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

In this lab

To complete this lab, you must be able to run PowerShell scripts on your developer workstation. You must also install the following software if it is not already installed.

1) PowerShell cmdlet library for AzureAD

2) DOTNET Core SDK

3) Node.js

4) Visual Studio Code

5) Visual Studio 2019 (optional)

Please refer to the setup document for this lab to ensure you have all the

### Exercise 1: Create a New .NET Core MVC Web Application 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. Aaaaa
2. sssss

### Exercise 2: Authenticate the User using Microsoft.Identity.Web

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. Dddd
2. hhh

### 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. Xxx
2. xxxx

### 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. Ss
2. ss

### 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. Xxxxx
2. ddddd

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

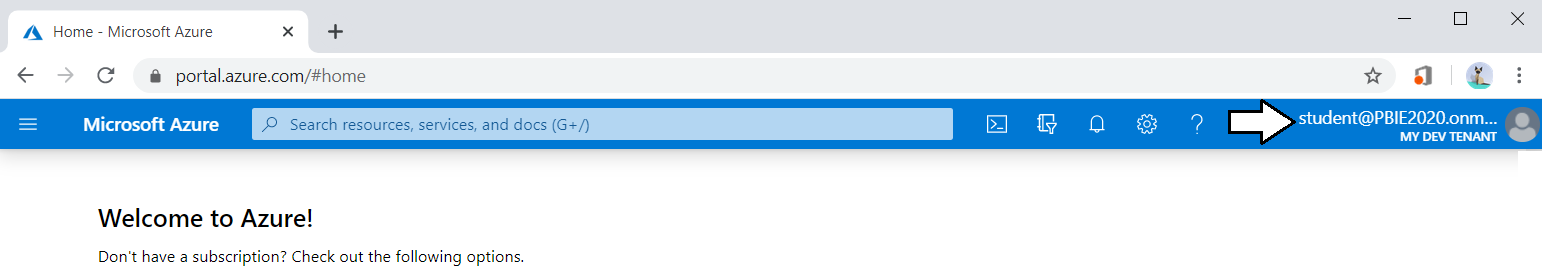
In this exercise, you will create a new confidential client application in the Azure portal and you will configure the application’s required permissions to provide the access you need to call into the Power BI Service API.

1. Sssss
2. sssss

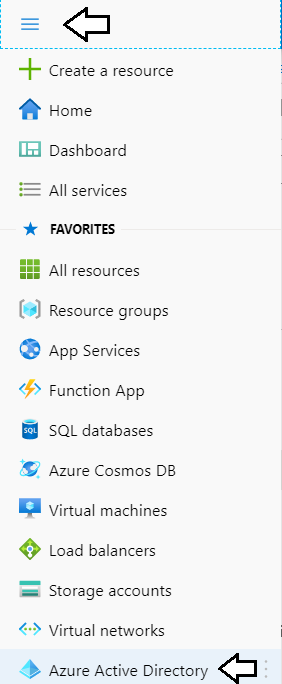
### Exercise 1: Register a New Azure AD Application in the Azure Portal

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.

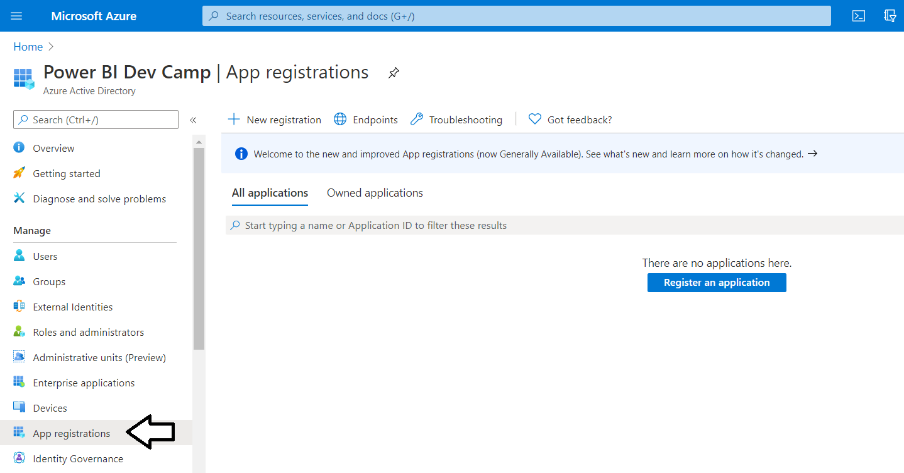
1. Log into the Azure Portal
   1. In the browser, navigate to the Azure portal at <https://portal.azure.com>.
   2. When you are prompted to log in, provide the credentials to log in with your Office 365 user account name.
   3. Once you have logged into the Azure portal, check the email address in the login menu in the upper right to make sure you are logged in with the correct identity for your new Office 365 user account.



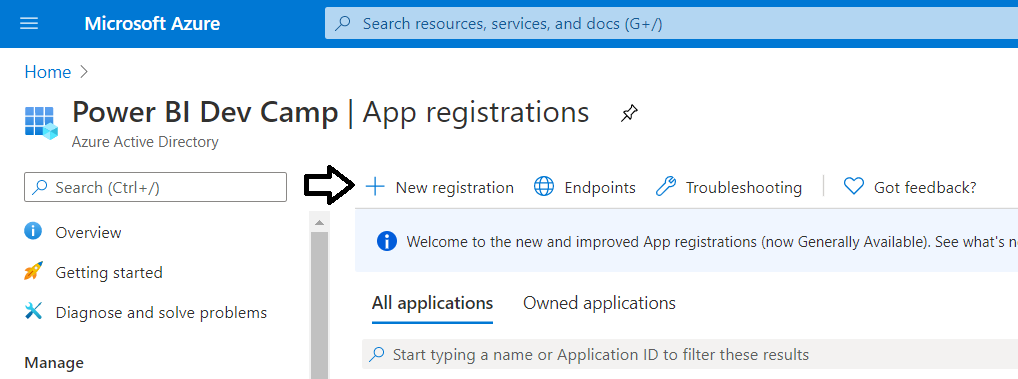
1. Register a new Azure AD application.
   1. Click hamburger icon in the top left corner of the page to drop down the Azure portal navigation menu.
   2. In the left navigation drop down, click the link for **Azure Active Directory**.



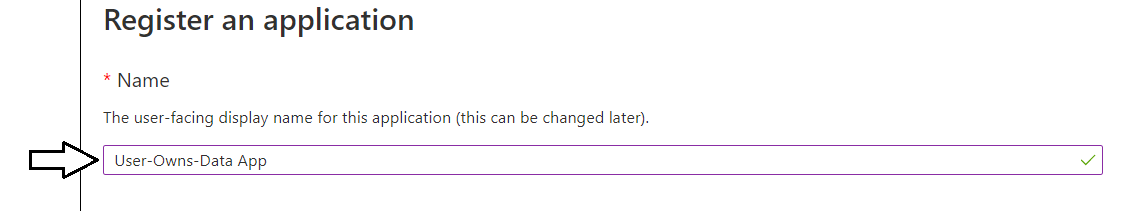
* 1. Once you’ve navigated to **Azure Active Directory** in the Azure portal, click **App registrations** in the left navigation.



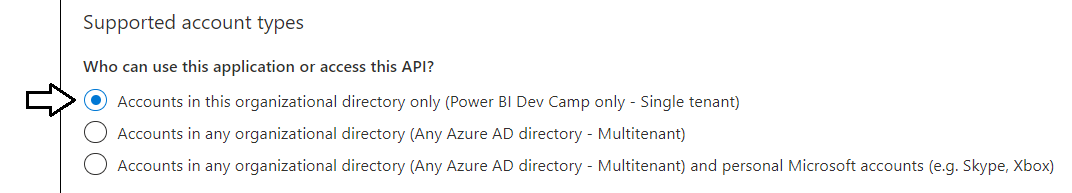
* 1. Click **New registration**.



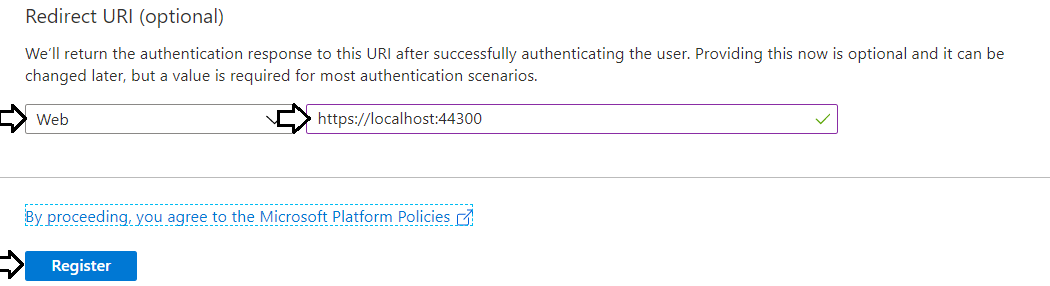
* 1. Enter a **Name** of **App-Owns-Data App**.



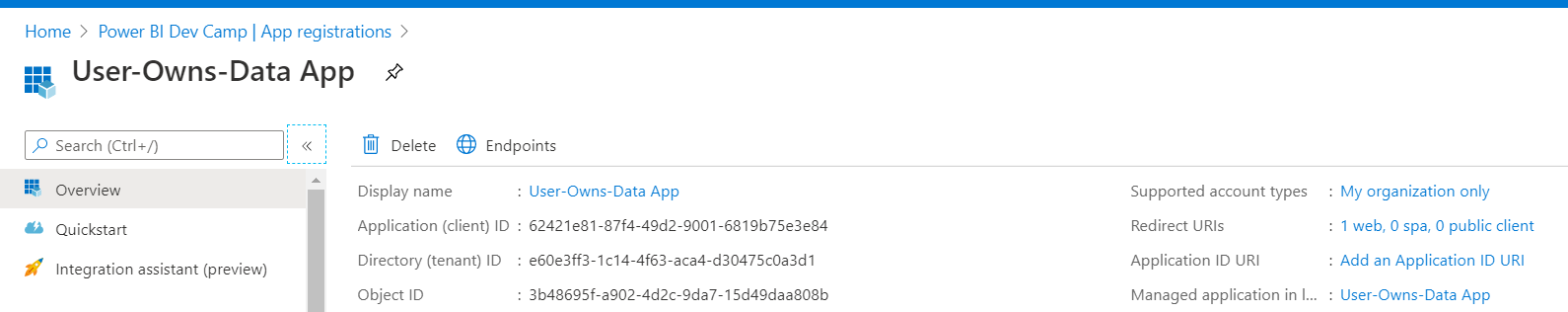
* 1. For the **Supported account types** option, leave the default value of **Accounts in this organizational directory only**.



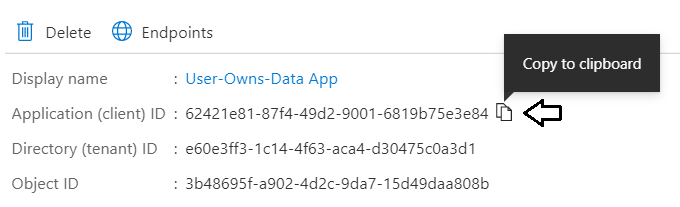
* 1. In the **Redirect URI** section, select **Web** in the left dropdown to create a new confidential client application.
  2. In the textbox to the right of the dropdown menu, enter **https://localhost:44300** as the **Redirect URI**.
  3. Click the **Register** button to create the new Azure AD application.



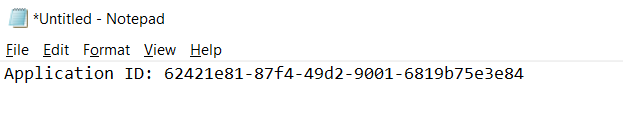
* 1. Once you've created the new application you should see the application summary view as shown in the following screenshot.



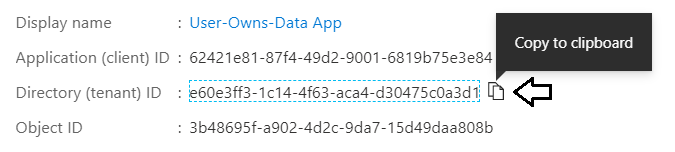
* 1. Copy the **Application ID** to the Windows clipboard.



* 1. Launch Notepad and paste the **Application ID** into a new text file.

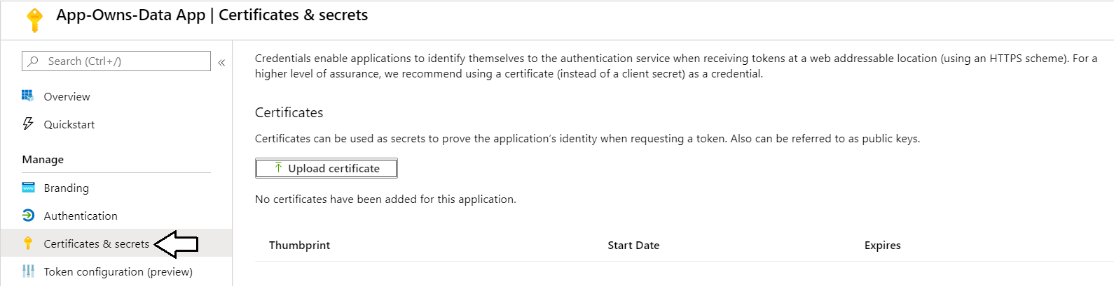


* 1. Return the web page in the Azure portal and copy the tenant ID to the Windows clipboard.

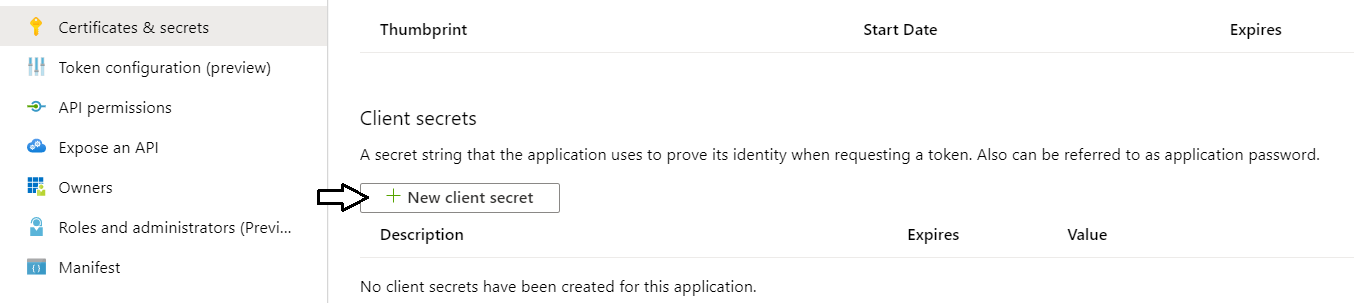


* 1. Copy the tenant ID into the new document you have created in Notepad.exe.

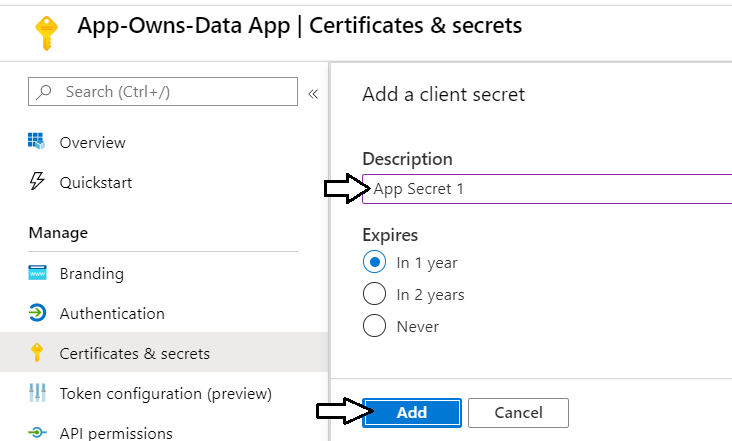
1. Create a new client secret (aka application secret) which will be used for app-only authentication.
   1. Click the **Certificates & secrets** link in the left navigation for **App-Owns-Data App**.



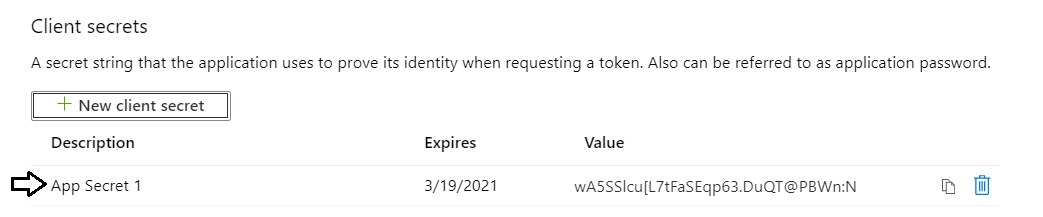
* 1. In the **Client secrets** section, click the **New client secret** button.



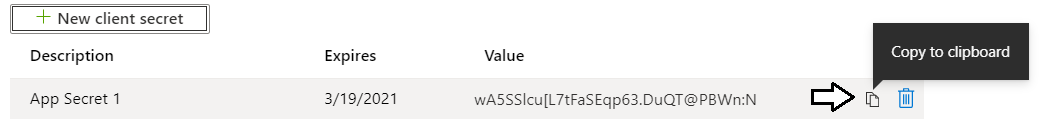
* 1. In the **Add a client secret** pane, enter a **Description** of **App Secret 1** and click the **Add** button.



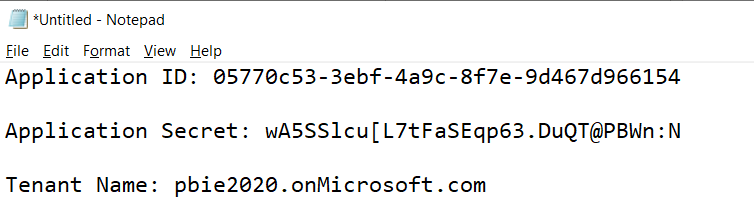
* 1. You should be able to confirm that **App Secret 1** now appears in the **Client secrets** list.



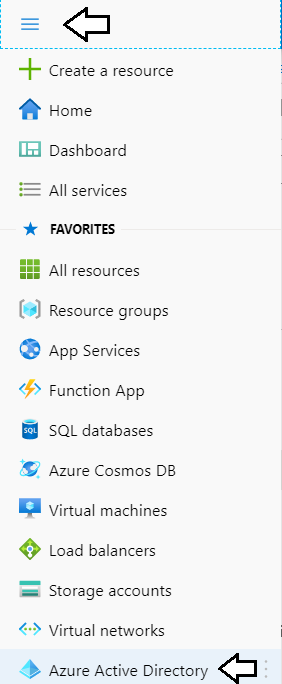
* 1. Click on the **Copy to clipboard** button to copy the new client secret to the Windows clipboard.



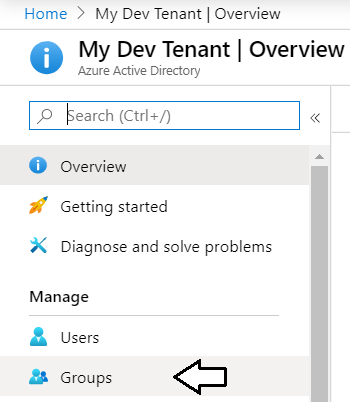
* 1. Paste the client ID into the new document in Notepad and name it Application ID.
  2. Add the Tenant Name to the Notepad document as shown in the following screenshot.



1. Add the Azure AD application named **App-Owns-Data App** to the **Power BI Apps** security group.
   1. Use the drop down navigation menu in the Azure portal to navigate to the **Azure Active Directory** root page.

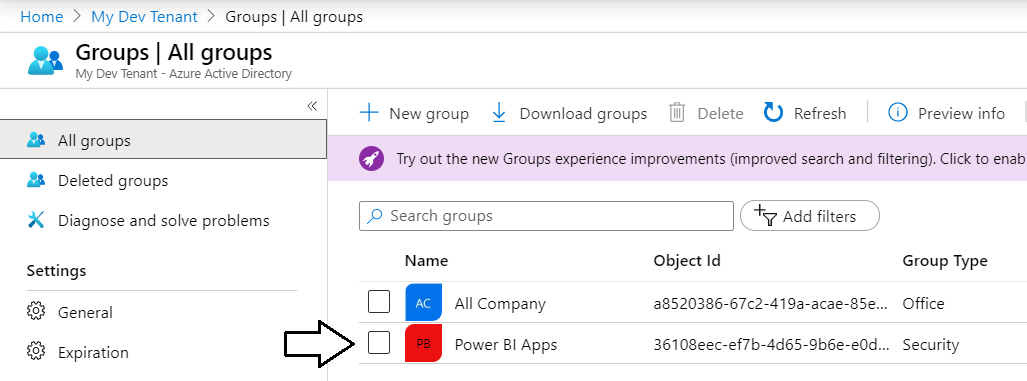


* 1. Select the **Groups** link in the left navigation of the Azure Active Directory section of the Azure portal.

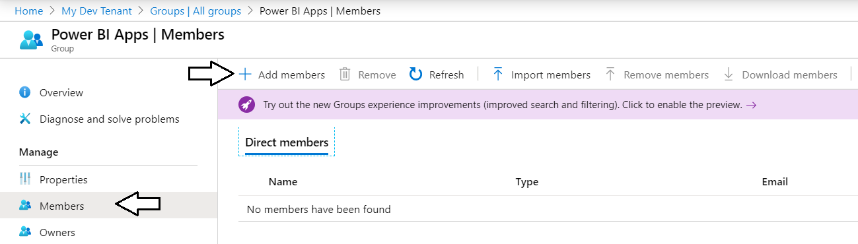


You should be able to see the security group you created earlier named **Power BI Apps**.

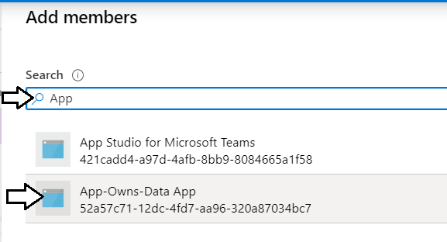
* 1. xxxx



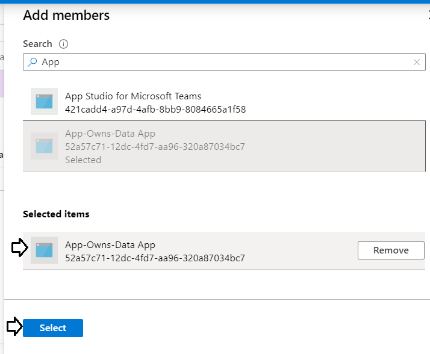
* 1. ssss



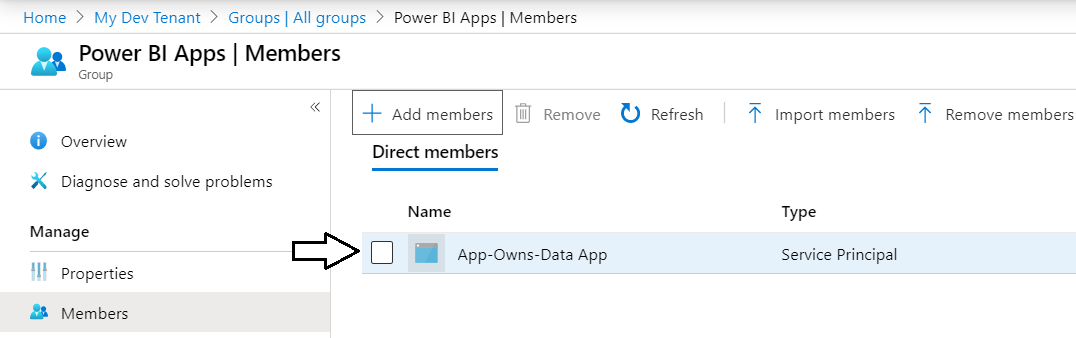
* 1. ssss



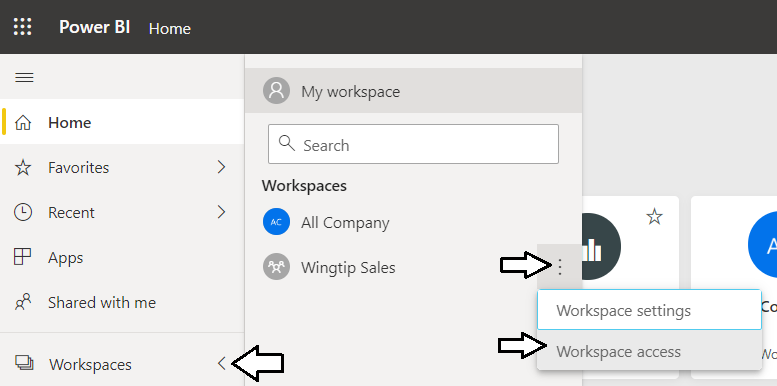
* 1. ssssss



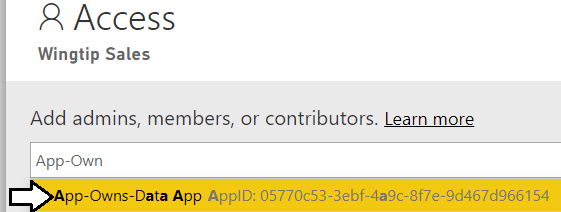
* 1. sssss



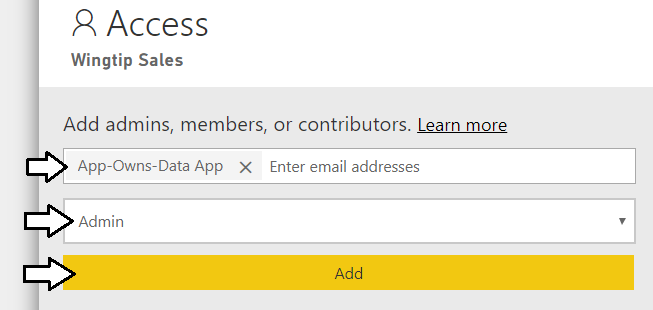
1. Add the service principal for App-Owns-Data App as an admin for the Wingtip Sales app workspace.
   1. Navigate to the Power BI portal.
   2. Expand the **Workspaces** flyout menu.
   3. Click the **Wingtip Sales** workspace context menu (**…**) and select **Workspace access**.



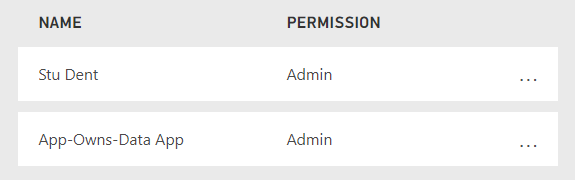
* 1. On the right of the page, you should see the **Access** pane for the **Wingtip Sales** workspace.
  2. Place the cursor into the *Enter email address* textbox and type **App-Owns-Data App**.



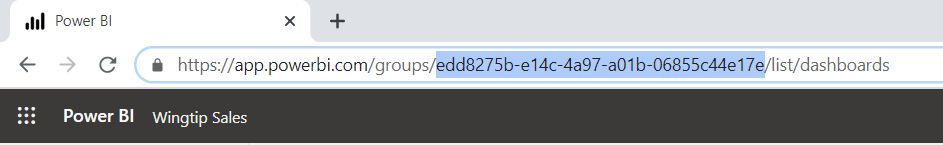
* 1. Change the member type from **Member** to **Admin**.
  2. Click to **Add** button.



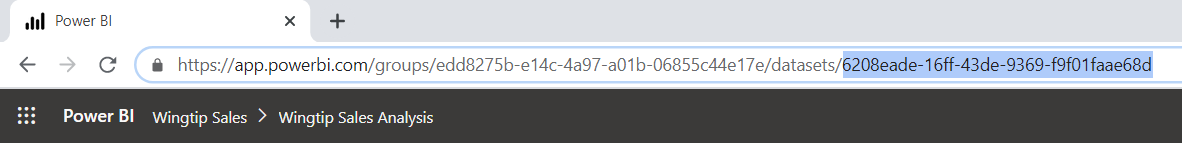
* 1. Verify that **App-Owns-Data App** has been added as a workspace member with **Admin** permissions.



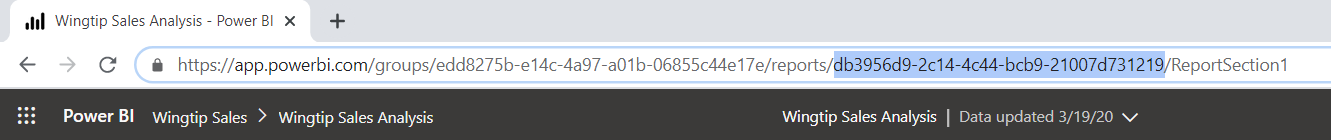
1. Gather the configuration data you will need for your Power BI embedding application.
2. Retrieve the GUID-based IDs for the **Wingtip Sales** app workspace and the embeddable resources inside.
   1. Navigate to the **Wingtip Sales** app workspace in the Power BI portal.
   2. Locate and copy the app workspace ID from the URL by copying the GUID that comes after **/groups/**.



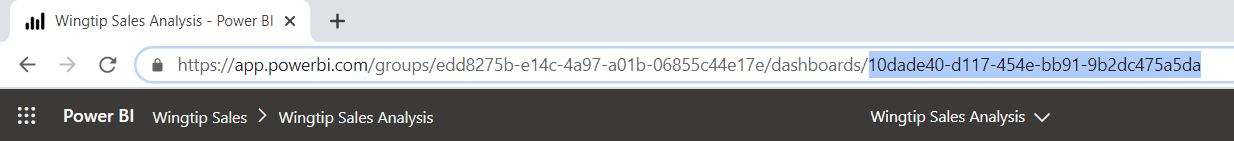
* 1. Copy the app workspace ID into the text file Notepad.
  2. Navigate to the **Wingtip Sales Analysis** dataset inside the **Wingtip Sales** app workspace to create a new report.
  3. Locate and copy the dataset ID from the URL by copying the GUID that comes after **/datasets/**.



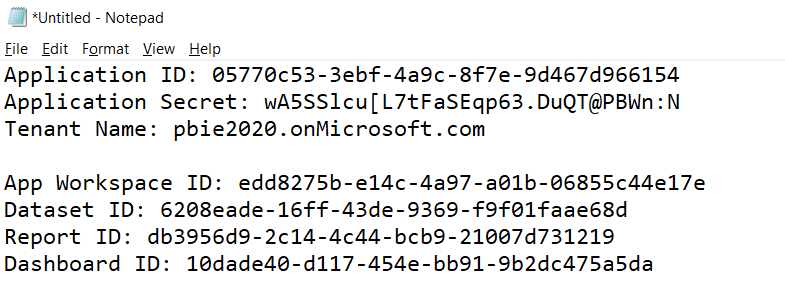
* 1. Copy the dataset ID into the text file Notepad.
  2. Navigate back to the **Wingtip Sales Analysis** report inside the **Wingtip Sales** app workspace.
  3. Locate and copy the report ID from the URL by copying the GUID that comes after **/reports/**.



* 1. Copy the report ID into the text file Notepad.
  2. Navigate to the **Wingtip Sales Analysis** dashboard.
  3. Locate and copy the dashboard ID from the URL by copying the GUID that comes after **/dashboards/**.



* 1. Copy the dashboard ID into the text file Notepad.
  2. You should have now updated the text file Notepad with all the configuration data you need.

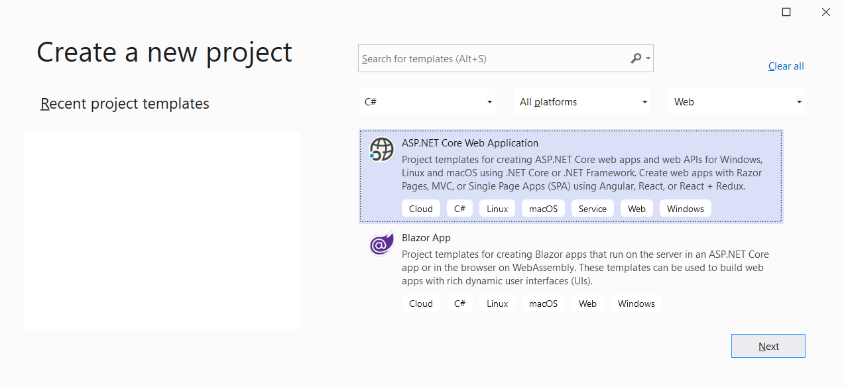


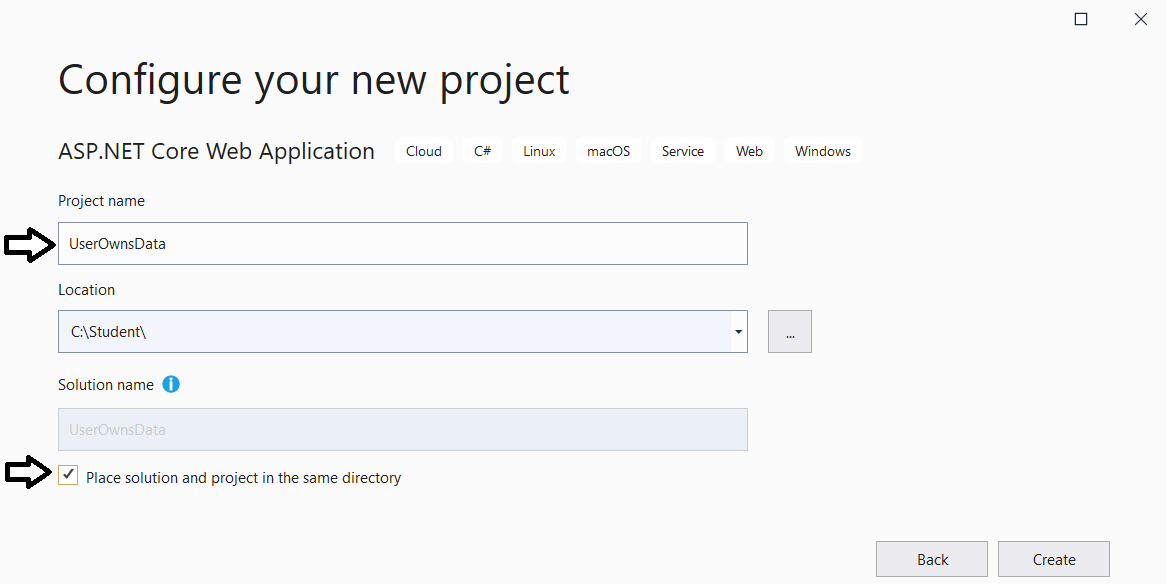
* 1. Save your changes to **AppOwnsDataApp.txt**.
  2. Leave **AppOwnsDataApp.txt** open as you will need it when developing a new application in the next exercise.

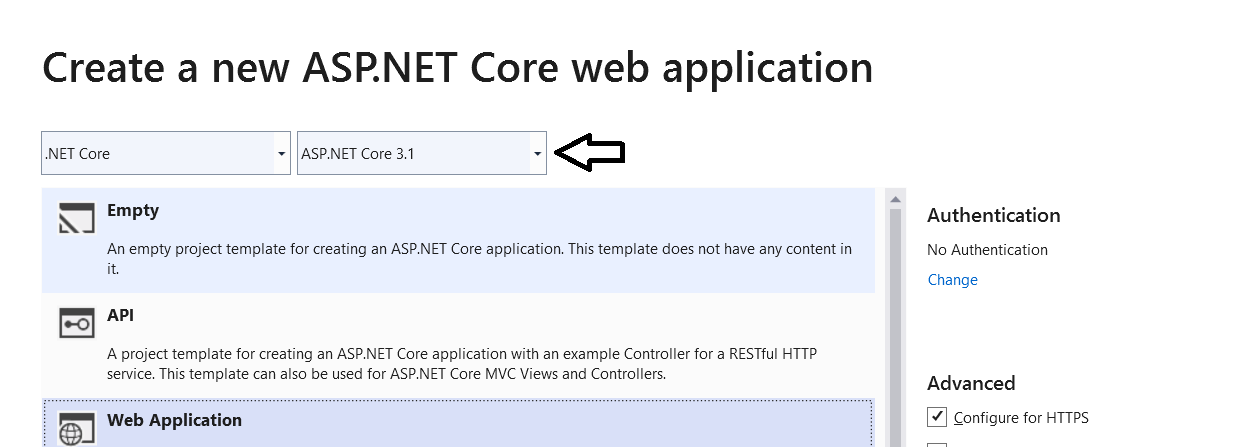
### Exercise 2: Create a new .NET Core Web Application in Visual Studio 2019

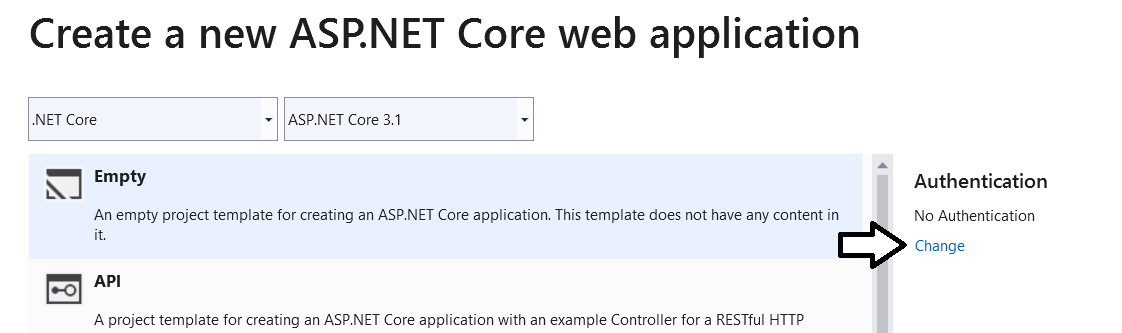
In this exercise you will create a new Web Application project using Visual Studio 2019 and the ASP.NET MVC framework.

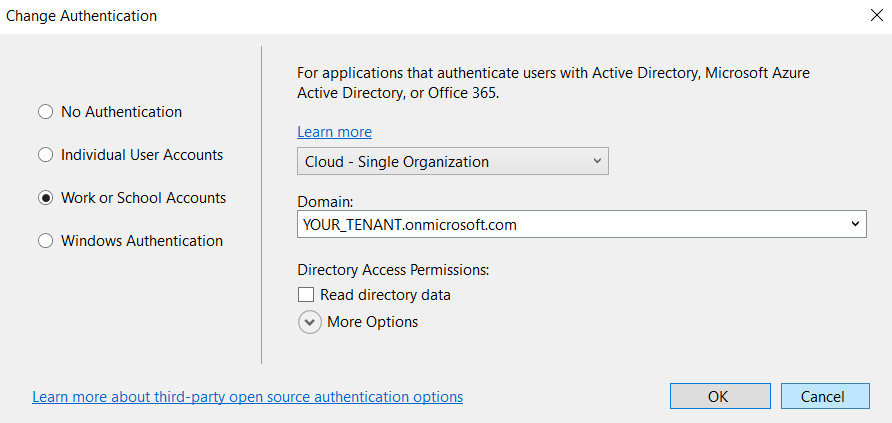
1. Launch **Visual Studio 2019**.
2. Create a new ASP.NET MVC project in Visual Studio 2019.
   1. In Visual Studio select **File > New > Project**.
   2. In the **New Project** dialog:
      1. Select **Installed > Templates > Visual C# > Web**.
      2. Select the **ASP.NET Web Application** project template.
      3. Name the new project **AppOwnsDataApp**.
      4. Add the new project into the folder at **C:\Student\Modules\05\_PowerBiEmbedding\Lab**.
      5. Click **OK** to display the **New ASP.Net Web Application** wizard.

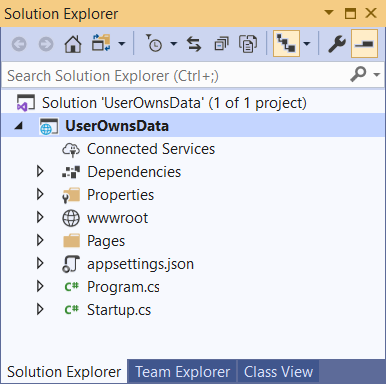


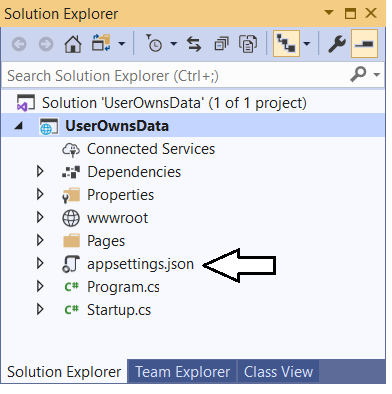












{

"AzureAd": {

"Instance": "https://login.microsoftonline.com/",

"Domain": "MSTDEV2020.onmicrosoft.com",

"TenantId": "a89a5a60-0169-470a-8032-de9e5e18cdf0",

"ClientId": "854409c8-b6ed-432d-bd79-68f6848a417b",

"CallbackPath": "/signin-oidc"

},

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft": "Warning",

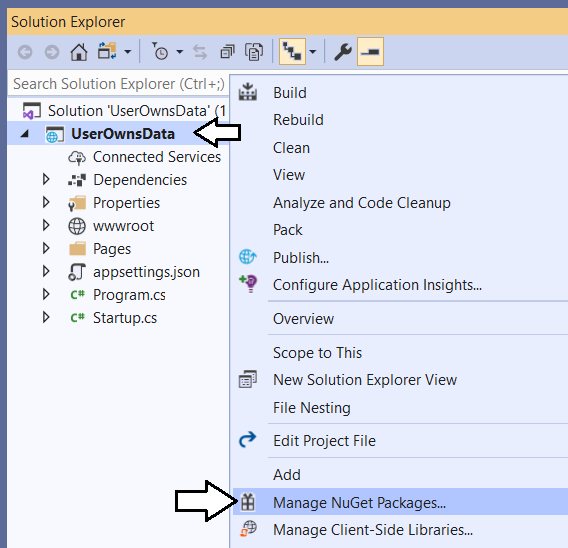
"Microsoft.Hosting.Lifetime": "Information"

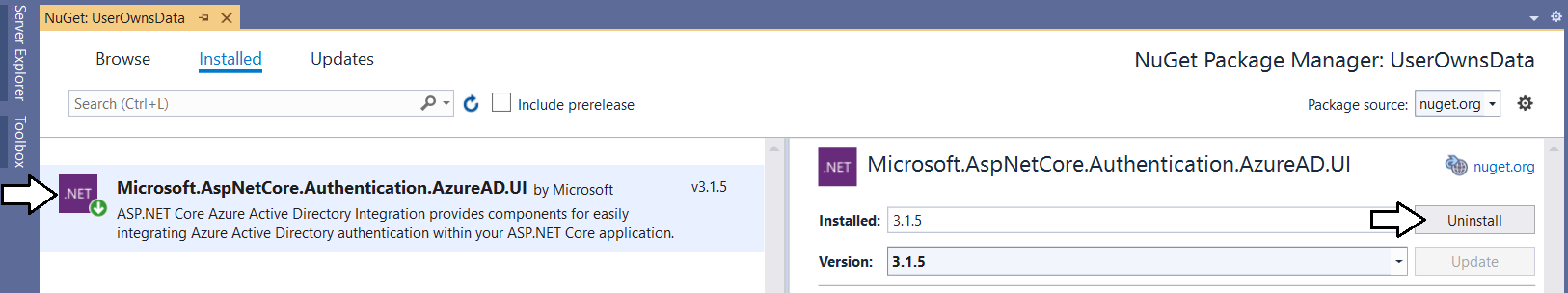
}

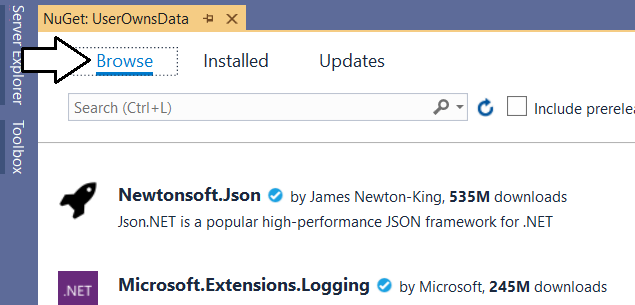
},

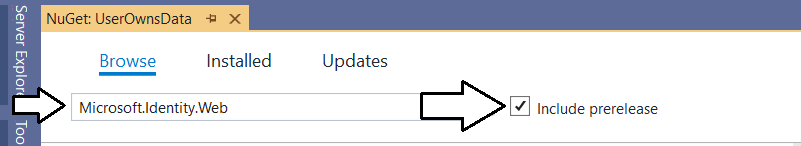
"AllowedHosts": "\*"

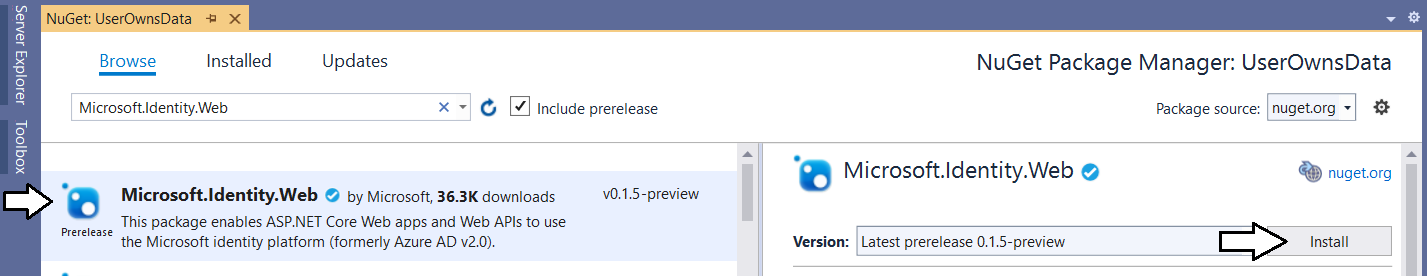
}

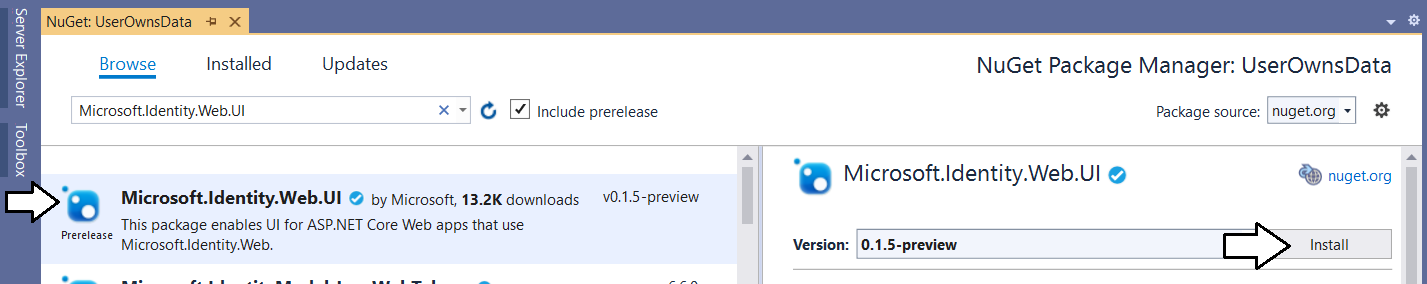


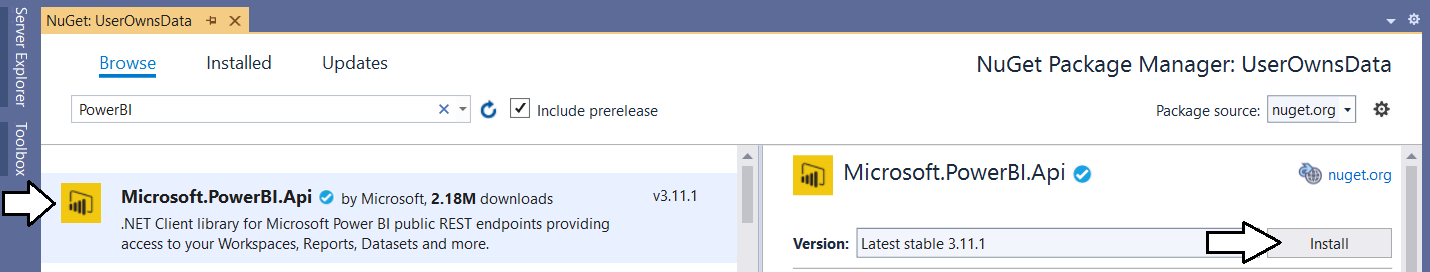


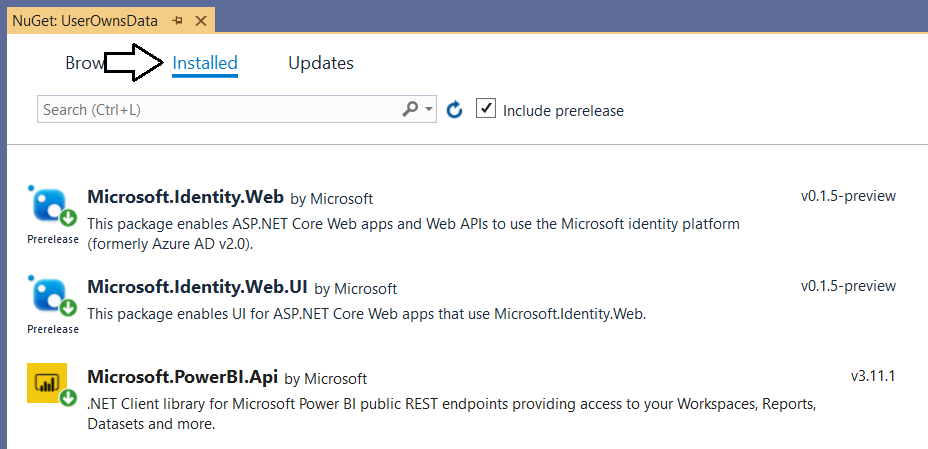


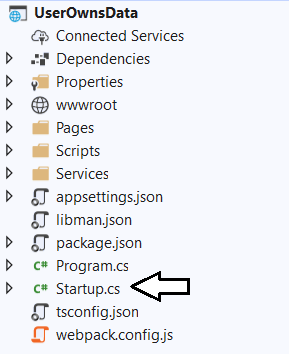












public void ConfigureServices(IServiceCollection services) {

services.AddAuthentication(AzureADDefaults.AuthenticationScheme)

.AddAzureAD(options => Configuration.Bind("AzureAd", options));

services.AddRazorPages().AddMvcOptions(options => {

var policy = new AuthorizationPolicyBuilder()

.RequireAuthenticatedUser()

.Build();

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

});

}

public void ConfigureServices(IServiceCollection services) {

//services.AddAuthentication(AzureADDefaults.AuthenticationScheme)

// .AddAzureAD(options => Configuration.Bind("AzureAd", options));

services.AddSignIn(Configuration);

services.AddRazorPages().AddMvcOptions(options => {

var policy = new AuthorizationPolicyBuilder()

.RequireAuthenticatedUser()

.Build();

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

});

}

public void ConfigureServices(IServiceCollection services) {

services.AddSignIn(Configuration);

services.AddControllersWithViews(options => {

var policy = new AuthorizationPolicyBuilder()

.RequireAuthenticatedUser()

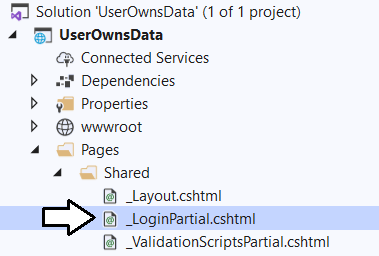
.Build();

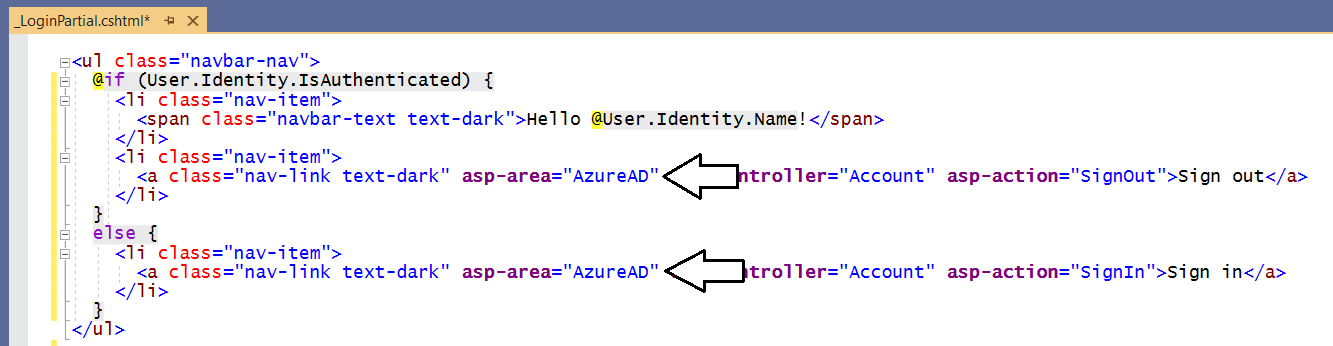
options.Filters.Add(new AuthorizeFilter(policy));

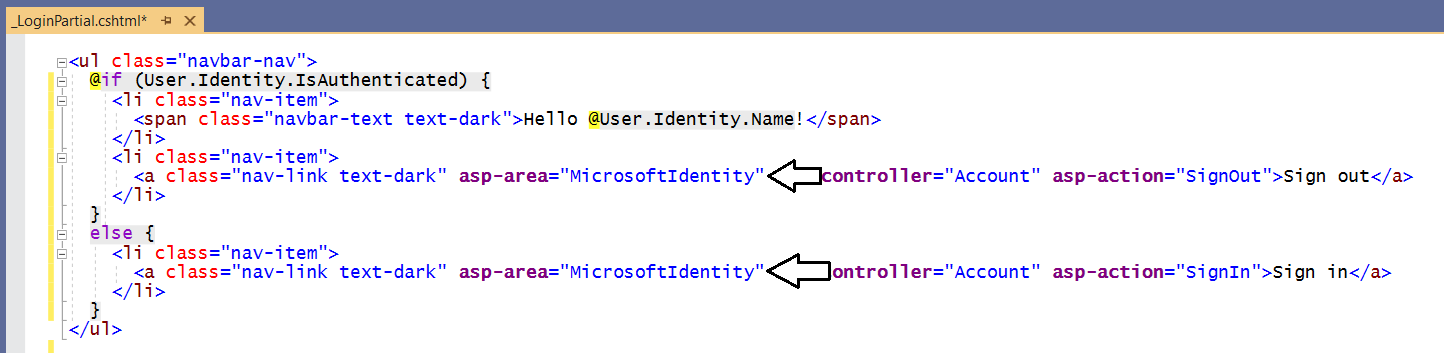
}).AddMicrosoftIdentityUI();

services.AddRazorPages();

}

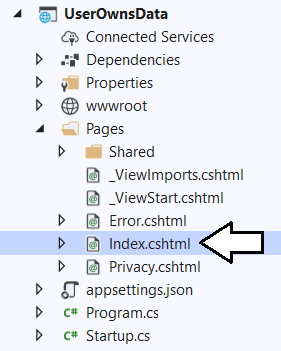


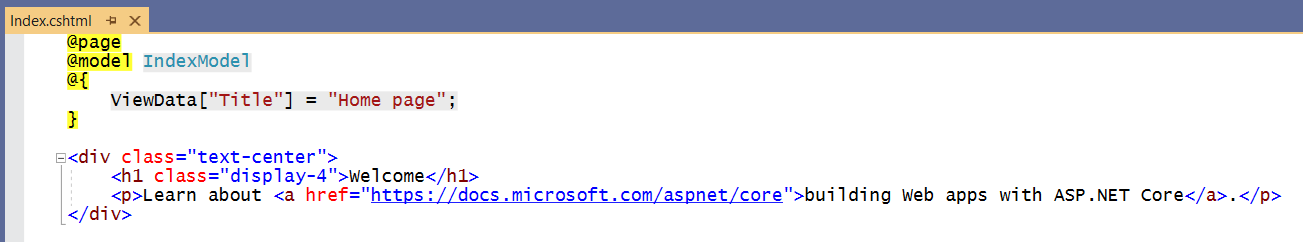




<span class="navbar-text text-dark">Hello @User.FindFirst("name").Value</span>







@page

@model IndexModel

@if (User.Identity.IsAuthenticated) {

string userName = @User.FindFirst("name").Value;

<div class="jumbotron">

<h1>Welcome @userName</h1>

<p class="lead">You are now logged into Azure AD as an authenticated user.</p>

</div>

}

else {

<div class="jumbotron">

<h1>Please login to get started</h1>

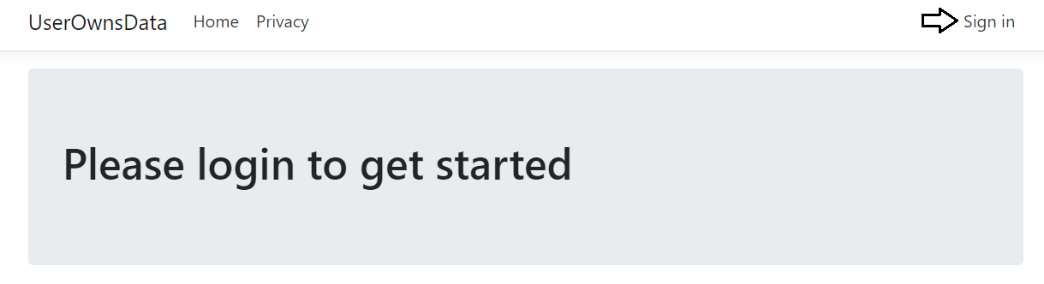
</div>

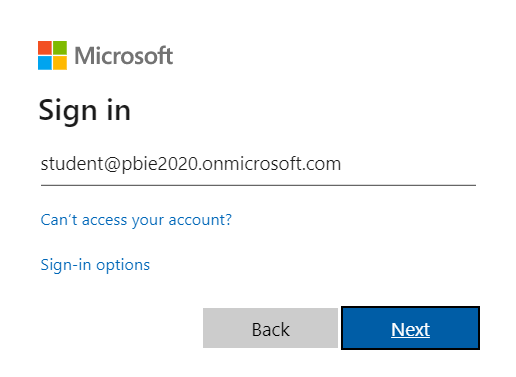
}

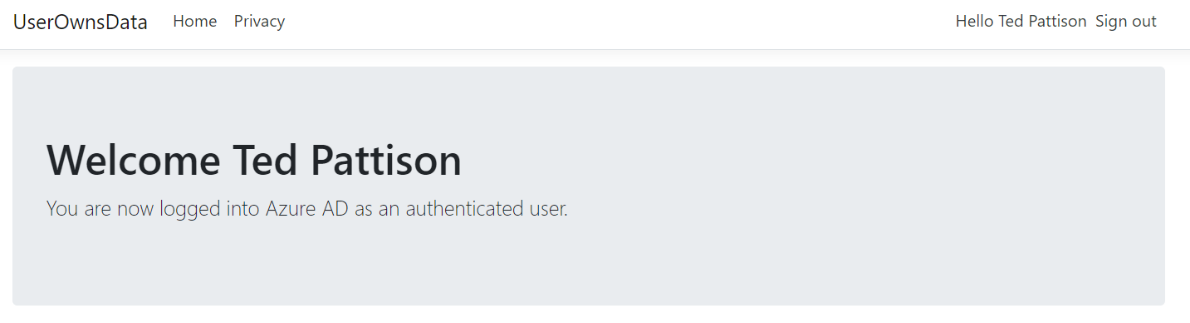
namespace UserOwnsData.Pages {

[AllowAnonymous]

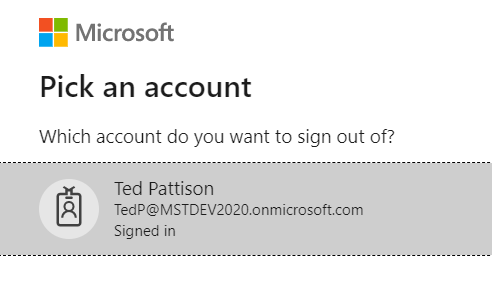
public class IndexModel : PageModel {





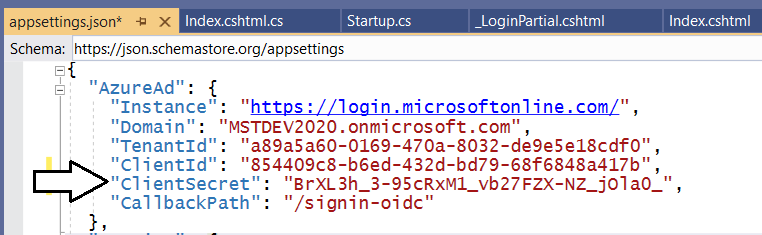


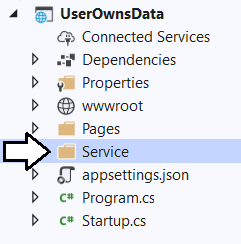


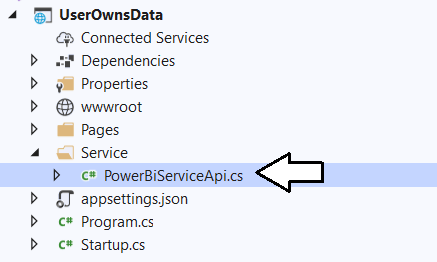


### Exercise 3: Call the Power BI Service API

In this exercise, you will.







using System;

using System.Threading.Tasks;

using Microsoft.Identity.Web;

using Microsoft.Rest;

using Microsoft.PowerBI.Api;

namespace UserOwnsData.Services {

public class EmbeddedReportViewModel {

public string Id;

public string Name;

public string EmbedUrl;

public string Token;

}

public class PowerBiServiceApi {

readonly ITokenAcquisition tokenAcquisition;

public PowerBiServiceApi(ITokenAcquisition tokenAcquisition) {

this.tokenAcquisition = tokenAcquisition;

}

const string urlPowerBiServiceApiRoot = "https://api.powerbi.com/";

public static readonly string[] RequiredScopes =

new string[] {

"https://analysis.windows.net/powerbi/api/Group.Read.All",

"https://analysis.windows.net/powerbi/api/Dashboard.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",

};

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

// get access token

var accessToken = this.tokenAcquisition.GetAccessTokenForUserAsync(RequiredScopes).Result;

// create PBI client to call Power BI Service API

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

PowerBIClient pbiClient = new PowerBIClient(new Uri(urlPowerBiServiceApiRoot), tokenCredentials);

// 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 = accessToken

};

}

}

}

Z

using Microsoft.Identity.Web.TokenCacheProviders.InMemory;

using UserOwnsData.Services;

It should currently look like this.

public void ConfigureServices(IServiceCollection services) {

services.AddSignIn(Configuration);

services.AddControllersWithViews(options => {

var policy = new AuthorizationPolicyBuilder()

.RequireAuthenticatedUser()

.Build();

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

}).AddMicrosoftIdentityUI();

services.AddRazorPages();

}

X

public void ConfigureServices(IServiceCollection services) {

services.AddSignIn(Configuration);

services

.AddWebAppCallsProtectedWebApi(Configuration, PowerBiServiceApi.RequiredScopes)

.AddInMemoryTokenCaches();

services.AddScoped(typeof(PowerBiServiceApi));

services.AddControllersWithViews(options => {

var policy = new AuthorizationPolicyBuilder()

.RequireAuthenticatedUser()

.Build();

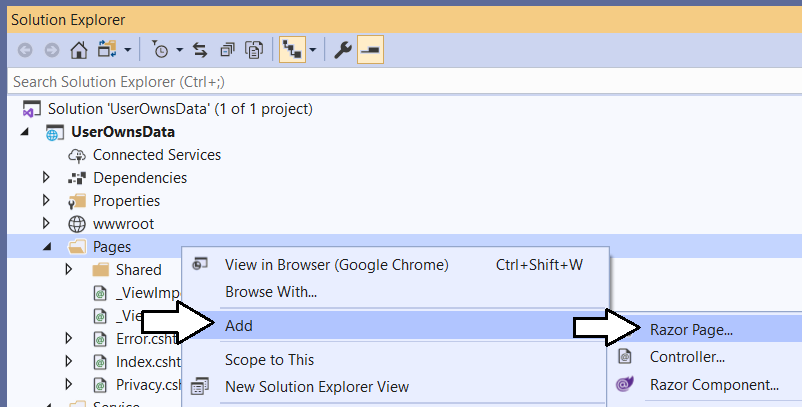
options.Filters.Add(new AuthorizeFilter(policy));

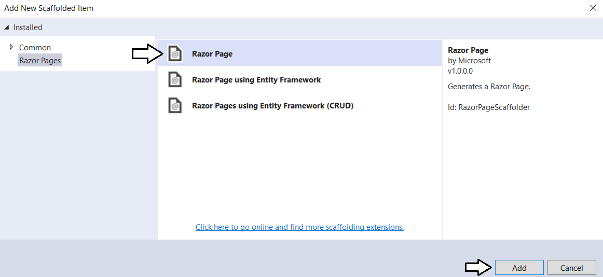
}).AddMicrosoftIdentityUI();

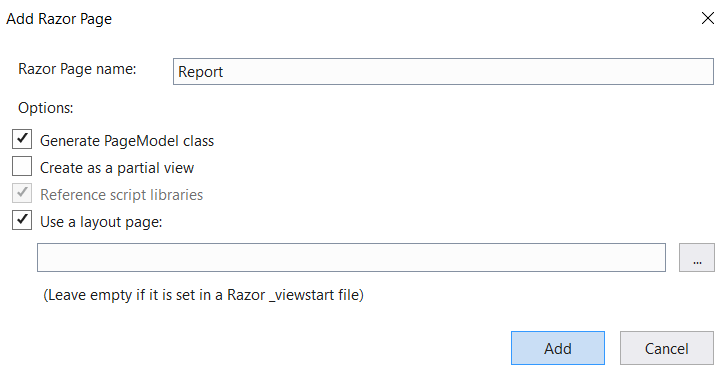
services.AddRazorPages();

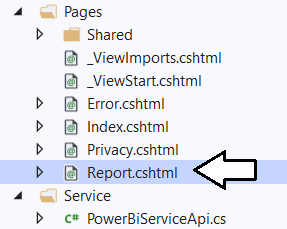
}

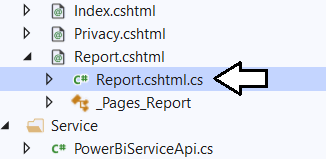
x











This is what it looks like at first

namespace UserOwnsData.Pages {

public class ReportModel : PageModel {

public void OnGet() {

}

}

}

Add this up top

using UserOwnsData.Services;

ssss

public class ReportModel : PageModel {

private PowerBiServiceApi powerBiServiceApi;

public ReportModel(PowerBiServiceApi powerBiServiceApi) {

this.powerBiServiceApi = powerBiServiceApi;

}

public EmbeddedReportViewModel report;

public void OnGet() {

}

}

switch from this

public void OnGet() {

to this

public async Task OnGet(PowerBiServiceApi powerBiServiceApi) {

Get workspace ID and report ID from earlier.

Guid workspaceId = new Guid("609c2d1b-c136-4bec-9a7b-e6829c50108b");

Guid reportId = new Guid("49398818-d952-4b52-a96b-226fec6b9911");

Xx

this.report = await powerBiServiceApi.GetReport(workspaceId, reportId);

x

public class ReportModel : PageModel {

private PowerBiServiceApi powerBiServiceApi;

public ReportModel(PowerBiServiceApi powerBiServiceApi) {

this.powerBiServiceApi = powerBiServiceApi;

}

public EmbeddedReportViewModel report;

public async Task OnGet() {

Guid workspaceId = new Guid("609c2d1b-c136-4bec-9a7b-e6829c50108b");

Guid reportId = new Guid("49398818-d952-4b52-a96b-226fec6b9911");

this.report = await powerBiServiceApi.GetReport(workspaceId, reportId);

}

}

X

@page

@model UserOwnsData.Pages.ReportModel

<h1>Embeded Report</h1>

<table class="table table-bordered">

<tr><td>Name</td><td>@Model.report.Name</td></tr>

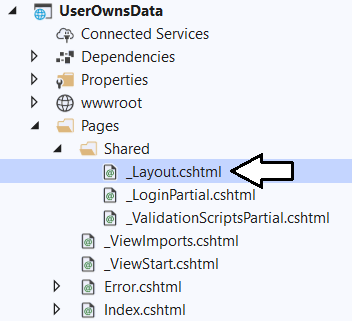
<tr><td>ID</td><td>@Model.report.Id</td></tr>

<tr><td>Embed Url</td><td style="word-break:break-all">@Model.report.EmbedUrl</td></tr>

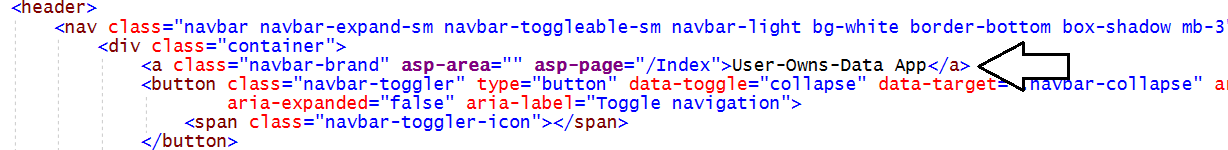
<tr><td>Token</td><td style="word-break:break-all">@Model.report.Token</td></tr>

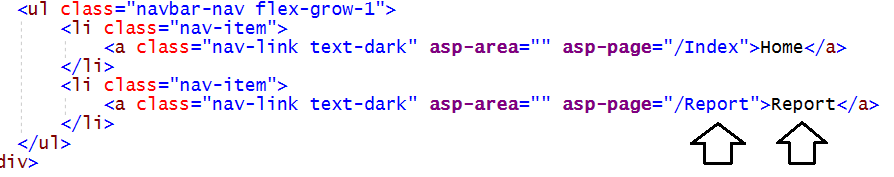
</table>

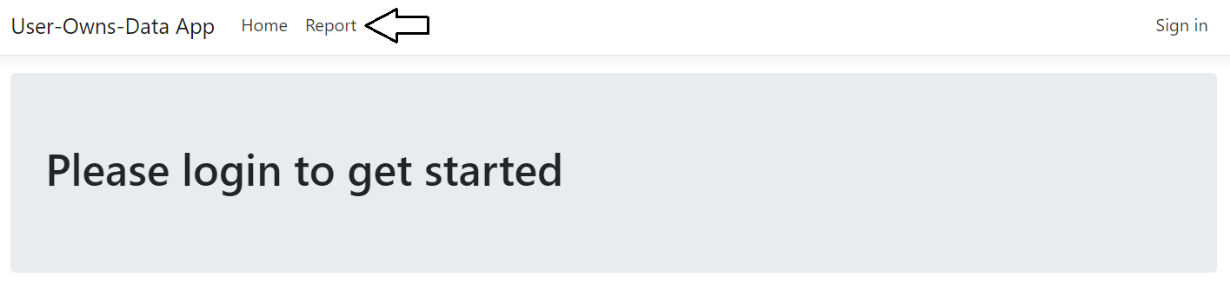
x

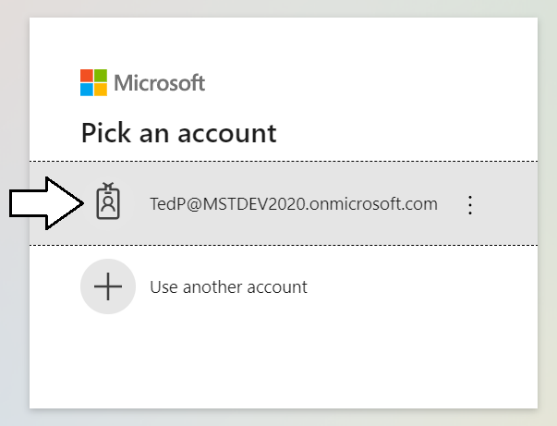


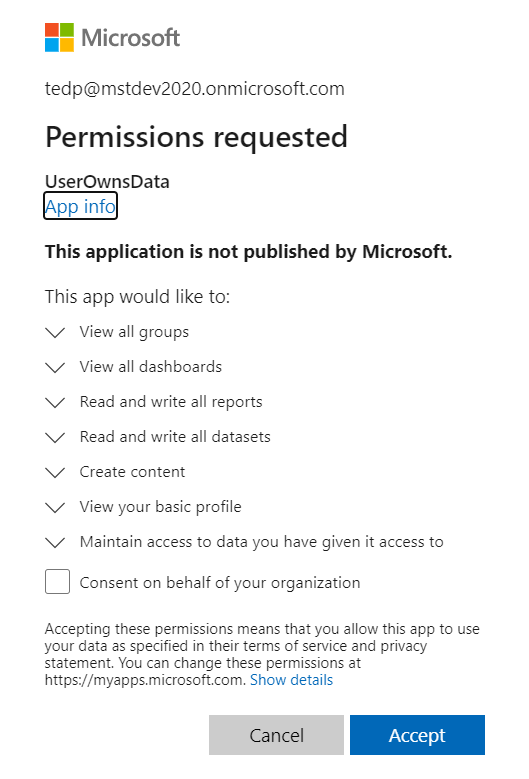


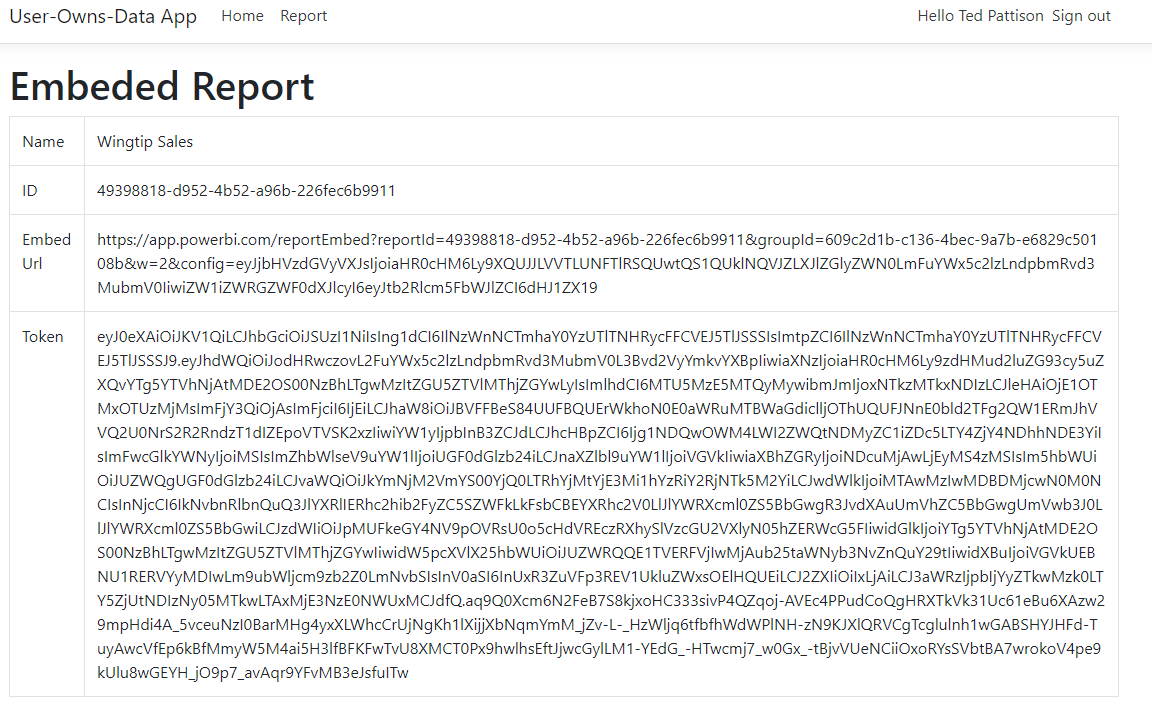






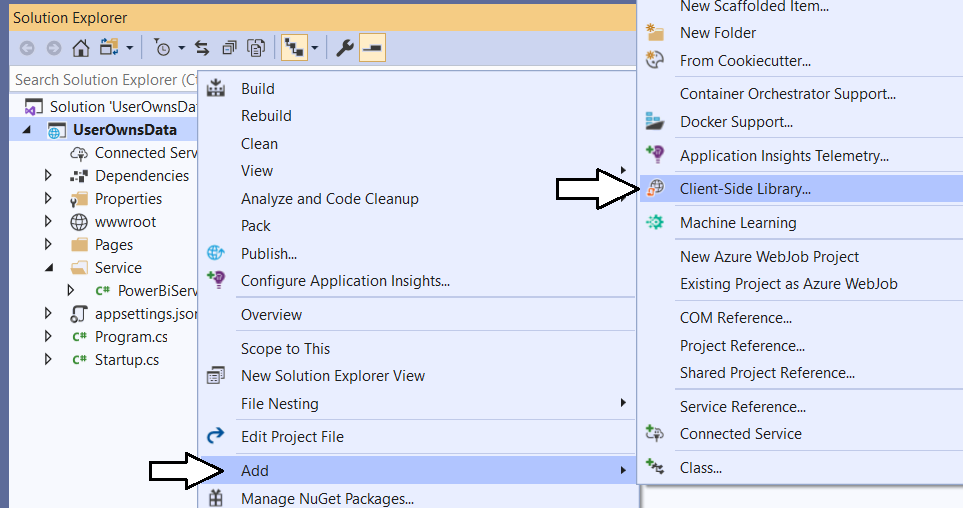


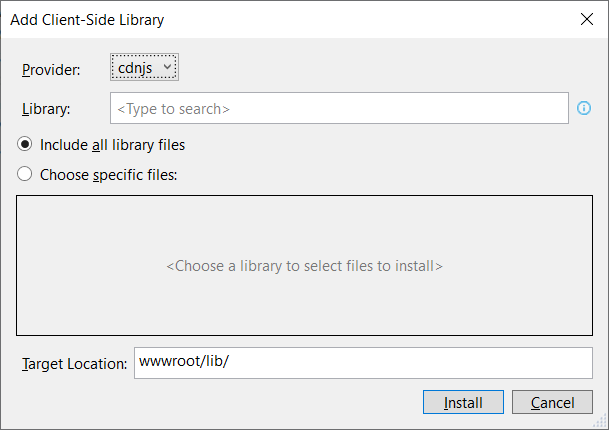


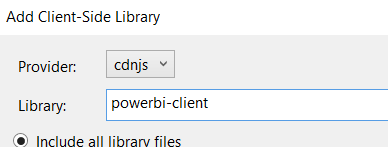


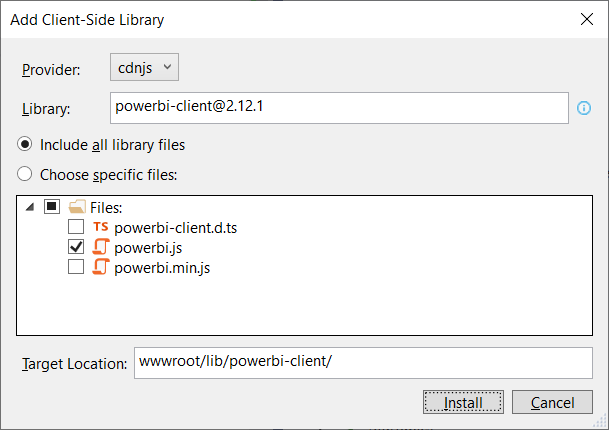
### Exercise 4: Embed a Report with the Power BI JavaScript API

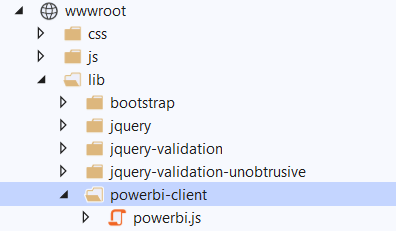
In this exercise, you will.

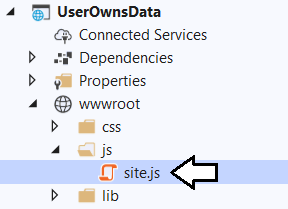


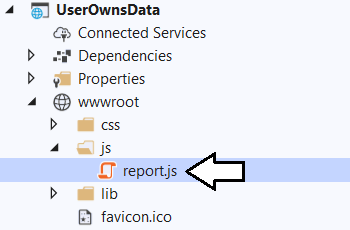












function embedReport(reportContainer, reportId, embedUrl, token) {

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: {

filterPaneEnabled: false,

navContentPaneEnabled: true,

}

};

// Embed the report and display it within the div container.

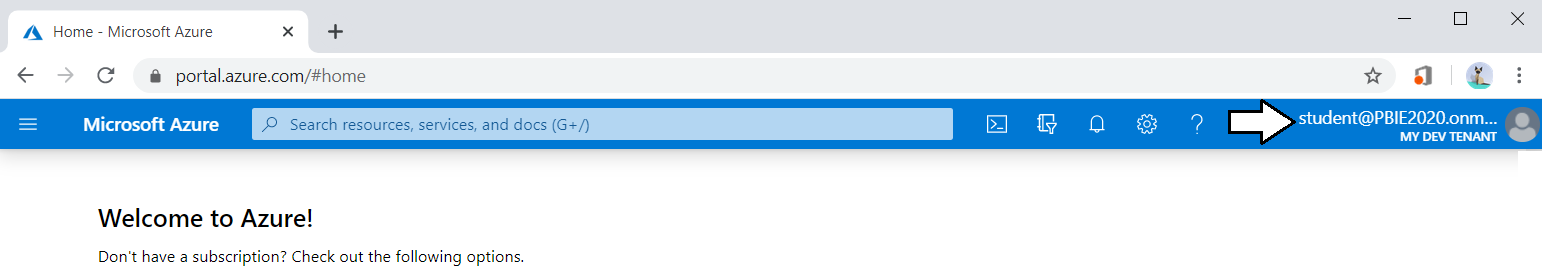
var report = powerbi.embed(reportContainer, config);

}

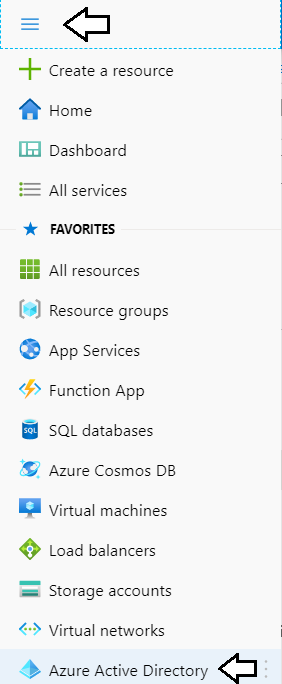
### Exercise 1: Register a New Azure AD Application in the Azure Portal

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.

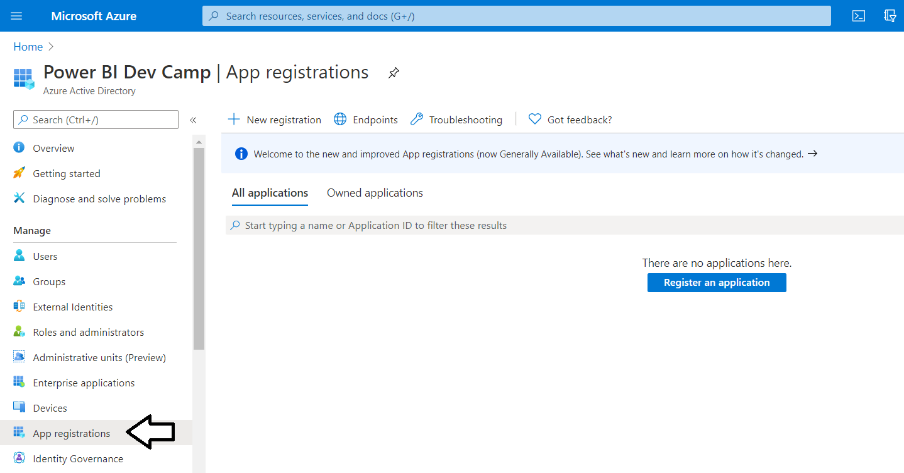
1. Download the lab files to a local drive.
2. Open the script named XYZ.ps1.
3. Log into the Azure Portal
   1. In the browser, navigate to the Azure portal at <https://portal.azure.com>.
   2. When you are prompted to log in, provide the credentials to log in with your Office 365 user account name.
   3. Once you have logged into the Azure portal, check the email address in the login menu in the upper right to make sure you are logged in with the correct identity for your new Office 365 user account.



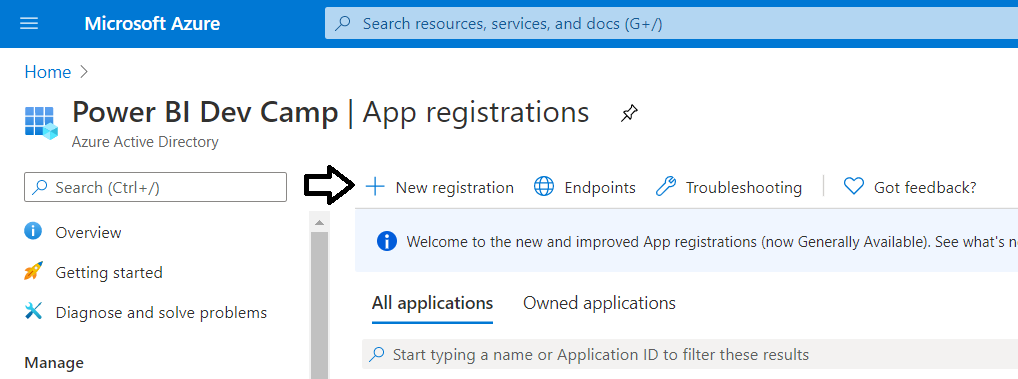
1. Register a new Azure AD application.
   1. Click hamburger icon in the top left corner of the page to drop down the Azure portal navigation menu.
   2. In the left navigation drop down, click the link for **Azure Active Directory**.



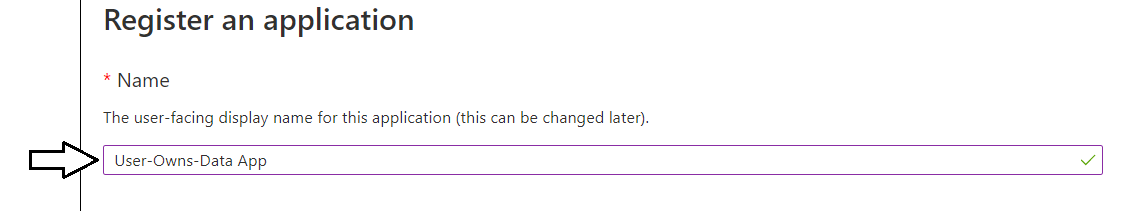
* 1. Once you’ve navigated to **Azure Active Directory** in the Azure portal, click **App registrations** in the left navigation.



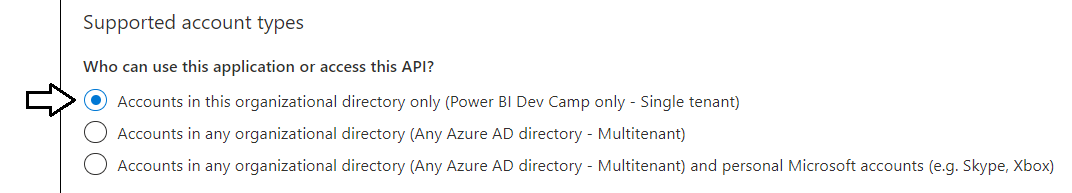
* 1. Click **New registration**.



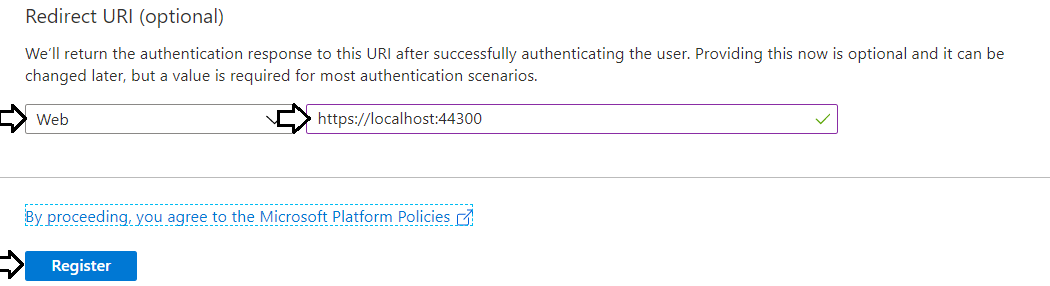
* 1. Enter a **Name** of **App-Owns-Data App**.



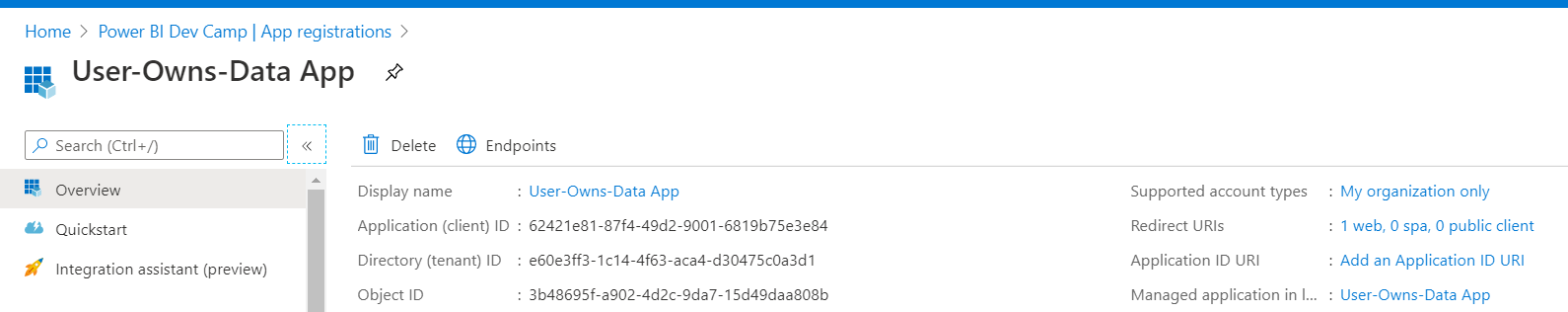
* 1. For the **Supported account types** option, leave the default value of **Accounts in this organizational directory only**.



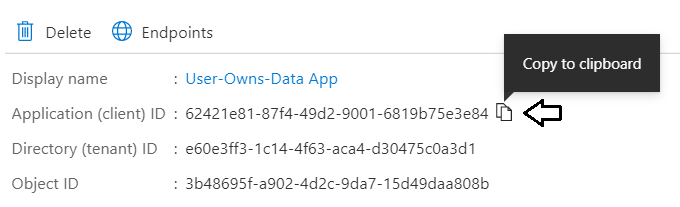
* 1. In the **Redirect URI** section, select **Web** in the left dropdown to create a new confidential client application.
  2. In the textbox to the right of the dropdown menu, enter **https://localhost:44300** as the **Redirect URI**.
  3. Click the **Register** button to create the new Azure AD application.



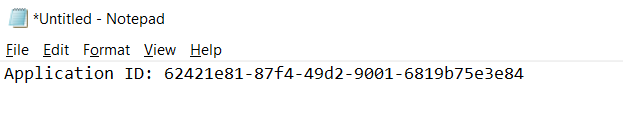
* 1. Once you've created the new application you should see the application summary view as shown in the following screenshot.



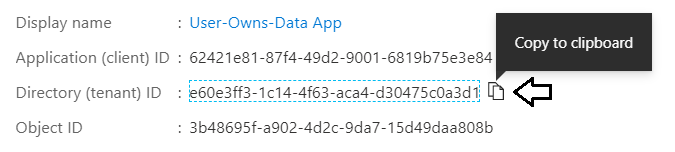
* 1. Copy the **Application ID** to the Windows clipboard.



* 1. Launch Notepad and paste the **Application ID** into a new text file.

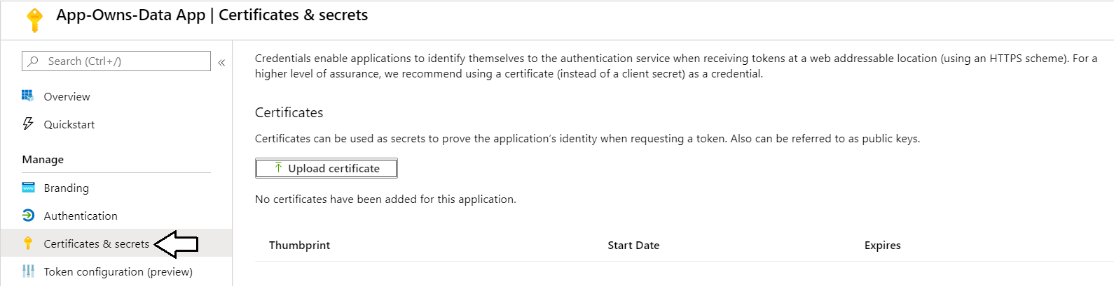


* 1. Return the web page in the Azure portal and copy the tenant ID to the Windows clipboard.

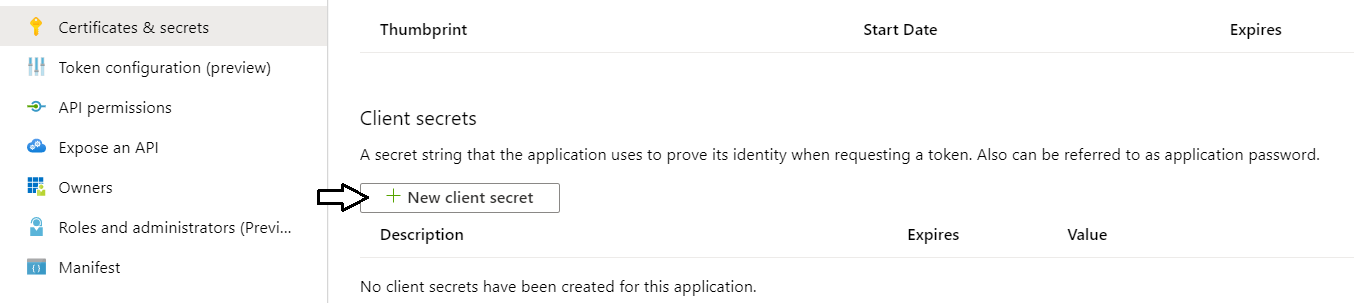


* 1. Copy the tenant ID into the new document you have created in Notepad.exe.

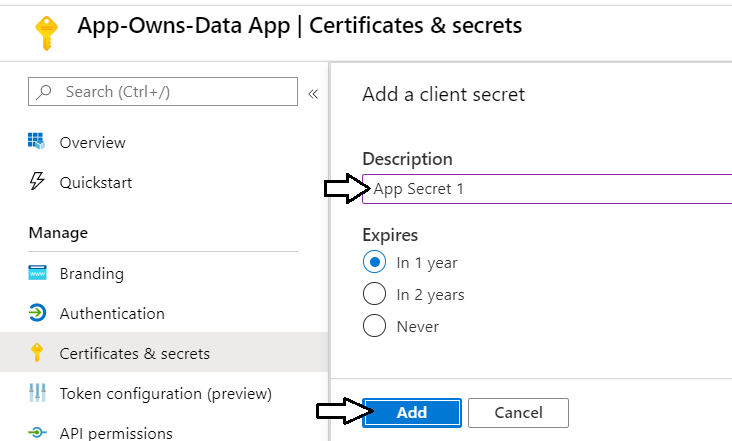
1. Create a new client secret (aka application secret) which will be used for app-only authentication.
   1. Click the **Certificates & secrets** link in the left navigation for **App-Owns-Data App**.



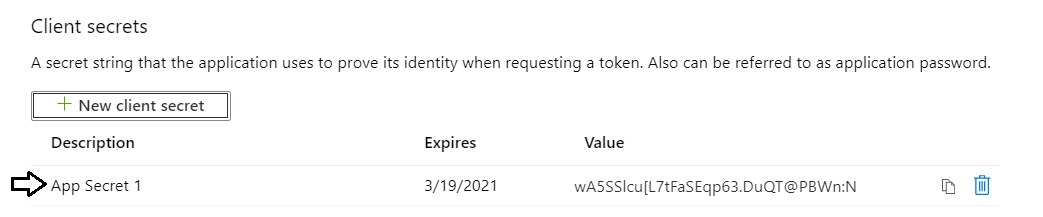
* 1. In the **Client secrets** section, click the **New client secret** button.



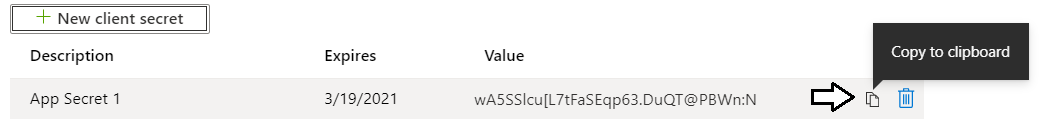
* 1. In the **Add a client secret** pane, enter a **Description** of **App Secret 1** and click the **Add** button.



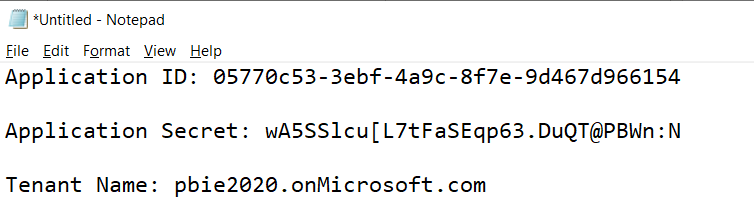
* 1. You should be able to confirm that **App Secret 1** now appears in the **Client secrets** list.



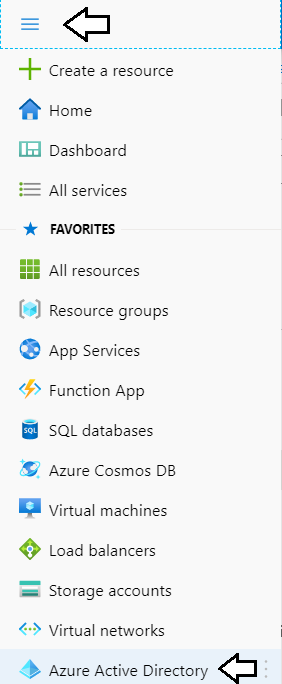
* 1. Click on the **Copy to clipboard** button to copy the new client secret to the Windows clipboard.



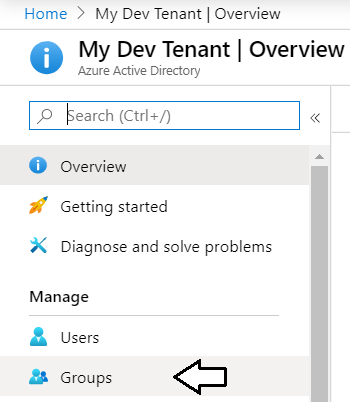
* 1. Paste the client ID into the new document in Notepad and name it Application ID.
  2. Add the Tenant Name to the Notepad document as shown in the following screenshot.



1. Add the Azure AD application named **App-Owns-Data App** to the **Power BI Apps** security group.
   1. Use the drop down navigation menu in the Azure portal to navigate to the **Azure Active Directory** root page.

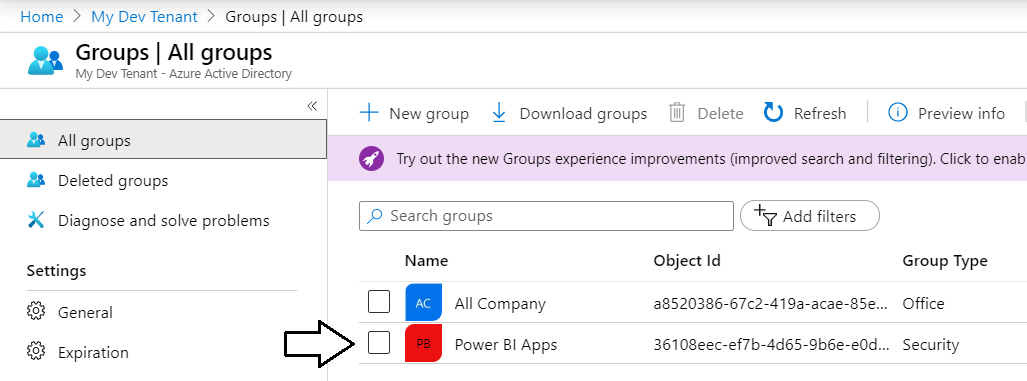


* 1. Select the **Groups** link in the left navigation of the Azure Active Directory section of the Azure portal.

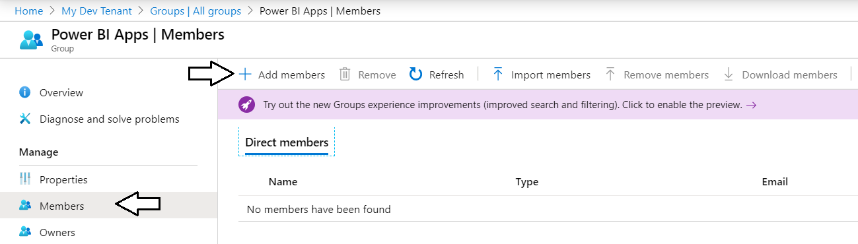


You should be able to see the security group you created earlier named **Power BI Apps**.

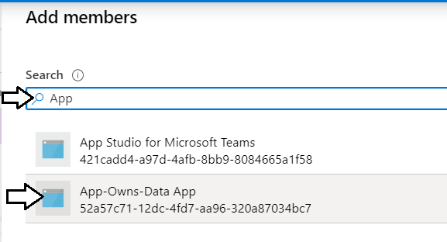
* 1. xxxx



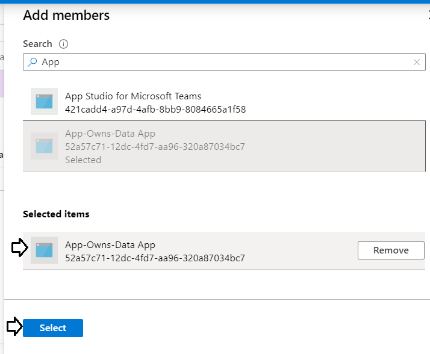
* 1. ssss



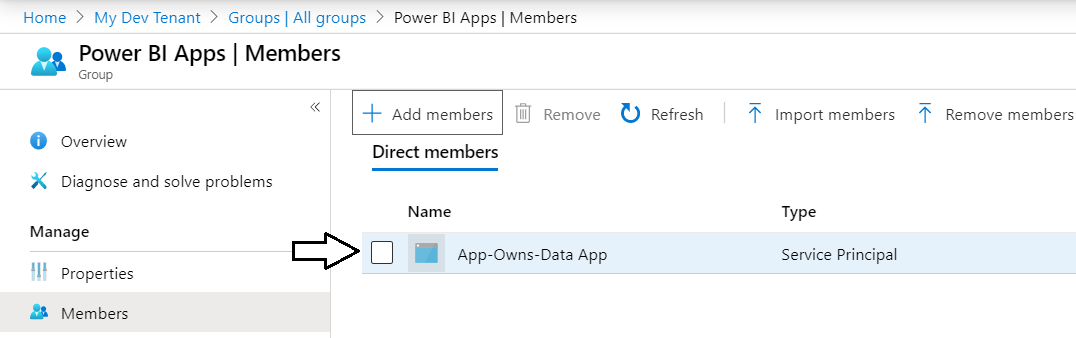
* 1. ssss



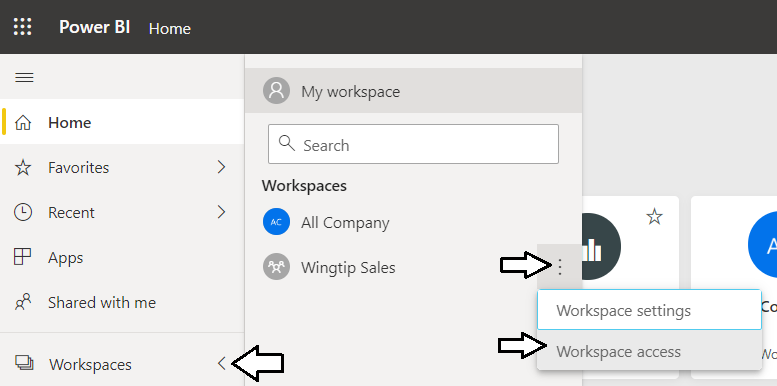
* 1. ssssss



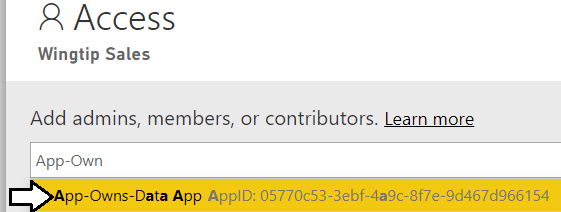
* 1. sssss



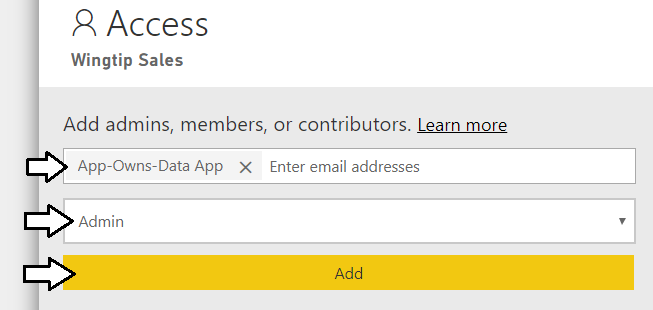
1. Add the service principal for App-Owns-Data App as an admin for the Wingtip Sales app workspace.
   1. Navigate to the Power BI portal.
   2. Expand the **Workspaces** flyout menu.
   3. Click the **Wingtip Sales** workspace context menu (**…**) and select **Workspace access**.



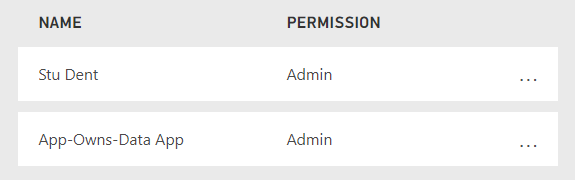
* 1. On the right of the page, you should see the **Access** pane for the **Wingtip Sales** workspace.
  2. Place the cursor into the *Enter email address* textbox and type **App-Owns-Data App**.



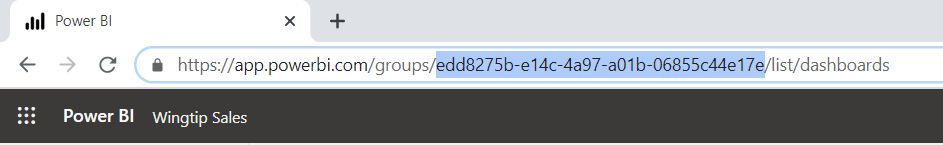
* 1. Change the member type from **Member** to **Admin**.
  2. Click to **Add** button.



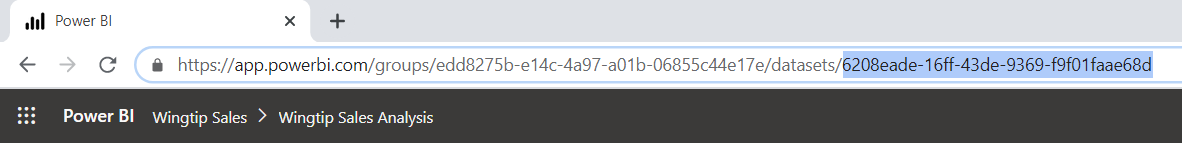
* 1. Verify that **App-Owns-Data App** has been added as a workspace member with **Admin** permissions.



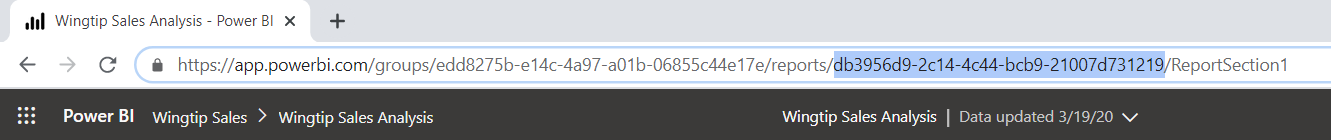
1. Gather the configuration data you will need for your Power BI embedding application.
2. Retrieve the GUID-based IDs for the **Wingtip Sales** app workspace and the embeddable resources inside.
   1. Navigate to the **Wingtip Sales** app workspace in the Power BI portal.
   2. Locate and copy the app workspace ID from the URL by copying the GUID that comes after **/groups/**.



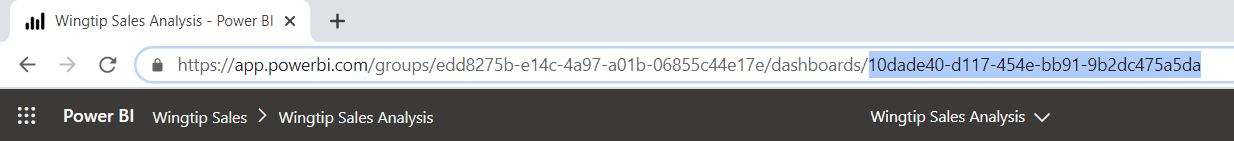
* 1. Copy the app workspace ID into the text file Notepad.
  2. Navigate to the **Wingtip Sales Analysis** dataset inside the **Wingtip Sales** app workspace to create a new report.
  3. Locate and copy the dataset ID from the URL by copying the GUID that comes after **/datasets/**.



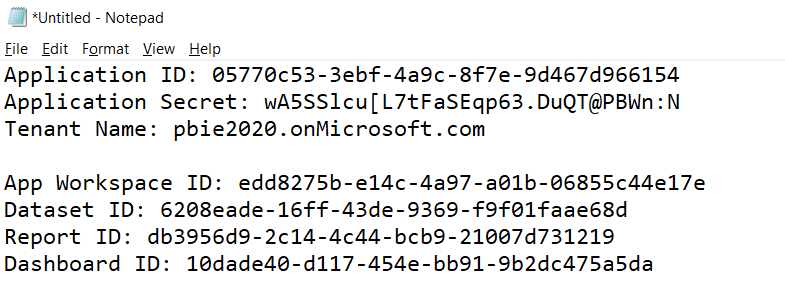
* 1. Copy the dataset ID into the text file Notepad.
  2. Navigate back to the **Wingtip Sales Analysis** report inside the **Wingtip Sales** app workspace.
  3. Locate and copy the report ID from the URL by copying the GUID that comes after **/reports/**.



* 1. Copy the report ID into the text file Notepad.
  2. Navigate to the **Wingtip Sales Analysis** dashboard.
  3. Locate and copy the dashboard ID from the URL by copying the GUID that comes after **/dashboards/**.



* 1. Copy the dashboard ID into the text file Notepad.
  2. You should have now updated the text file Notepad with all the configuration data you need.

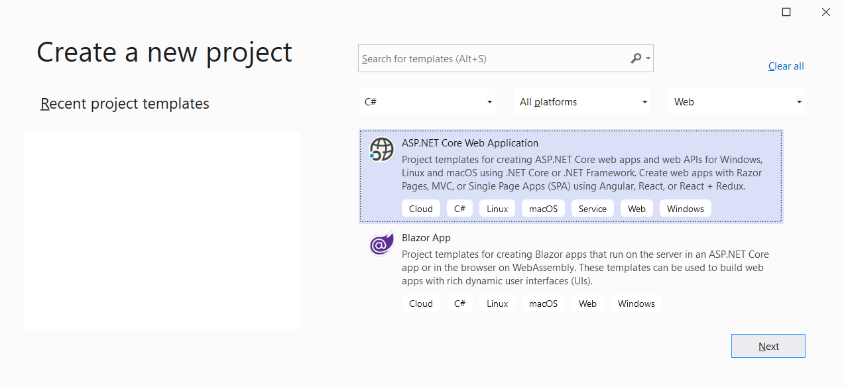


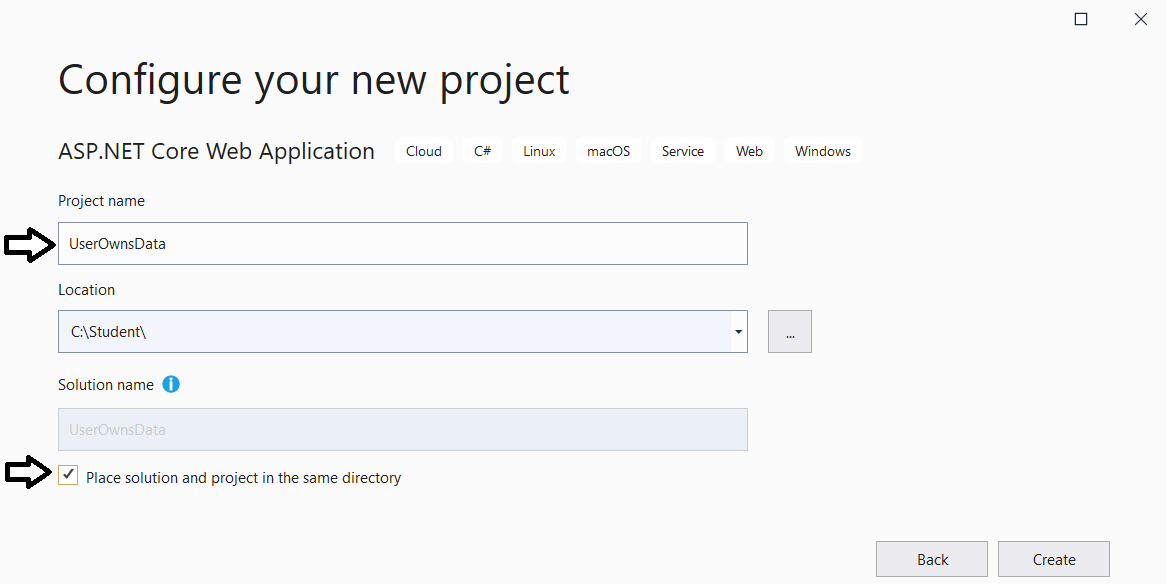
* 1. Save your changes to **AppOwnsDataApp.txt**.
  2. Leave **AppOwnsDataApp.txt** open as you will need it when developing a new application in the next exercise.

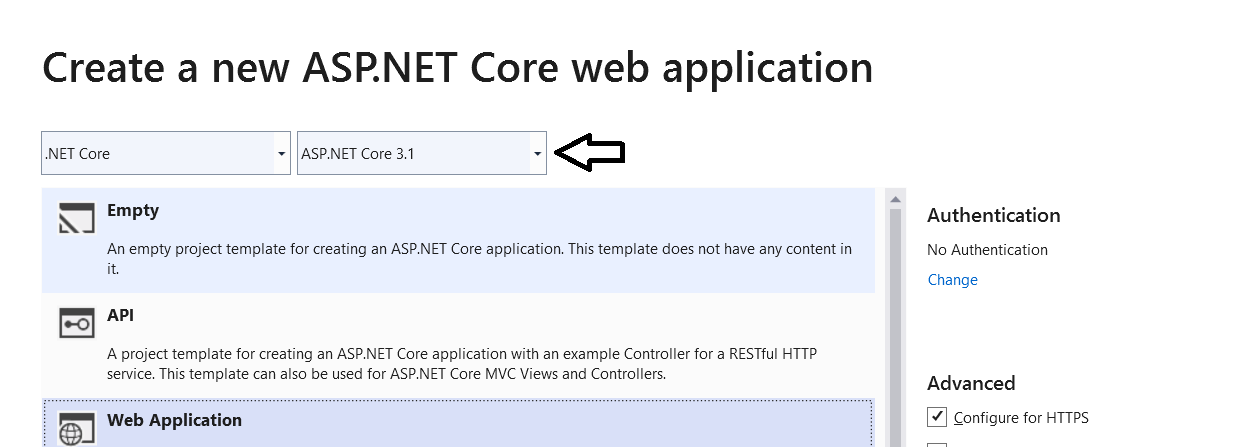
### Exercise 2: Create a new .NET Core Web Application in Visual Studio 2019

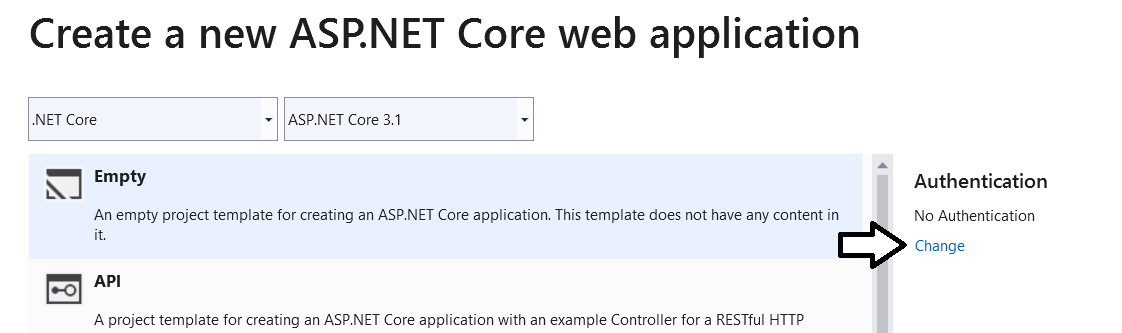
In this exercise you will create a new Web Application project using Visual Studio 2019 and the ASP.NET MVC framework.

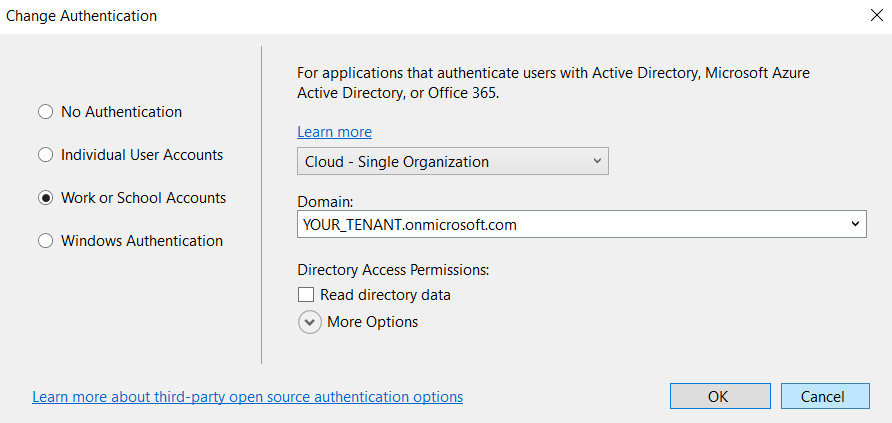
1. Launch **Visual Studio 2019**.
2. Create a new ASP.NET MVC project in Visual Studio 2019.
   1. In Visual Studio select **File > New > Project**.
   2. In the **New Project** dialog:
      1. Select **Installed > Templates > Visual C# > Web**.
      2. Select the **ASP.NET Web Application** project template.
      3. Name the new project **AppOwnsDataApp**.
      4. Add the new project into the folder at **C:\Student\Modules\05\_PowerBiEmbedding\Lab**.
      5. Click **OK** to display the **New ASP.Net Web Application** wizard.

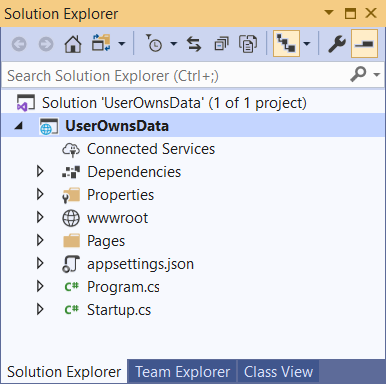


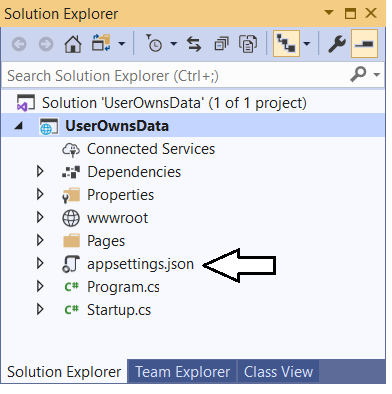












{

"AzureAd": {

"Instance": "https://login.microsoftonline.com/",

"Domain": "MSTDEV2020.onmicrosoft.com",

"TenantId": "a89a5a60-0169-470a-8032-de9e5e18cdf0",

"ClientId": "854409c8-b6ed-432d-bd79-68f6848a417b",

"CallbackPath": "/signin-oidc"

},

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft": "Warning",

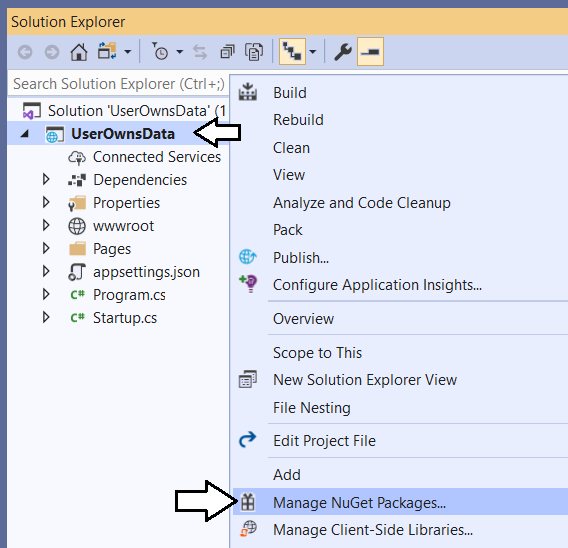
"Microsoft.Hosting.Lifetime": "Information"

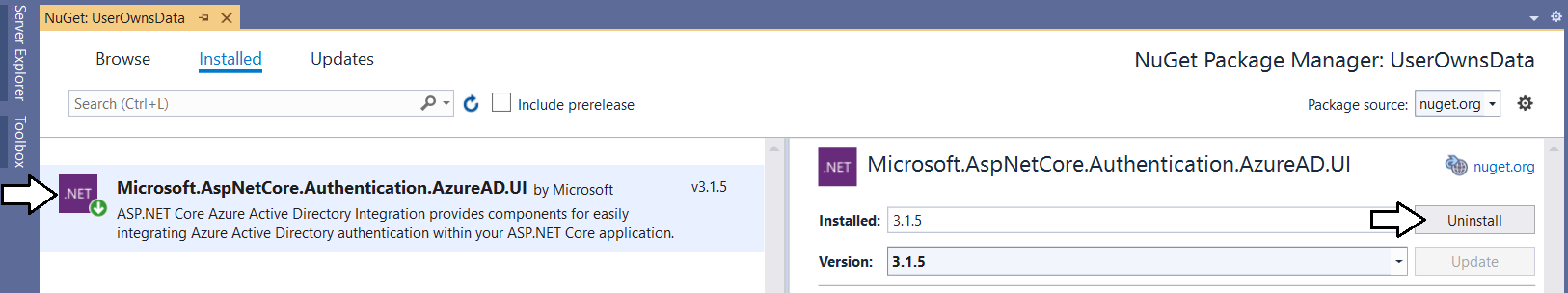
}

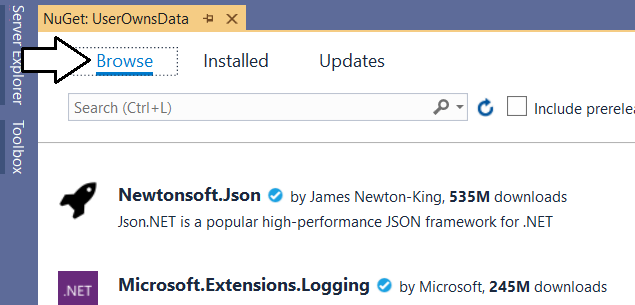
},

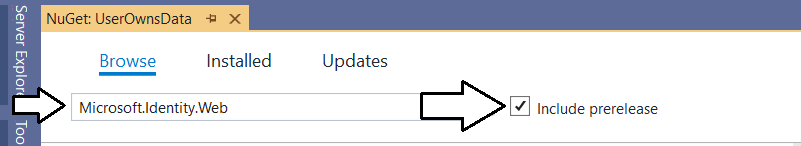
"AllowedHosts": "\*"

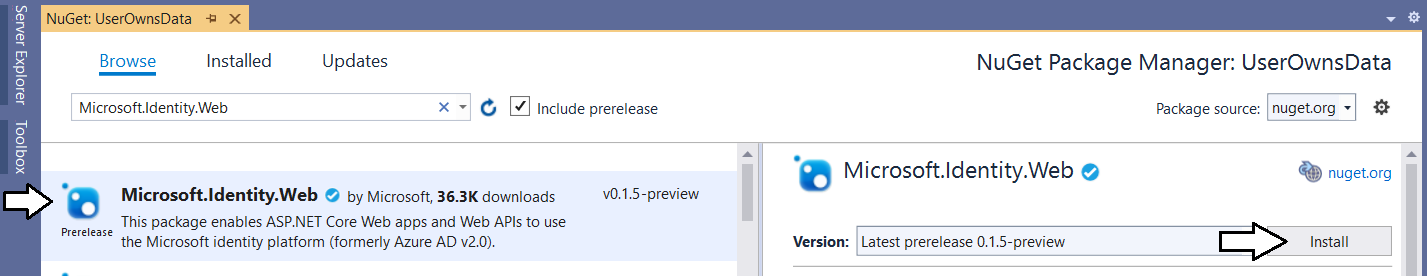
}

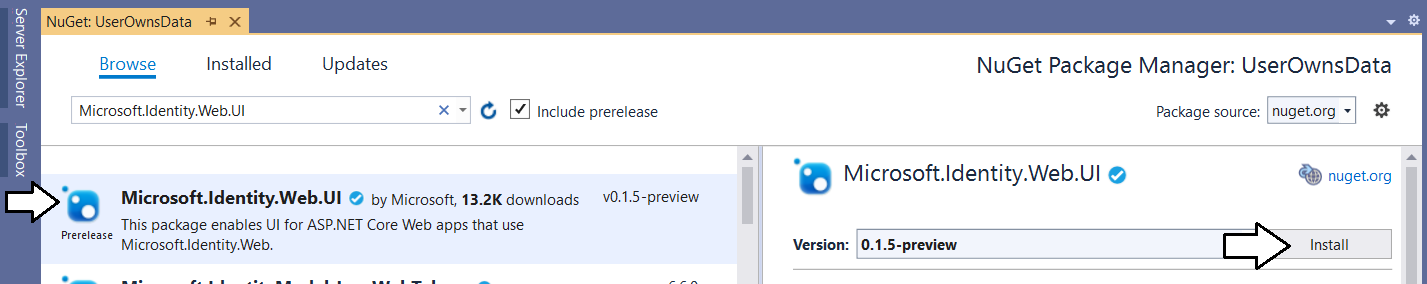


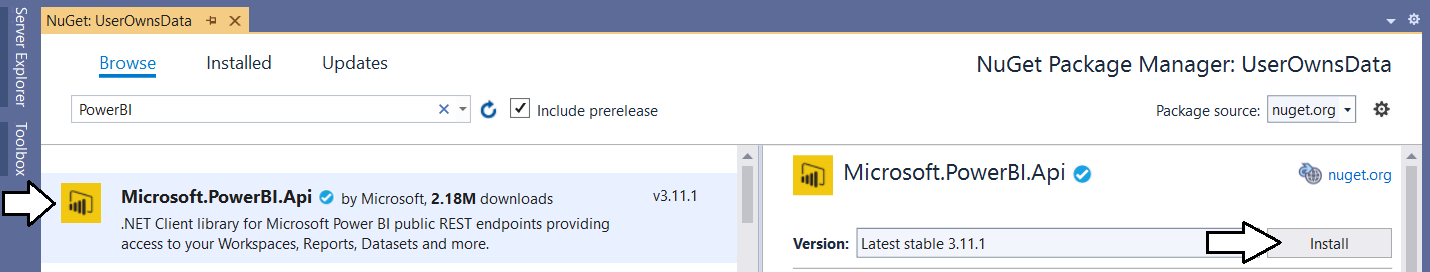


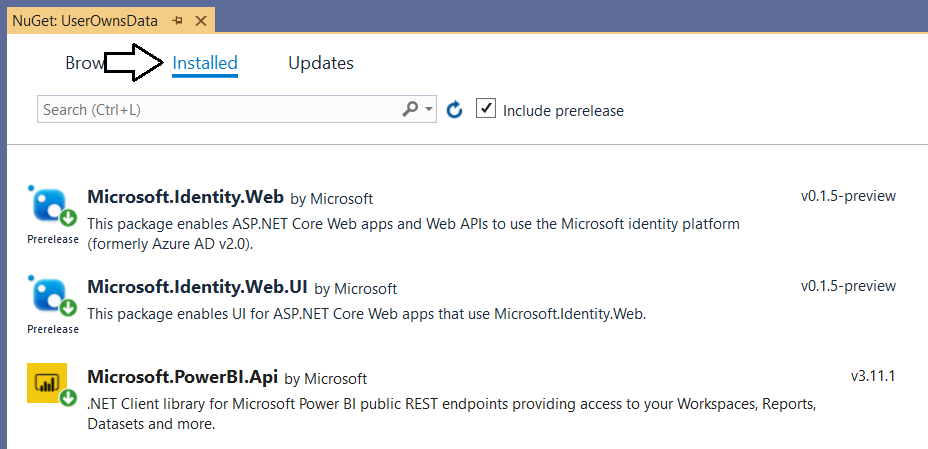


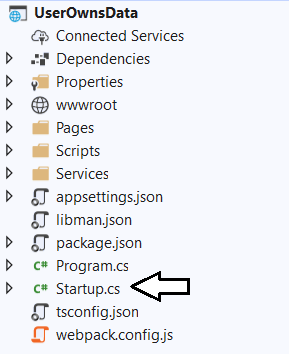












public void ConfigureServices(IServiceCollection services) {

services.AddAuthentication(AzureADDefaults.AuthenticationScheme)

.AddAzureAD(options => Configuration.Bind("AzureAd", options));

services.AddRazorPages().AddMvcOptions(options => {

var policy = new AuthorizationPolicyBuilder()

.RequireAuthenticatedUser()

.Build();

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

});

}

public void ConfigureServices(IServiceCollection services) {

//services.AddAuthentication(AzureADDefaults.AuthenticationScheme)

// .AddAzureAD(options => Configuration.Bind("AzureAd", options));

services.AddSignIn(Configuration);

services.AddRazorPages().AddMvcOptions(options => {

var policy = new AuthorizationPolicyBuilder()

.RequireAuthenticatedUser()

.Build();

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

});

}

public void ConfigureServices(IServiceCollection services) {

services.AddSignIn(Configuration);

services.AddControllersWithViews(options => {

var policy = new AuthorizationPolicyBuilder()

.RequireAuthenticatedUser()

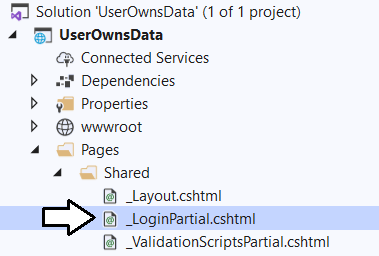
.Build();

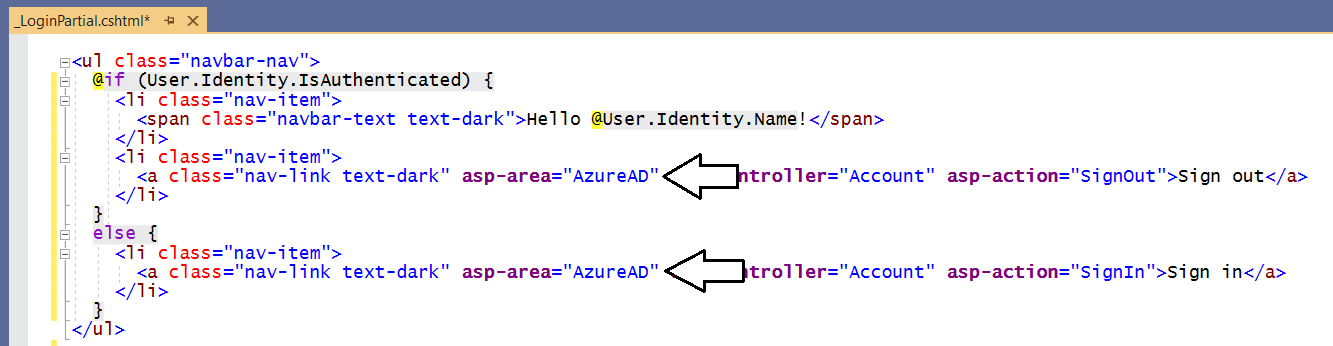
options.Filters.Add(new AuthorizeFilter(policy));

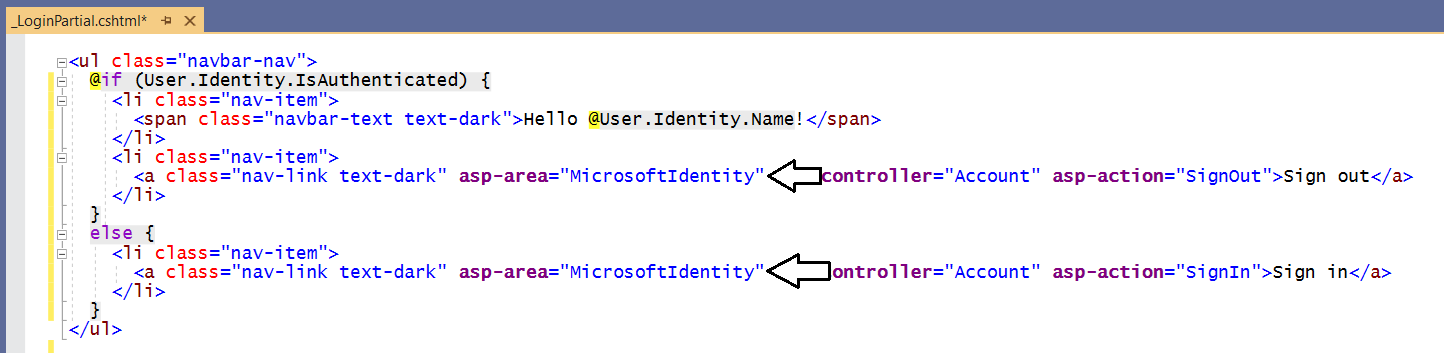
}).AddMicrosoftIdentityUI();

services.AddRazorPages();

}

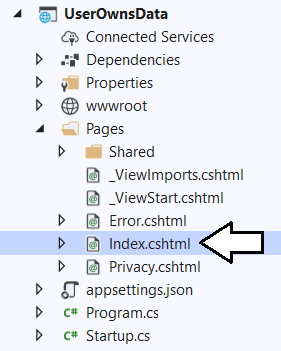


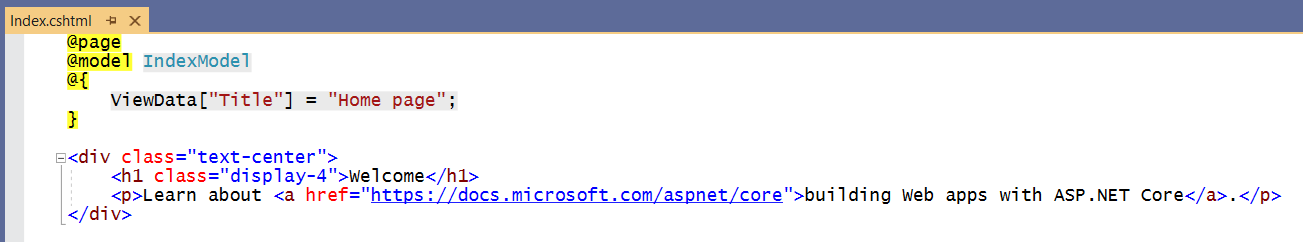




<span class="navbar-text text-dark">Hello @User.FindFirst("name").Value</span>







@page

@model IndexModel

@if (User.Identity.IsAuthenticated) {

string userName = @User.FindFirst("name").Value;

<div class="jumbotron">

<h1>Welcome @userName</h1>

<p class="lead">You are now logged into Azure AD as an authenticated user.</p>

</div>

}

else {

<div class="jumbotron">

<h1>Please login to get started</h1>

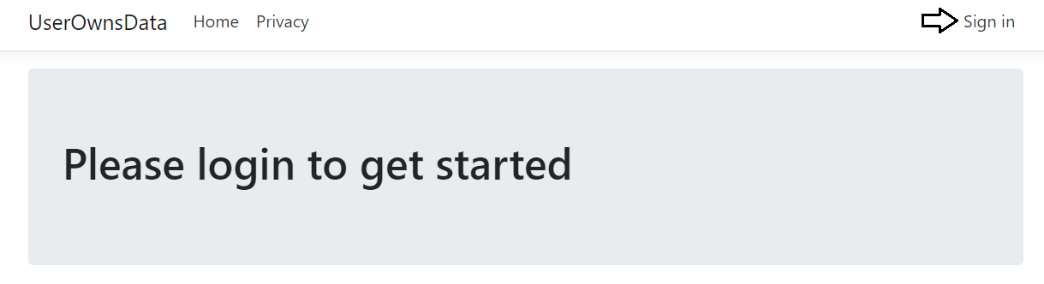
</div>

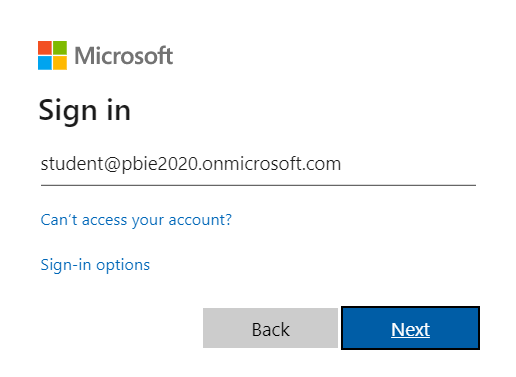
}

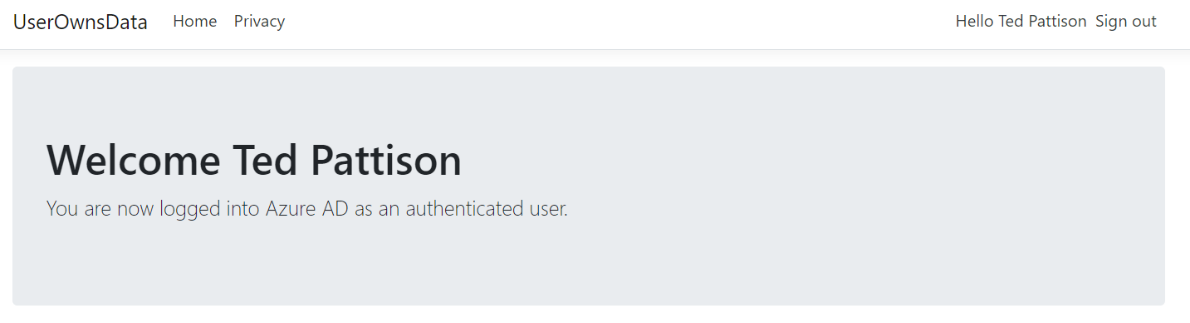
namespace UserOwnsData.Pages {

[AllowAnonymous]

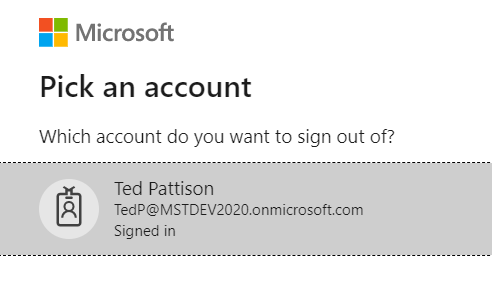
public class IndexModel : PageModel {





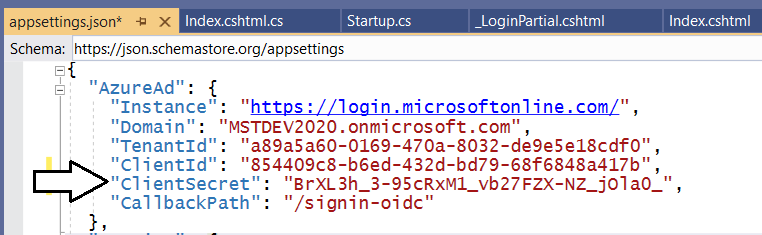


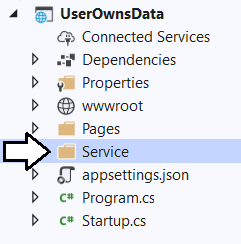


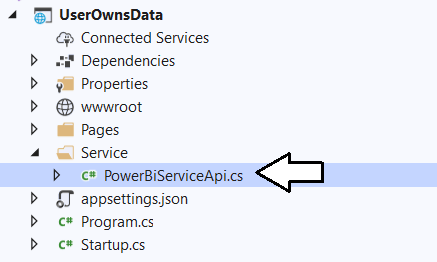


### Exercise 3: Call the Power BI Service API

In this exercise, you will.







using System;

using System.Threading.Tasks;

using Microsoft.Identity.Web;

using Microsoft.Rest;

using Microsoft.PowerBI.Api;

namespace UserOwnsData.Services {

public class EmbeddedReportViewModel {

public string Id;

public string Name;

public string EmbedUrl;

public string Token;

}

public class PowerBiServiceApi {

readonly ITokenAcquisition tokenAcquisition;

public PowerBiServiceApi(ITokenAcquisition tokenAcquisition) {

this.tokenAcquisition = tokenAcquisition;

}

const string urlPowerBiServiceApiRoot = "https://api.powerbi.com/";

public static readonly string[] RequiredScopes =

new string[] {

"https://analysis.windows.net/powerbi/api/Group.Read.All",

"https://analysis.windows.net/powerbi/api/Dashboard.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",

};

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

// get access token

var accessToken = this.tokenAcquisition.GetAccessTokenForUserAsync(RequiredScopes).Result;

// create PBI client to call Power BI Service API

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

PowerBIClient pbiClient = new PowerBIClient(new Uri(urlPowerBiServiceApiRoot), tokenCredentials);

// 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 = accessToken

};

}

}

}

Z

using Microsoft.Identity.Web.TokenCacheProviders.InMemory;

using UserOwnsData.Services;

It should currently look like this.

public void ConfigureServices(IServiceCollection services) {

services.AddSignIn(Configuration);

services.AddControllersWithViews(options => {

var policy = new AuthorizationPolicyBuilder()

.RequireAuthenticatedUser()

.Build();

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

}).AddMicrosoftIdentityUI();

services.AddRazorPages();

}

X

public void ConfigureServices(IServiceCollection services) {

services.AddSignIn(Configuration);

services

.AddWebAppCallsProtectedWebApi(Configuration, PowerBiServiceApi.RequiredScopes)

.AddInMemoryTokenCaches();

services.AddScoped(typeof(PowerBiServiceApi));

services.AddControllersWithViews(options => {

var policy = new AuthorizationPolicyBuilder()

.RequireAuthenticatedUser()

.Build();

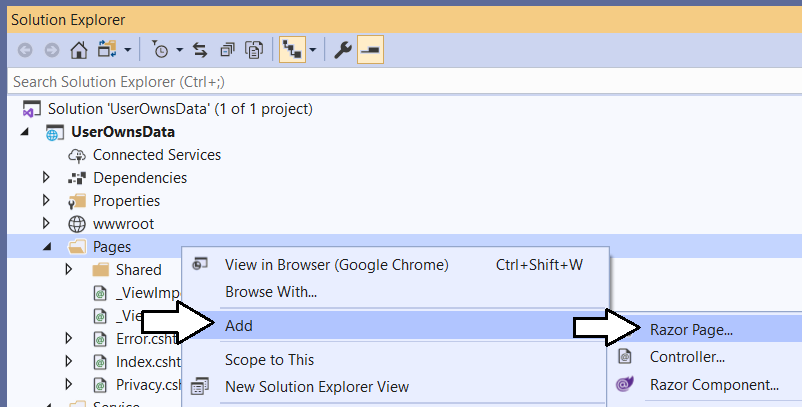
options.Filters.Add(new AuthorizeFilter(policy));

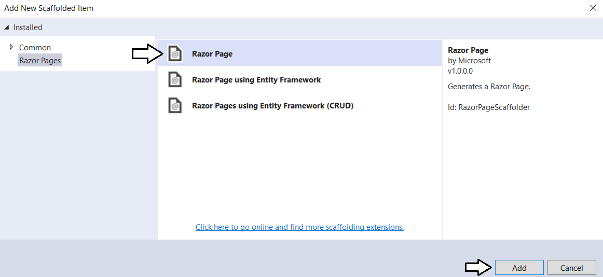
}).AddMicrosoftIdentityUI();

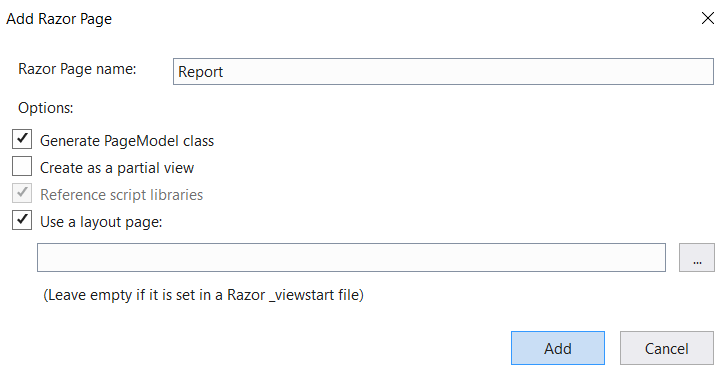
services.AddRazorPages();

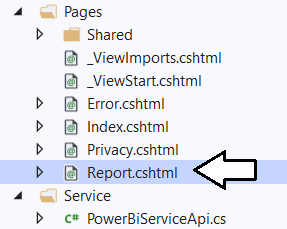
}

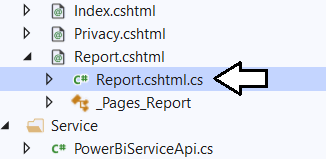
x











This is what it looks like at first

namespace UserOwnsData.Pages {

public class ReportModel : PageModel {

public void OnGet() {

}

}

}

Add this up top

using UserOwnsData.Services;

ssss

public class ReportModel : PageModel {

private PowerBiServiceApi powerBiServiceApi;

public ReportModel(PowerBiServiceApi powerBiServiceApi) {

this.powerBiServiceApi = powerBiServiceApi;

}

public EmbeddedReportViewModel report;

public void OnGet() {

}

}

switch from this

public void OnGet() {

to this

public async Task OnGet(PowerBiServiceApi powerBiServiceApi) {

Get workspace ID and report ID from earlier.

Guid workspaceId = new Guid("609c2d1b-c136-4bec-9a7b-e6829c50108b");

Guid reportId = new Guid("49398818-d952-4b52-a96b-226fec6b9911");

Xx

this.report = await powerBiServiceApi.GetReport(workspaceId, reportId);

x

public class ReportModel : PageModel {

private PowerBiServiceApi powerBiServiceApi;

public ReportModel(PowerBiServiceApi powerBiServiceApi) {

this.powerBiServiceApi = powerBiServiceApi;

}

public EmbeddedReportViewModel report;

public async Task OnGet() {

Guid workspaceId = new Guid("609c2d1b-c136-4bec-9a7b-e6829c50108b");

Guid reportId = new Guid("49398818-d952-4b52-a96b-226fec6b9911");

this.report = await powerBiServiceApi.GetReport(workspaceId, reportId);

}

}

X

@page

@model UserOwnsData.Pages.ReportModel

<h1>Embeded Report</h1>

<table class="table table-bordered">

<tr><td>Name</td><td>@Model.report.Name</td></tr>

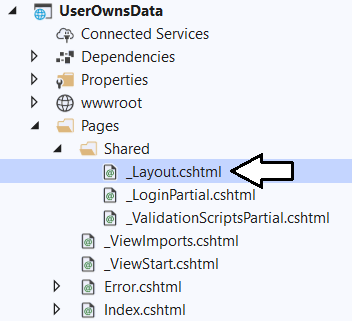
<tr><td>ID</td><td>@Model.report.Id</td></tr>

<tr><td>Embed Url</td><td style="word-break:break-all">@Model.report.EmbedUrl</td></tr>

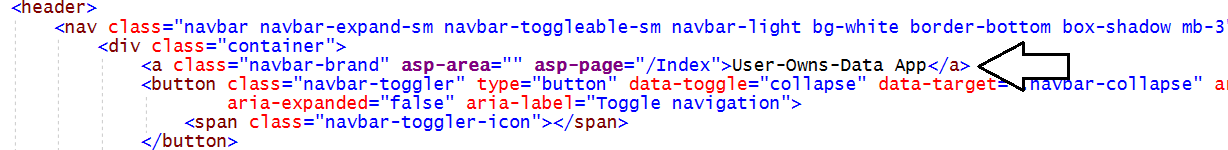
<tr><td>Token</td><td style="word-break:break-all">@Model.report.Token</td></tr>

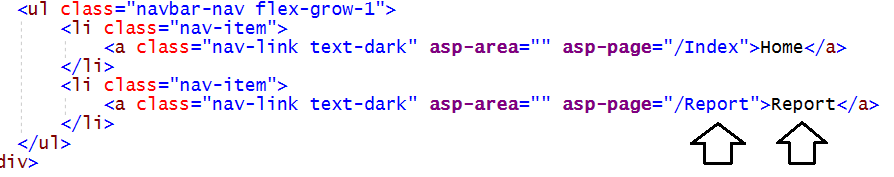
</table>

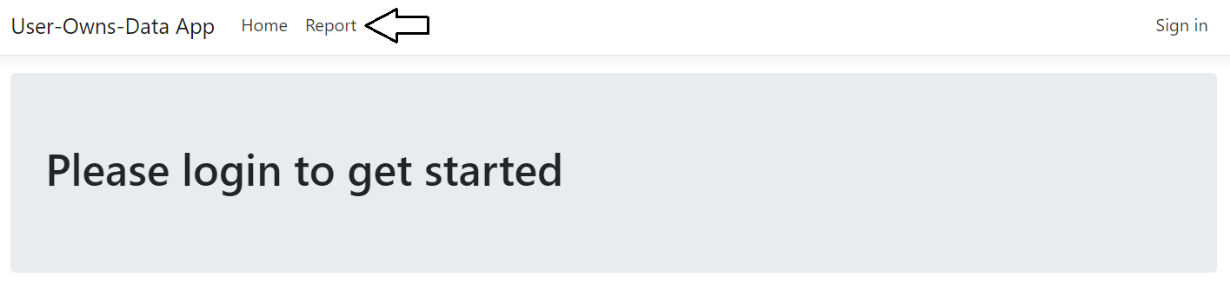
x

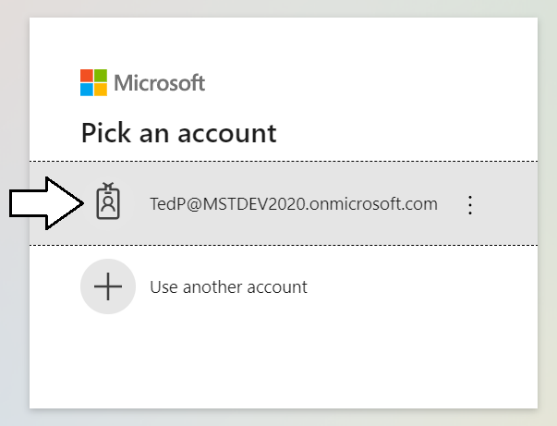


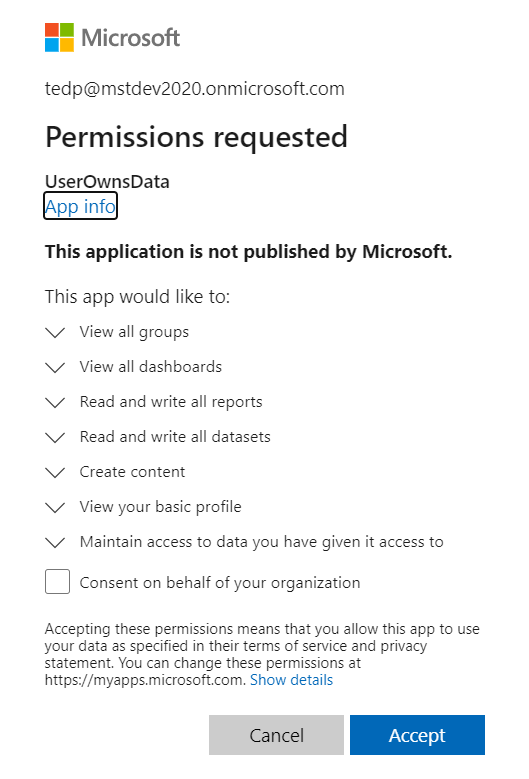


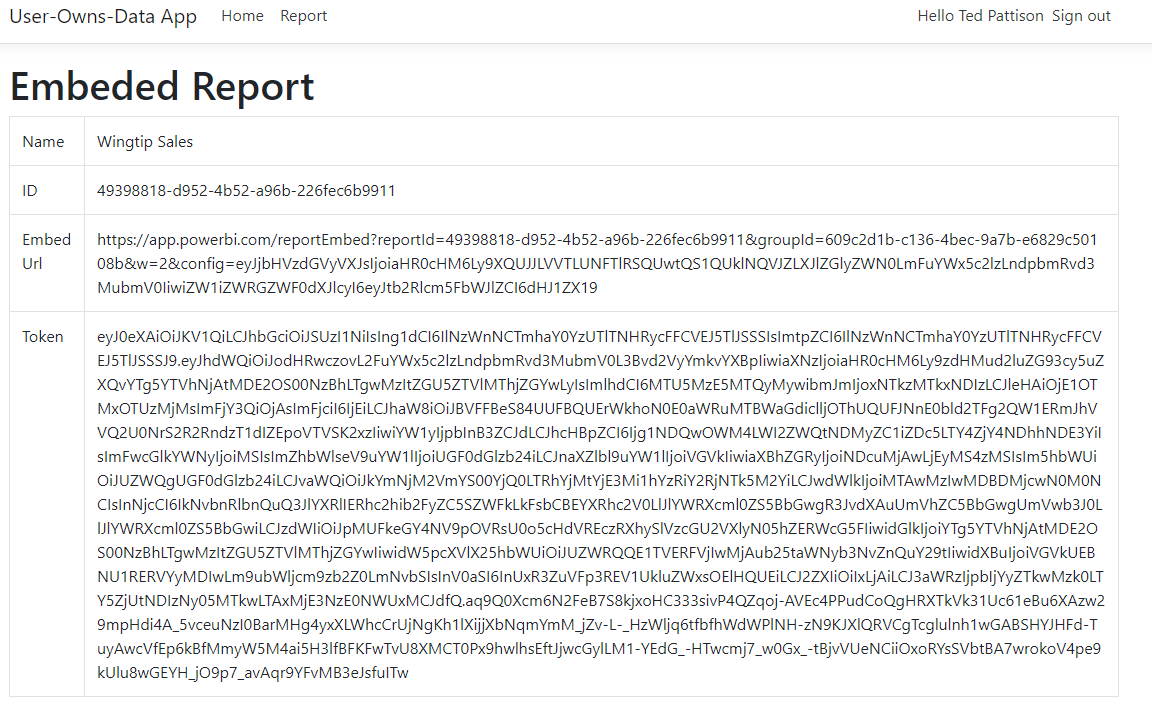






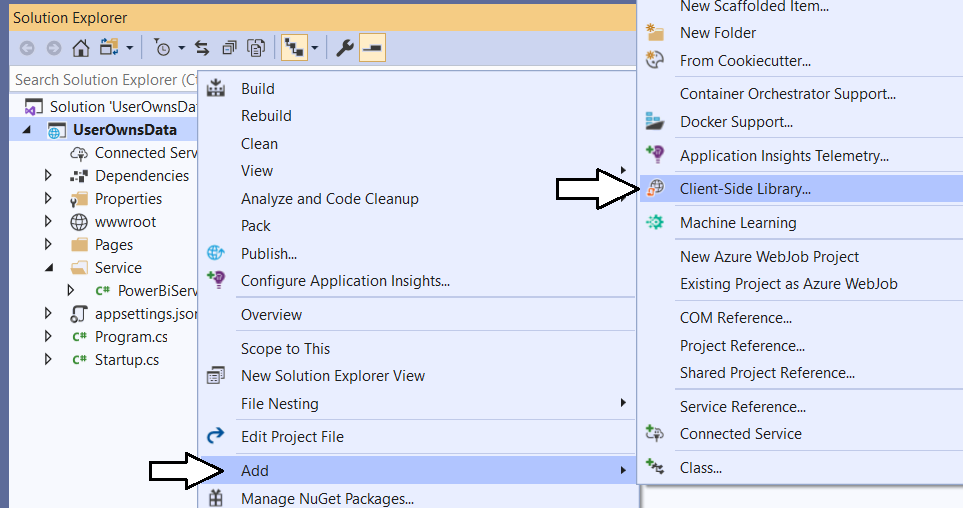


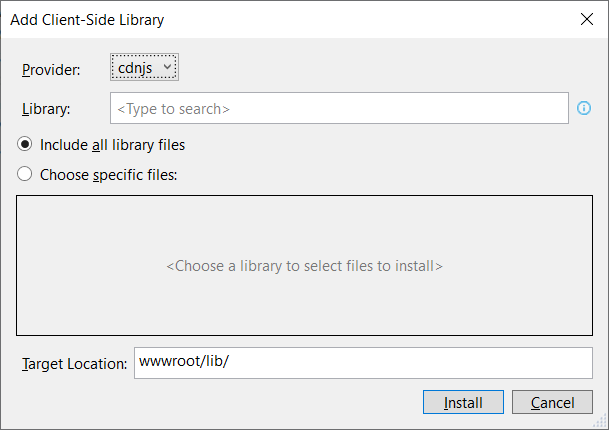


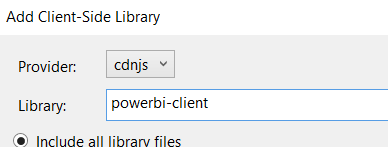


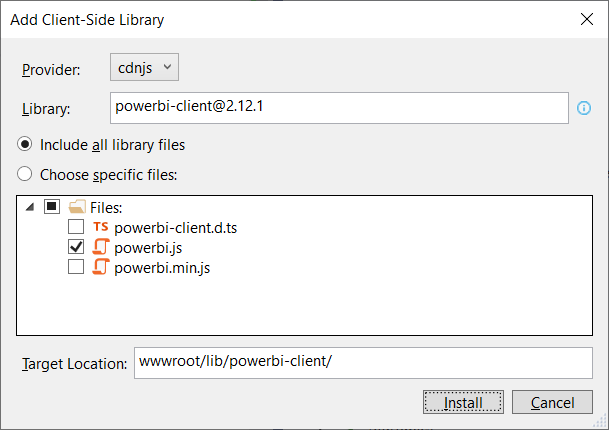
### Exercise 4: Embed a Report with the Power BI JavaScript API

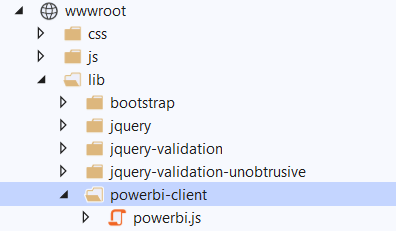
In this exercise, you will.

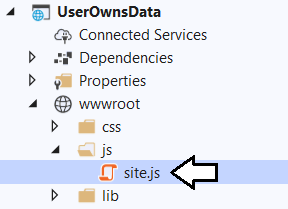


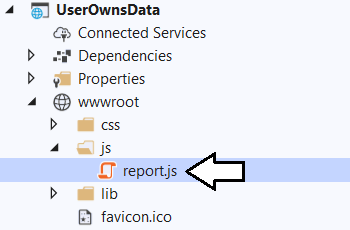












function embedReport(reportContainer, reportId, embedUrl, token) {

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: {

filterPaneEnabled: false,

navContentPaneEnabled: true,

}

};

// Embed the report and display it within the div container.

var report = powerbi.embed(reportContainer, config);

}