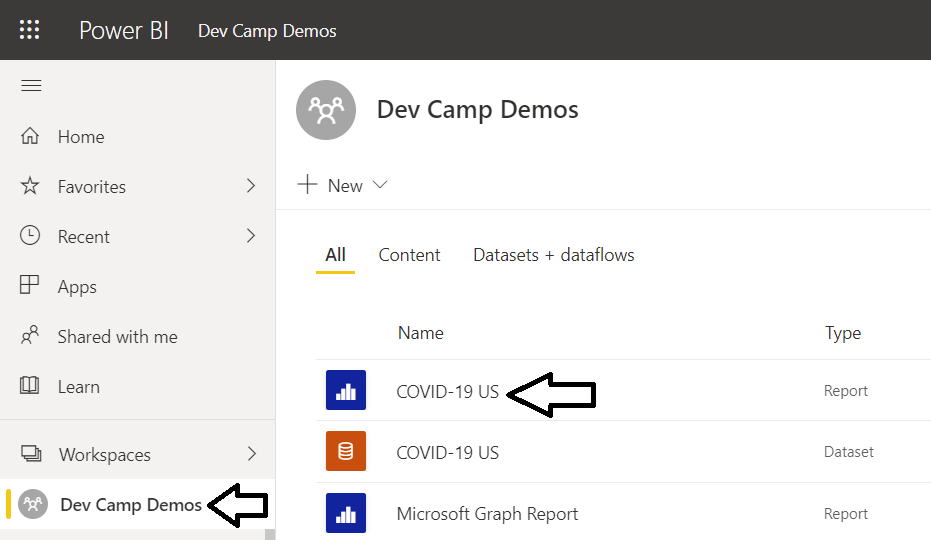
* 1. X

This lab will return to using Visual Studio Code. However, you should be able to complete the same work in Visual Studio 2019 if you like that environment better..

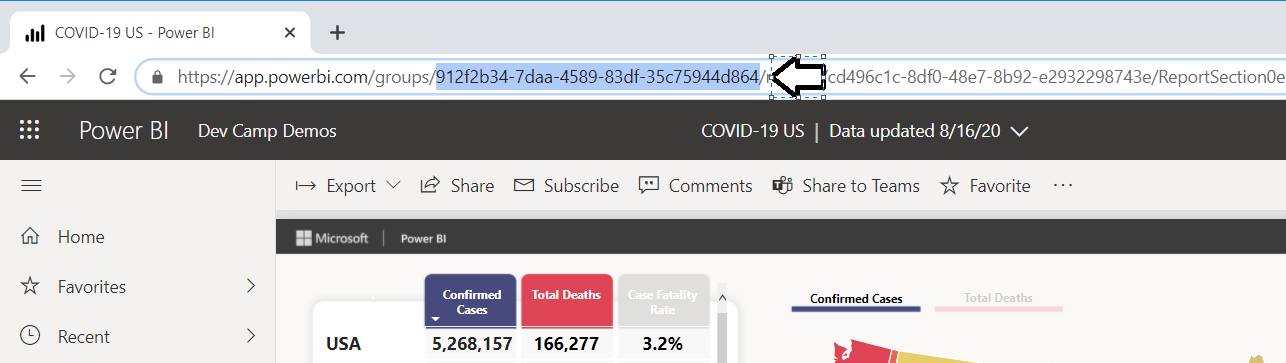
### Exercise 3: Call the Power BI Service API

In this exercise, you will create a new confidential client application in the Azure portal and you will configure the application’s required permissions to provide the access you need to call into the Power BI Service API.

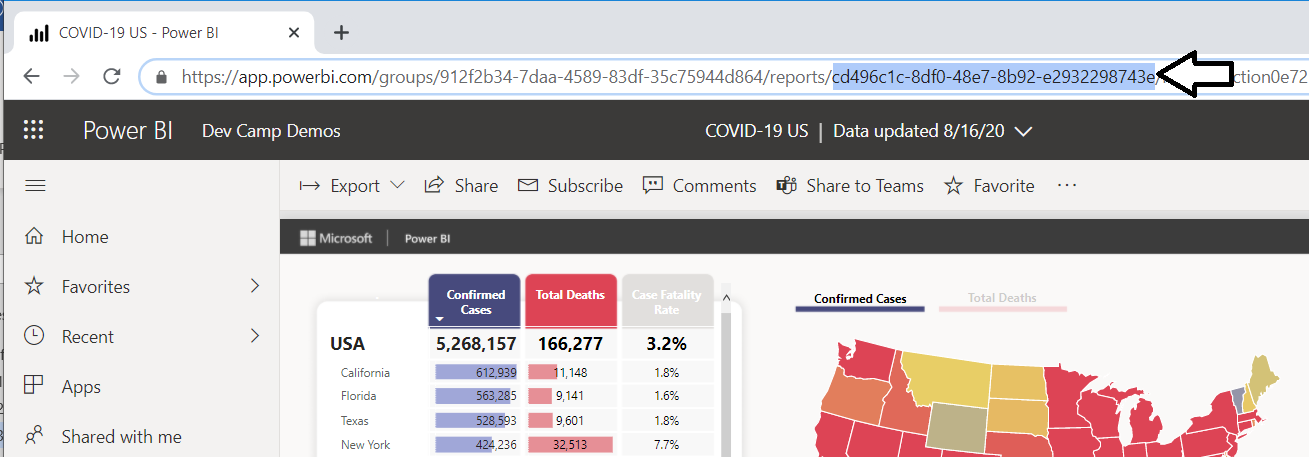
1. In Power BI Service, create a new workspace and add a report.
   1. Get GUIDs for these



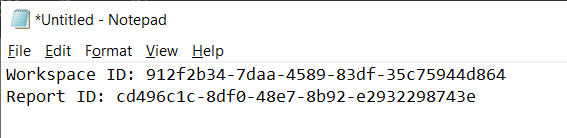
* 1. X



* 1. X

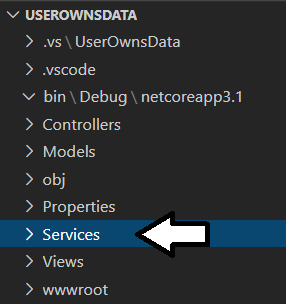


* 1. X

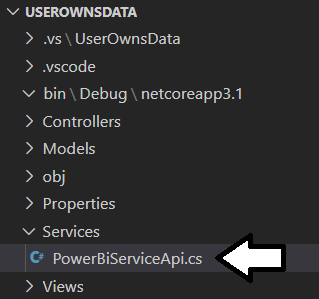


* 1. x

1. Add a new service for PowerBiApi.cs.
   1. Create a Service folder in project.



* 1. Ssss



* 1. Ss

using System;

using System.Threading.Tasks;

using Microsoft.Identity.Web;

using Microsoft.Rest;

using Microsoft.PowerBI.Api;

namespace UserOwnsData.Services {

public class EmbeddedReportViewModel {

}

public class PowerBiServiceApi {

}

}

* 1. Sss

public class EmbeddedReportViewModel {

public string Id;

public string Name;

public string EmbedUrl;

public string Token;

}

* 1. Sss

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;

}

}

* 1. Xxx

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"

};

* 1. Xxx

public string GetAccessToken() {

return this.tokenAcquisition.GetAccessTokenForUserAsync(RequiredScopes).Result;

}

* 1. ddd

public PowerBIClient GetPowerBiClient() {

var tokenCredentials = new TokenCredentials(GetAccessToken(), "Bearer");

return new PowerBIClient(new Uri(urlPowerBiServiceApiRoot), tokenCredentials);

}

* 1. ssss

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()

};

}

* 1. Save and close PowerBIServiceApi.cs.

1. Modify Startup.cs to register service
   1. Underneath the existing import statements, add the following import statement;

using UserOwnsData.Services;

* 1. Dddd

public void ConfigureServices(IServiceCollection services) {

services.AddMicrosoftWebAppAuthentication(Configuration);

* 1. Replace with this.

services

.AddMicrosoftWebAppAuthentication(Configuration)

.AddMicrosoftWebAppCallsWebApi(Configuration, PowerBiServiceApi.RequiredScopes)

.AddInMemoryTokenCaches();

* 1. Now this.

services.AddScoped(typeof(PowerBiServiceApi));

* 1. The whole this

public void ConfigureServices(IServiceCollection services) {

services

.AddMicrosoftWebAppAuthentication(Configuration)

.AddMicrosoftWebAppCallsWebApi(Configuration, 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();

}

1. Modify Home controller to inject PowerBiApi.
   1. Open HomeController.cs.
   2. Add import

using UserOwnsData.Services;

* 1. Locate

[Authorize]

public class HomeController : Controller {

private readonly ILogger<HomeController> \_logger;

public HomeController(ILogger<HomeController> logger) {

\_logger = logger;

}

* 1. Replace with this

[Authorize]

public class HomeController : Controller {

private PowerBiServiceApi powerBiServiceApi;

public HomeController(PowerBiServiceApi powerBiServiceApi) {

this.powerBiServiceApi = powerBiServiceApi;

}

* 1. Replace

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);

}

1. Add HTML to Embed.cshtml to create a table.
   1. Sss

@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>

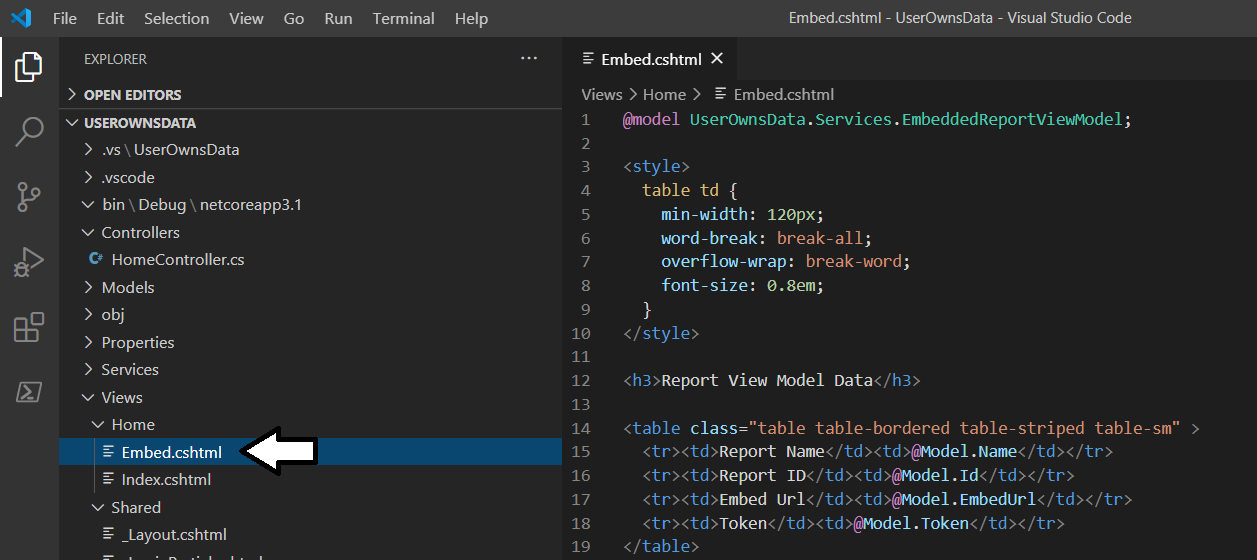
<tr><td>Report ID</td><td>@Model.Id</td></tr>

<tr><td>Embed Url</td><td>@Model.EmbedUrl</td></tr>

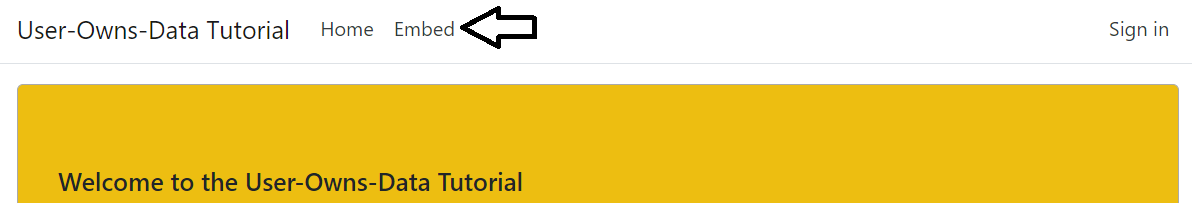
<tr><td>Token</td><td>@Model.Token</td></tr>

</table>

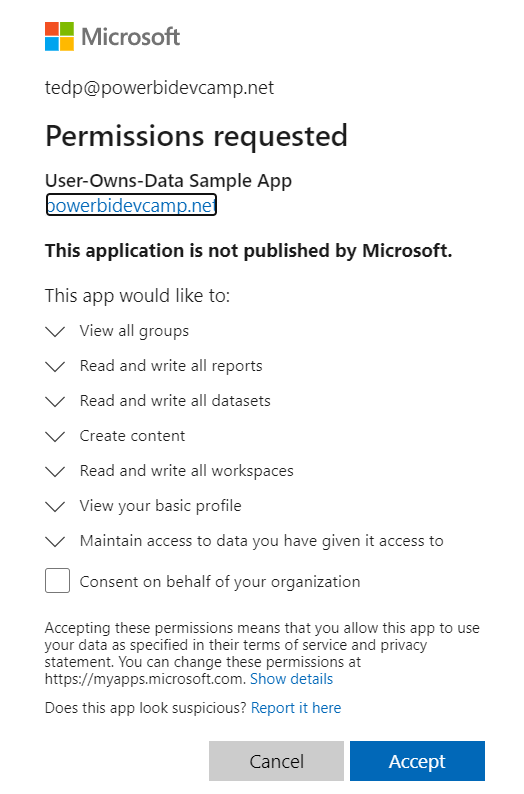
* 1. Xx



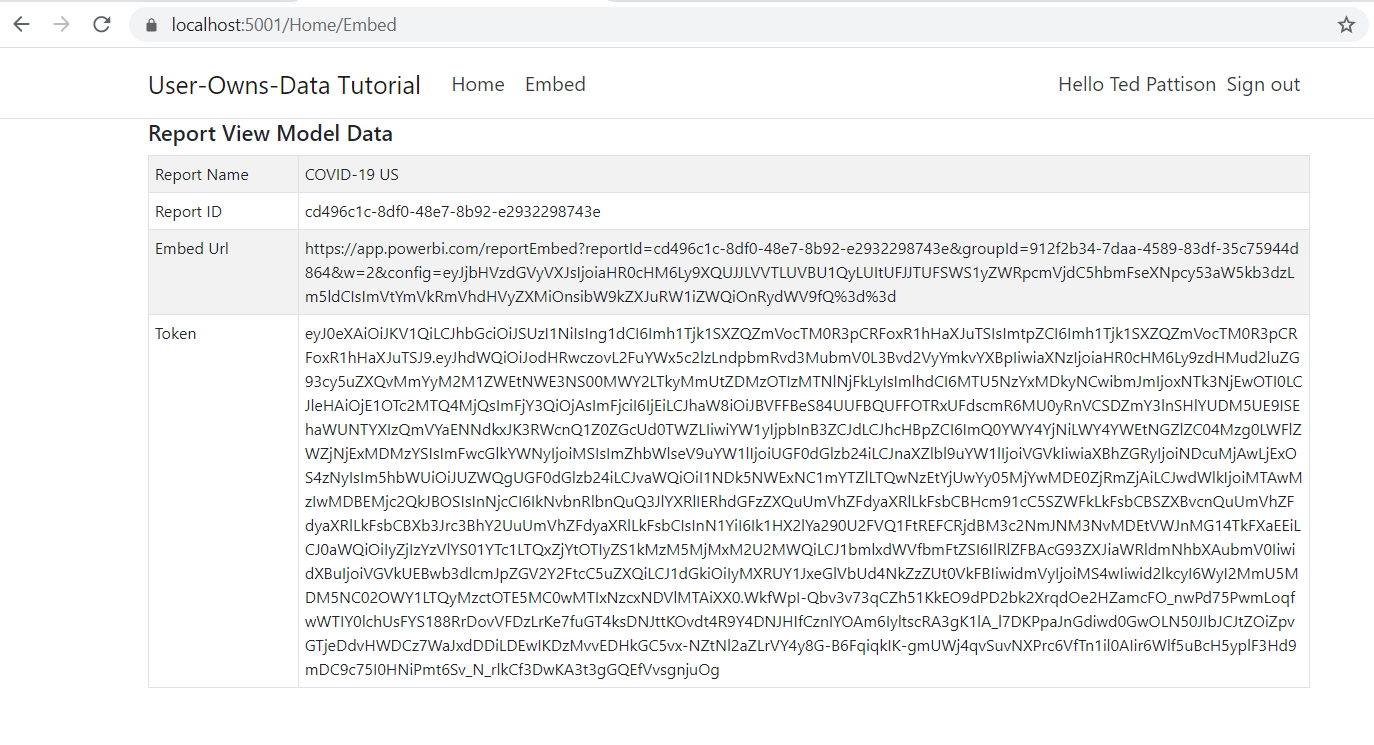
1. Test
   1. Ssss



* 1. S



* 1. X



* 1. x

### Exercise 4: Embedding a Report using powerbi.js

In this exercise, you will create a new confidential client application in the Azure portal and you will configure the application’s required permissions to provide the access you need to call into the Power BI Service API.

1. Add client-side library support with Libman,json
   1. sss
2. Add view model to Embed.cshtml
   1. Sss
3. Add Embed,js
4. Test

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

In this exercise, you will create a new confidential client application in the Azure portal and you will configure the application’s required permissions to provide the access you need to call into the Power BI Service API.

1. Add package.json – npm init
   1. sss
2. Add tsconfig.json
   1. sss
3. Add webpack.config.js
   1. Ssss
4. Add embed.js
   1. Create Script folder
   2. Add embed.js
   3. Compile from command line
5. Update UserOwnsData.csproj
   1. sss

### Exercise 6: Creating a View Model for App Workspaces

In this exercise, you will create a new confidential client application in the Azure portal and you will configure the application’s required permissions to provide the access you need to call into the Power BI Service API.

1. Add new method to PowerBiApi.cs
   1. Sss
2. Modify Embed method in HomeController to call new method.
   1. Pass view model as object not as string
3. Rewrite Embed.cshtml
4. Rewrite Embed.ts.