1. OAuth 2.0 and Open ID Connect
   1. Creating and Configuring Azure AD Application
   2. Delegated Permissions versus Application Permissions
   3. Understanding Scopes and the Default Required Permission Set
   4. The Common Consent Framework and Granting Permissions
   5. Migrating from ADAL to MSAL
   6. Getting Started with the Microsoft Authentication Library
      1. Making calls without Microsoft.PowerBI.API
      2. Making calls with Microsoft.PowerBI.API
2. Developing Public Client Application
   1. The Role of ReplyUrls in User Authentication
   2. Interactive Login from a Desktop Application
   3. User Password Credential Flow
   4. Device Code Flow
3. Developing Confidential Client Applications
   1. Client Credentials Flow
   2. Implicit Flow for Single Page Applications
   3. Implementing Web Application using Authorization Grant Flow

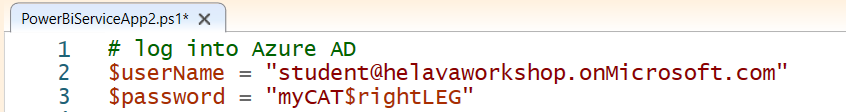
### Exercise 5: Authenticate using the Microsoft Authentication Library (MSAL)

In this exercise, you will create another console application which will use the Power BI .NET SDK to call the Power BI Service API. However, the console application will be different from the one created earlier because you will use the *Microsoft Authentication Library (MSAL)* instead of the *Azure Active Directory Authentication library (ADAL)*. This will give you a chance to see what's different between *MSAL* and *ADAL* and you'll learn how Power BI Service API permissions can be incrementally expanded over time.

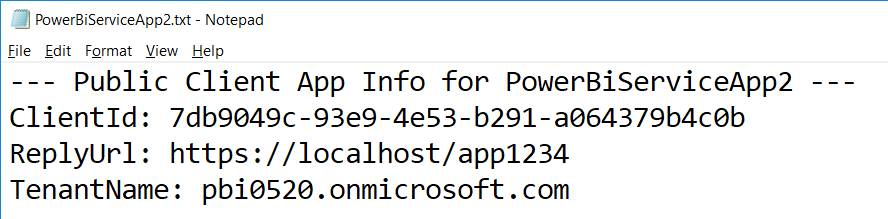
1. Use a PowerShell script to create a new public client application in your Azure AD tenant.
   1. Open a PowerShell script editor such as the PowerShell ISE or Visual Studio Code.
   2. Open the PowerShell script named **RegisterPowerBiServiceApp2.ps1** which is located at the following path.

C:\Student\Modules\04\_PowerBiServiceAPI\Lab\Scripts\RegisterPowerBiServiceApp2.ps1

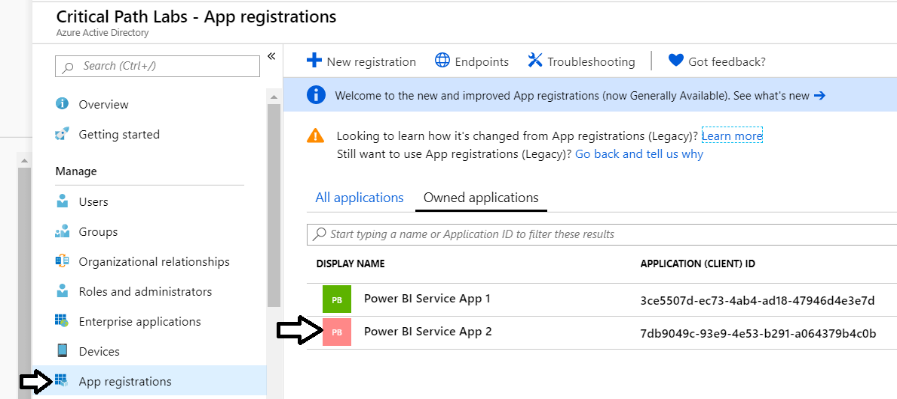
* 1. Update the variables named **$userName** and **$password** with the credentials for your Office 365 user account.



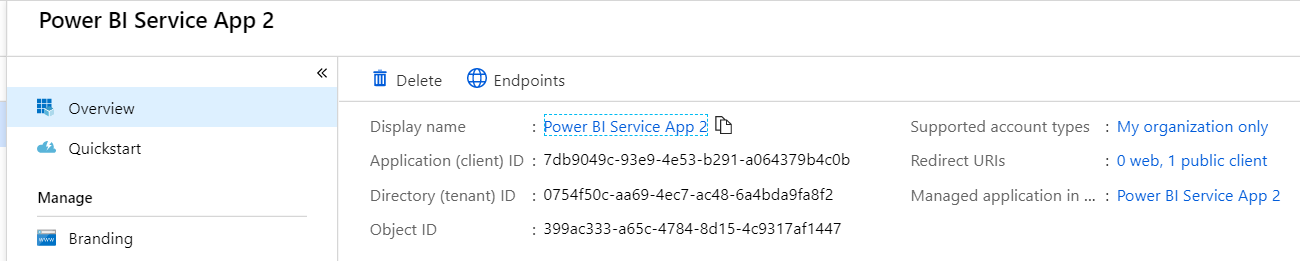
* 1. Save you changes to **PowerBiServiceApp2.ps1** and run the script.
  2. When the script runs, it will create a new public client application and display application details in a text file shown in Notepad.



1. Inspect the new public client application in the Azure portal.
   1. Navigate to the Azure portal at <https://portal.azure.com/>.
   2. Once you are log in, check the email address in the login menu to make sure you are logged with the correct identity.
   3. Click on the **Azure Active Directory** link in the left navigation and then click the link for **App registration**.
   4. Locate and click the link for the new app named **Power BI Service App 2**.

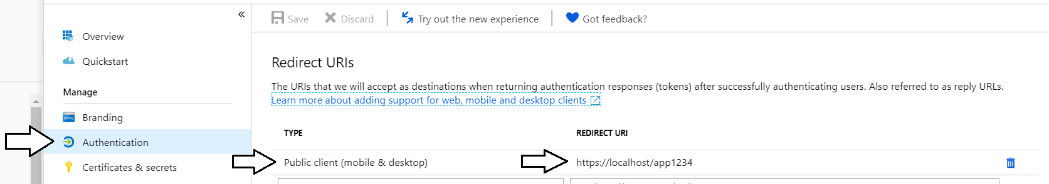


* 1. You should now see the summary page for **Power BI Service App 2**.

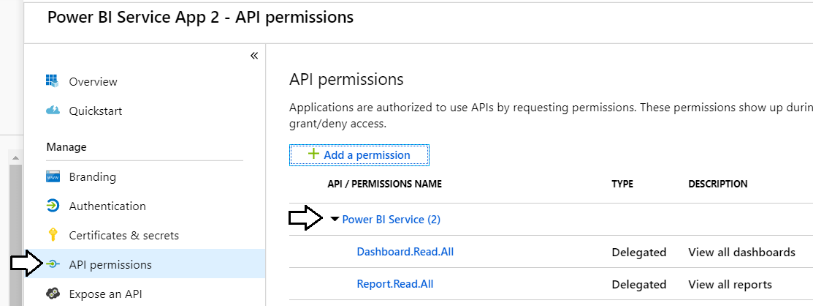


Note that you do not need to modify this new Azure AD application because the PowerShell script was able to configure it with all the required settings. However, you will now quickly review the application settings that were configured by the PowerShell script.

* 1. Click the **Authentication** link on the left.
  2. You should be able to verify that the **TYPE** is set to **Public client** and **REDIRECT URI** is set to **https://localhost/app1234**.

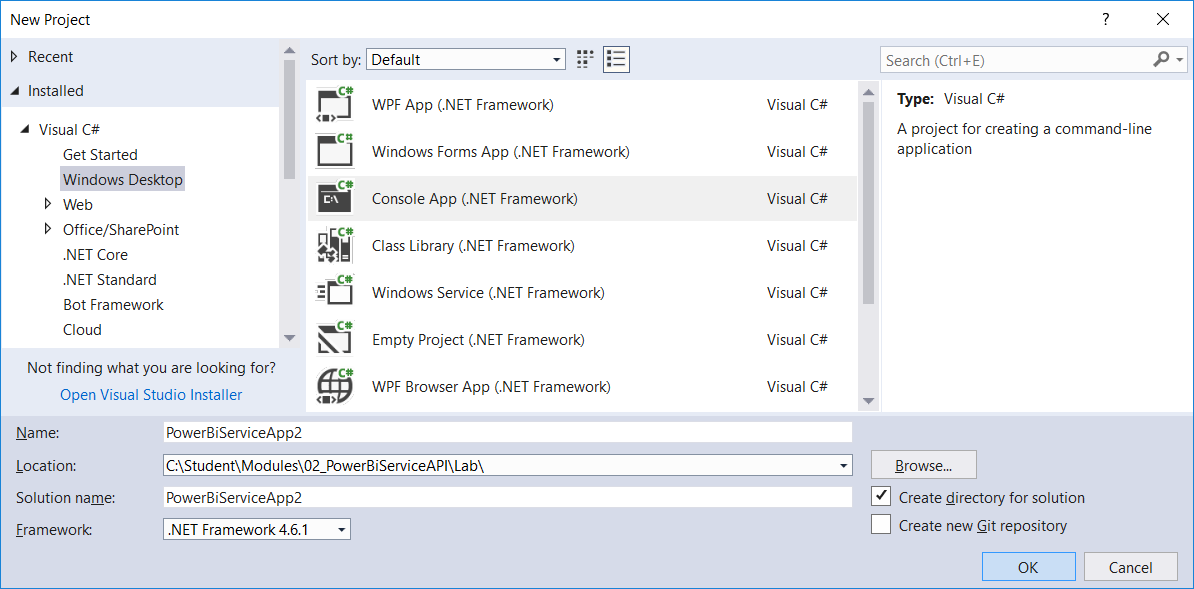


* 1. Click the **API Permissions** link on the left.
  2. You should be able to verify that app has two Power BI permissions which are **Dashboard.Read.All** and **Report.Read.All**.

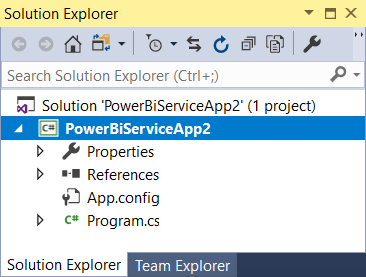


Now you have seen that an Azure AD application can be created and configured using a PowerShell script. Now it's time to move ahead and create an application that authenticates with this Azure AD application using the Microsoft Authentication Library (MSAL).

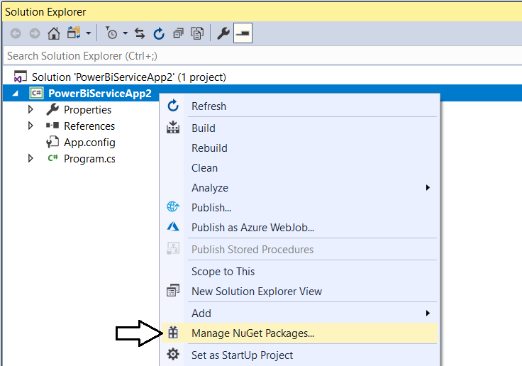
1. Create a new C# Console application in Visual Studio.
   1. Launch a new instance of Visual Studio.
   2. Create a new project by running the **File > New Project** command.
   3. Select a project type of **Console App (.NET Framework)** from the **Visual C# > Windows Desktop** project templates.
   4. Give the project a **Name** of **PowerBiServiceApp2**.
   5. Give the project a **Location** of **C:\Student\Modules\04\_PowerBiServiceAPI\Lab**. and click **OK** to create the new project.



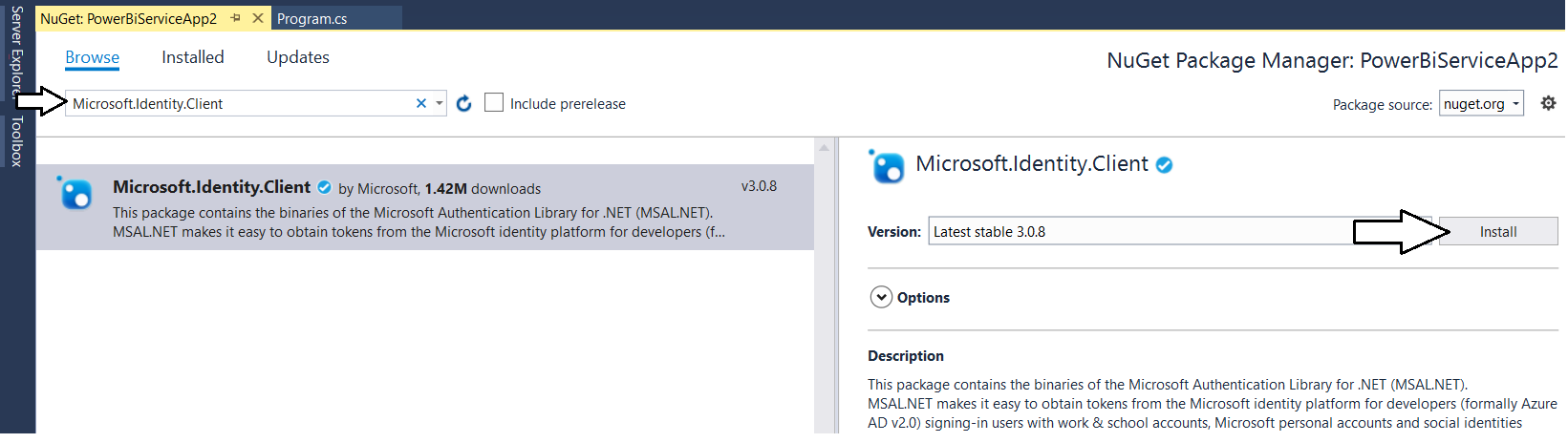
* 1. You should now have a new project named **PowerBiServiceApp2**.



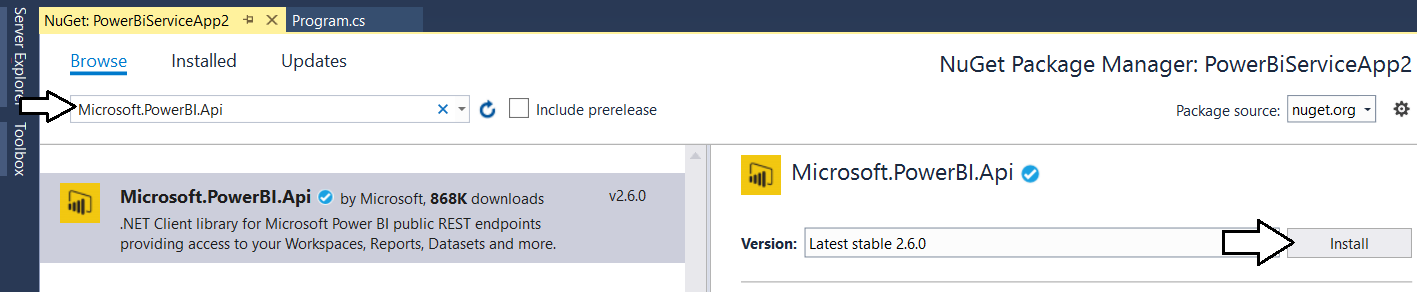
1. Add the NuGet packages to the project required to program the Power BI Service API using the Power BI .NET SDK.
   1. Right-click the top-level node for the **PowerBiServiceApp2** project and select **Manage NuGet Packages…**.



* 1. Click the Browse tab and type **Microsoft.Identity.Client** into the search box.
  2. Locate and install the package **Microsoft.Identity.Client**. This is the package for the *Microsoft Authentication library (MSAL)*.

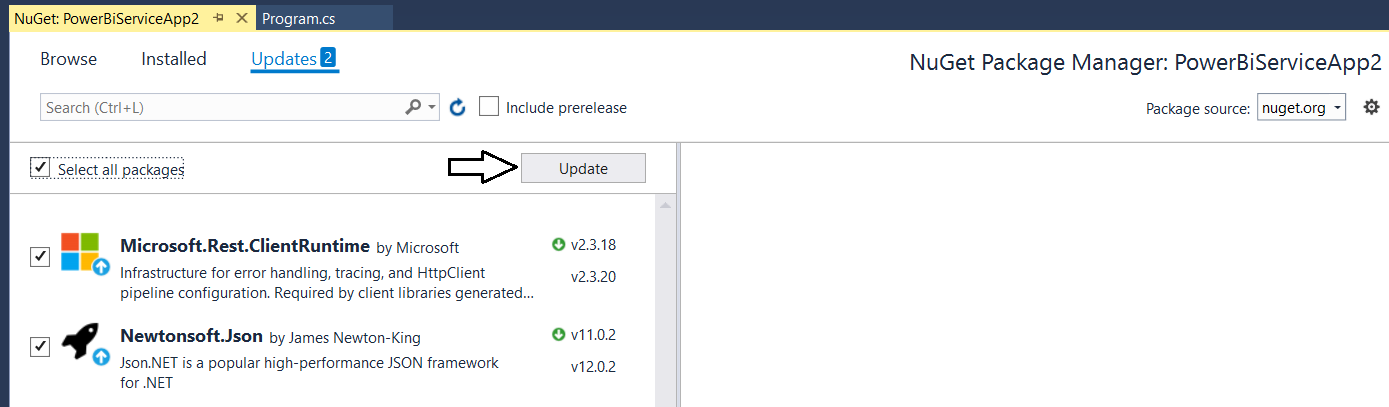


* 1. If you are prompted about **Preview Changes**, click **OK**.
  2. When prompted about **License Acceptance**, click **I Agree**.
  3. Search for and install the package named **Microsoft.PowerBI.Api** to add the Power BI .NET SDK to your project.

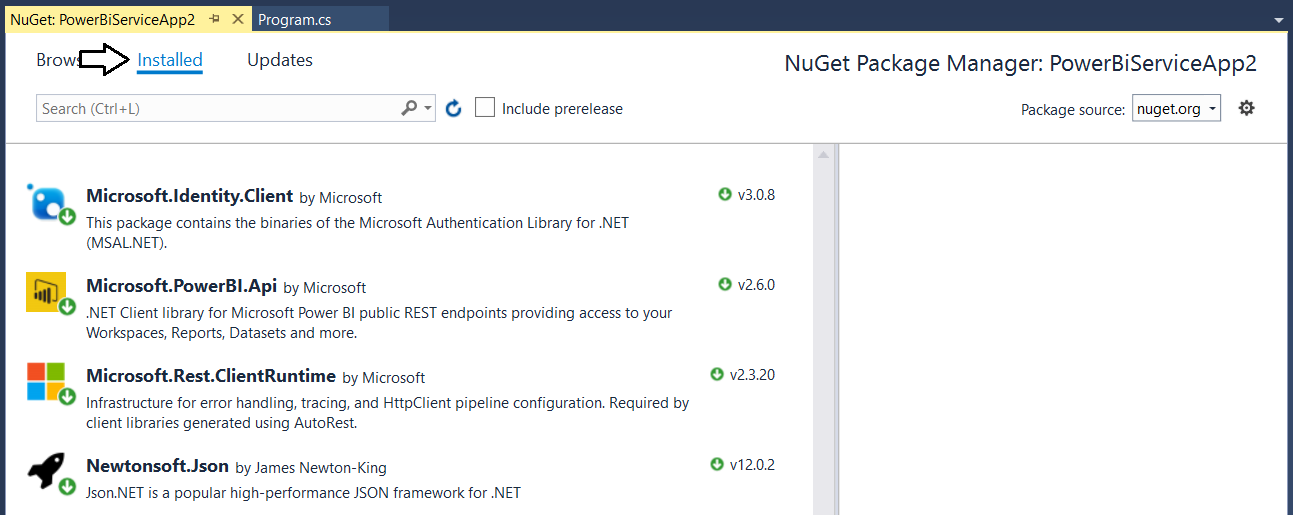


* 1. When prompted about the licensing agreement, click **I Agree**.

1. Update all NuGet packages.
   1. Navigate to the **Update** tab and update any packages that have updates available.



* 1. Click on the **Installed** tab and ensure you have the following four packages installed.

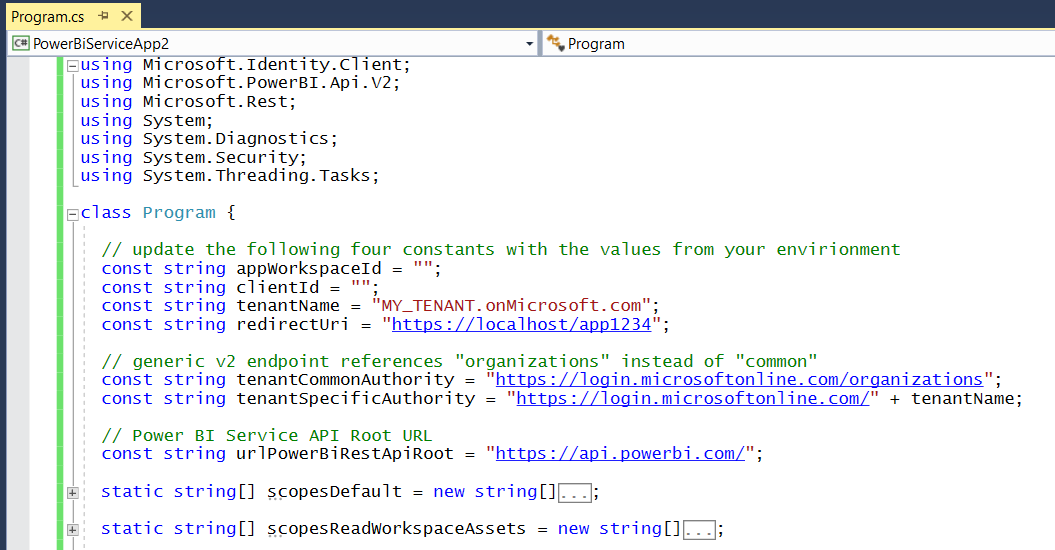


* 1. Close the window for the Nuget Package Manager.

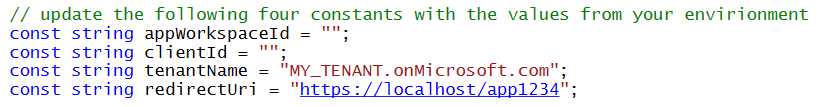
1. Add the C# starter code to **program.cs**.
   1. Using Windows Explorer, locate the file **PowerBiServiceApp2\_Starter.cs.txt** in the **Student** folder at the following path.

C:\Student\Modules\02\_PBIRestApi\Lab\StarterFiles\PowerBiServiceApp2\_Starter.cs.txt

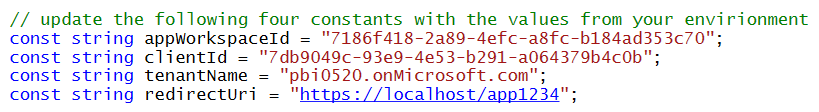
* 1. Open the file named **PowerBiServiceApp2\_Starter.cs.txt** in Notepad and copy its contents into the Window clipboard.
  2. Return to the **PowerBiServiceApp2** project in Visual Studio.
  3. Open the source file named **program.cs**.
  4. Delete all the code inside **program.cs** and replace it with the content you copied into the Windows clipboard.
  5. You should now have the basic C# code for a simple console application which access the Power BI Service API.



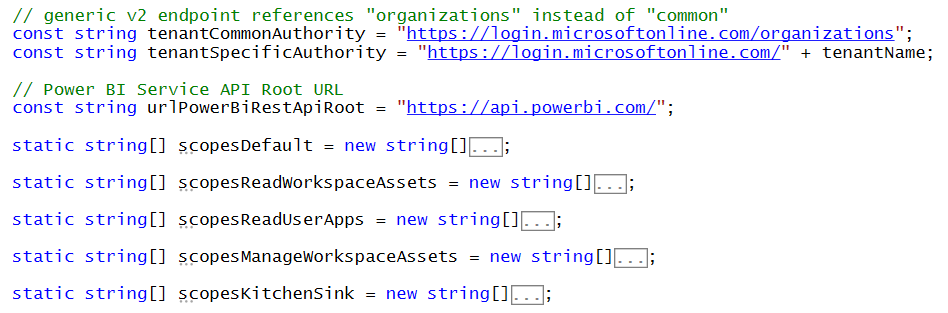
* 1. At the top of the **Program** class, you will find four constants named **appWorkspaceId**, **clientId, tenantName** and **redirectUri**.



* 1. Modify these constants with the specific values for your Power BI app workspace and your Azure AD tenant.



1. Review the pre-provided code inside **Program.cs**.
   1. You should see three constants named **tenantCommonAuthority, tenantSpecificAuthority** and **urlPowerBiRestApiRoot**.
   2. There are static string array fields with names starting with "**scopes**" which represent delegated permission sets.



* 1. Move down in **Program.cs** and inspect the implementation of the static method named **GetAccessTokenInteractive**.

static string GetAccessTokenInteractive(string[] scopes) {

var appPublic = PublicClientApplicationBuilder.Create(clientIdPublicApp)

.WithAuthority(tenantAuthrority)

.WithRedirectUri(redirectUri)

.Build();

var authResult = appPublic.AcquireTokenInteractive(scopes)

.WithPrompt(Prompt.SelectAccount)

.ExecuteAsync().Result;

return authResult.AccessToken;

}

* 1. Move down in **Program.cs** and inspect the implementation of the static function named **DisplayAppWorkspaceAssets**.

static void DisplayAppWorkspaceAssets() {

string AccessToken = GetAccessTokenInteractive(scopesDefault);

var pbiClient = new PowerBIClient(new Uri(urlPowerBiRestApiRoot),

new TokenCredentials(AccessToken, "Bearer"));

Console.WriteLine();

Console.WriteLine("Dashboards:");

var dashboards = pbiClient.Dashboards.GetDashboardsInGroup(appWorkspaceId).Value;

foreach (var dashboard in dashboards) {

Console.WriteLine(" - " + dashboard.DisplayName + " [" + dashboard.Id + "]");

}

Console.WriteLine();

Console.WriteLine("Reports:");

var reports = pbiClient.Reports.GetReportsInGroup(appWorkspaceId).Value;

foreach (var report in reports) {

Console.WriteLine(" - " + report.Name + " [" + report.Id + "]");

}

//Console.WriteLine();

//Console.WriteLine("Datasets:");

//var datasets = pbiClient.Datasets.GetDatasetsInGroup(appWorkspaceId).Value;

//foreach (var dataset in datasets) {

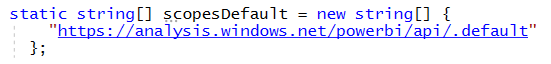
// Console.WriteLine(" - " + dataset.Name + " [" + dataset.Id + "]");

//}

Console.WriteLine();

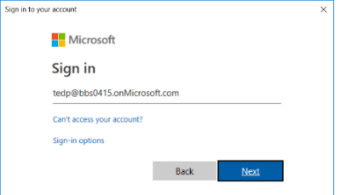
}

* 1. Code in **DisplayAppWorkspaceAssets** calls **GetAccessTokenInteractive** passing a parameter value of **scopesDefault**.

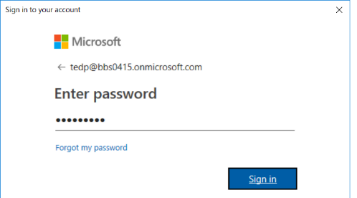


The first time you run the program, Azure AD will prompt you to consent to the default permissions configured for the application.

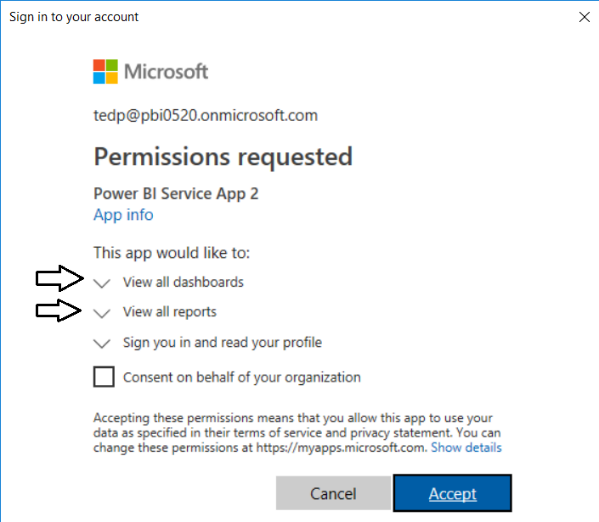
1. Run the application to test your work.
   1. Run the console application in the Visual Studio debugger by pressing the **CTRL** + **{F5}** keyboard combination.



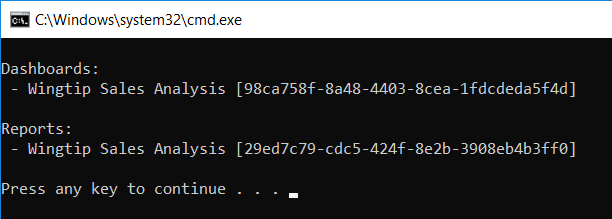
* 1. When prompted to sign in, enter your user name and password.



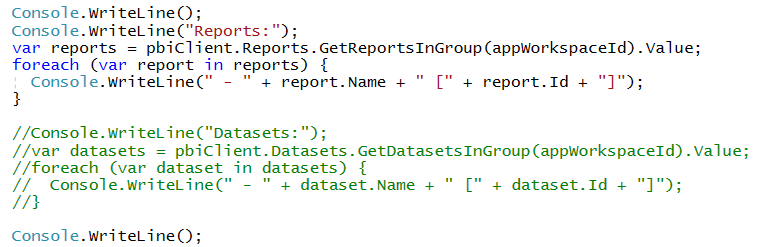
* 1. When prompted to consent to the default permissions of **View all dashboards** and **View all reports**, click **Accept**.



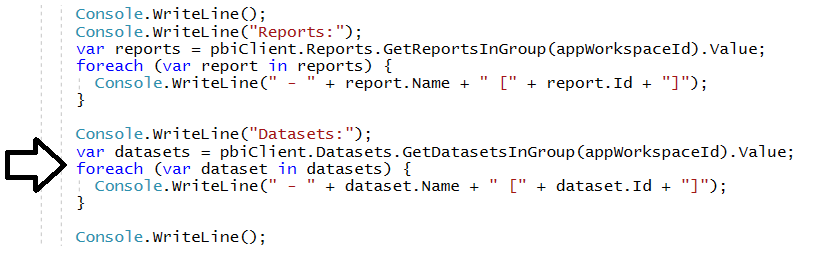
* 1. The program should run and display the dashboard and report in the **Wingtip Sales** app workspace.



1. Try running the console application again after uncommenting the code to retrieve information about datasets
   1. Locate the commented code at the bottom of the **DisplayAppWorkspaceAssets** method and uncomment it.

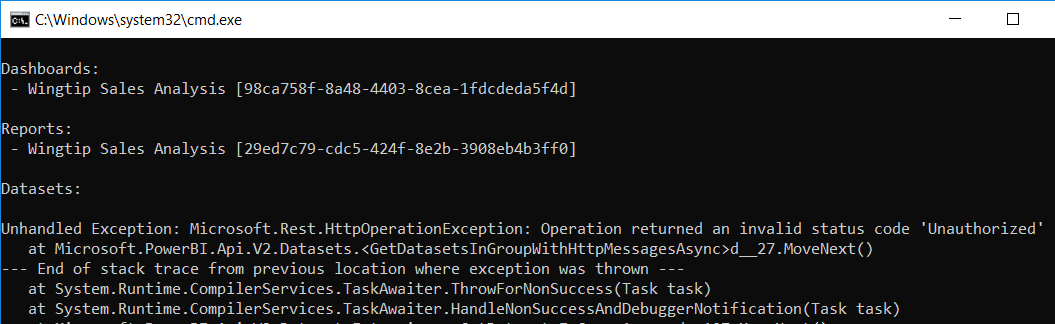


* 1. There should now be code in **DisplayAppWorkspaceAssets** that calls **GetDatasetsInGroup**.



Note that the default permission set does not include the required permissions to call **GetDatasetsInGroup**.

* 1. Run the console application in the Visual Studio debugger by pressing the **CTRL** + **{F5}** keyboard combination.
  2. When prompted, sign in.
  3. The program should run and then fail with an ***Unauthorized*** exception when it attempts to call **GetDatasetsInGroup**.



* 1. Close the console window and then return to your project in Visual Studio and the source file named **program.cs**.

1. Acquire an access token interactively with the required scopes to successfully call **DisplayAppWorkspaceAssets**.
   1. Inspect the static field named **scopesReadWorkspaceAssets** to see what scopes it contains.

static string[] scopesReadWorkspaceAssets = new string[] {

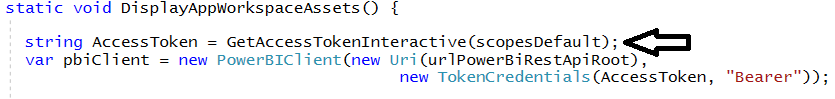
"https://analysis.windows.net/powerbi/api/Dashboard.Read.All",

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

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

};

* 1. Inspect the code inside **DisplayAppWorkspaceAssets** to find where it calls **GetAccessTokenInteractive**.



* 1. In the call to **DisplayAppWorkspaceAssets**, replace **scopesDefault** with **scopesReadWorkspaceAssets**.

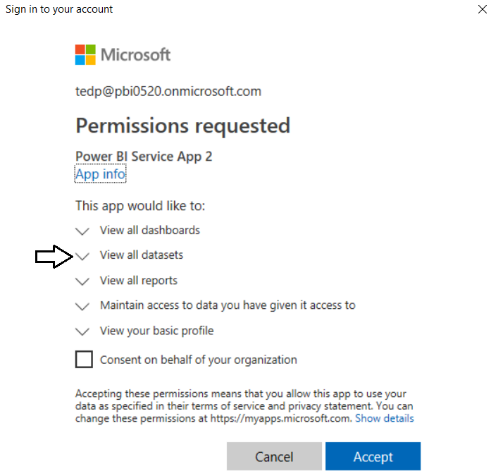
static void DisplayAppWorkspaceAssets() {

string AccessToken = GetAccessTokenInteractive(scopesReadWorkspaceAssets);

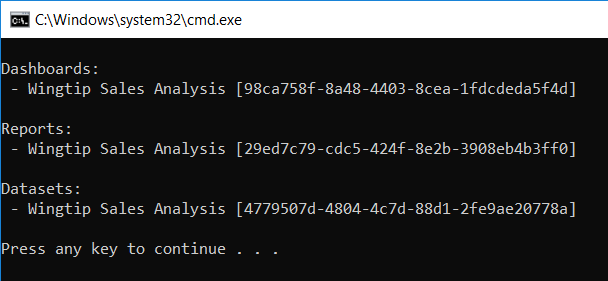
var pbiClient = new PowerBIClient(new Uri(urlPowerBiRestApiRoot),

new TokenCredentials(AccessToken, "Bearer"));

* 1. Run the console application again by pressing the **CTRL** + **{F5}** keyboard combination.
  2. When prompted, sign in.
  3. After signing in, you should be prompted to consent to permissions including **View all datasets**.



* 1. The program should now succeed when calling **GetDatasetsInGroup**.



1. Authenticate using (*the dreaded*) *User Credential Password* flow.
   1. Review the C# code inside the method named **GetAccessTokenWithUserPassword**.

static string GetAccessTokenWithUserPassword(string[] scopes) {

var appPublic = PublicClientApplicationBuilder.Create(clientId)

.WithAuthority(tenantCommonAuthority)

.Build();

string username = "user1@MY\_TENANT.onMicrosoft.com";

string userPassword = "";

SecureString userPasswordSecure = new SecureString();

foreach (char c in userPassword) {

userPasswordSecure.AppendChar(c);

}

var authResult = appPublic.AcquireTokenByUsernamePassword(scopes, username, userPasswordSecure)

.ExecuteAsync().Result;

return authResult.AccessToken;

}

* 1. Locate the two variables inside **GetAccessTokenWithUserPassword** named **username** and **userPassword**.
  2. Update **username** and **userPassword** with your user name and password of your primary Office 365 user account.

string username = "tedp@pbi0520.onMicrosoft.com";

string userPassword = "myCAT$rightLEG";

* 1. Move down to the method named **DisplayAppWorkspaceAssets** and locate the call to **GetAccessTokenInteractive**.

static void DisplayAppWorkspaceAssets() {

string AccessToken = GetAccessTokenInteractive(scopesReadWorkspaceAssets);

var pbiClient = new PowerBIClient(new Uri(urlPowerBiRestApiRoot),

new TokenCredentials(AccessToken, "Bearer"));

* 1. Replace the call to **GetAccessTokenInteractive** with a call to **GetAccessTokenWithUserPassword**.

static void DisplayAppWorkspaceAssets() {

string AccessToken = GetAccessTokenWithUserPassword(scopesReadWorkspaceAssets);

var pbiClient = new PowerBIClient(new Uri(urlPowerBiRestApiRoot),

new TokenCredentials(AccessToken, "Bearer"));

* 1. Run the program again. This time it should run successfully without requiring any interactive behavior.

Keep in mind that using the *User Credential Password* flow like this should not be used in a secure environment. In fact, the *Microsoft Authentication Library (MSAL)* prohibits this *User Credential Password* flow for code running in the .NET CORE environment. It is recommended that you learn how to authenticate users with an authentication flow that's more secure such as the *Device Code Flow*.

1. Authenticate the user using the *Device Code Authentication* flow.
   1. Review the method named **GetAccessTokenWithDeviceCode**.

static string GetAccessTokenWithDeviceCode(string[] scopes) {

// device code authentication requires tenant-specific authority URL

var appPublic = PublicClientApplicationBuilder.Create(clientId)

.WithAuthority(tenantSpecificAuthority)

.Build();

// this method call will block until you have logged in using the generated device code

var authResult = appPublic.AcquireTokenWithDeviceCode(scopes, deviceCodeCallbackParams => {

// retrieve device code and verification URL from deviceCodeCallbackParams

string deviceCode = deviceCodeCallbackParams.UserCode;

string verificationUrl = deviceCodeCallbackParams.VerificationUrl;

Console.WriteLine();

Console.WriteLine("When prompted, copy-and-paste the following device code: " + deviceCode);

Console.WriteLine();

Console.WriteLine("Opening Browser at " + verificationUrl);

Process.Start("chrome.exe", verificationUrl);

Console.WriteLine();

Console.WriteLine("This console app will now block until you enter the device code and log in");

// return task result

return Task.FromResult(0);

}).ExecuteAsync().Result;

Console.WriteLine("The call to AcquireTokenWithDeviceCode has completed");

return authResult.AccessToken;

}

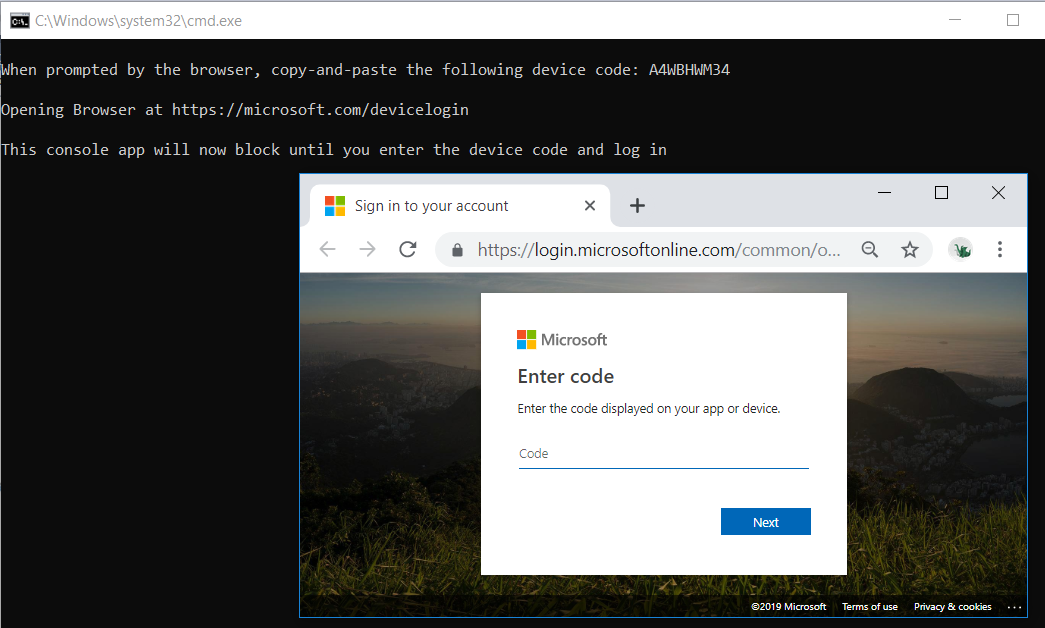
* 1. Return to the method named **DisplayAppWorkspaceAssets** and find the call to **GetAccessTokenWithUserPassword**.
  2. Replace the call to **GetAccessTokenWithUserPassword** with a call to **GetAccessTokenWithDeviceCode**.

string AccessToken = GetAccessTokenWithDeviceCode(scopesReadWorkspaceAssets);

var pbiClient = new PowerBIClient(new Uri(urlPowerBiRestApiRoot),

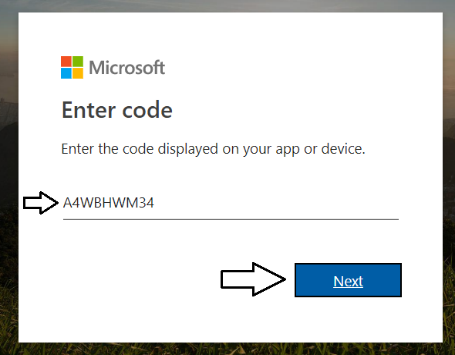
new TokenCredentials(AccessToken, "Bearer"));

1. Run the program to test the device code authentication flow.
   1. Run the console application in the Visual Studio debugger by pressing the **CTRL** + **{F5}** keyboard combination.
   2. When the program runs, it should display output in console window and open the Chrome browser at the verification URL as shown in the following screenshot.

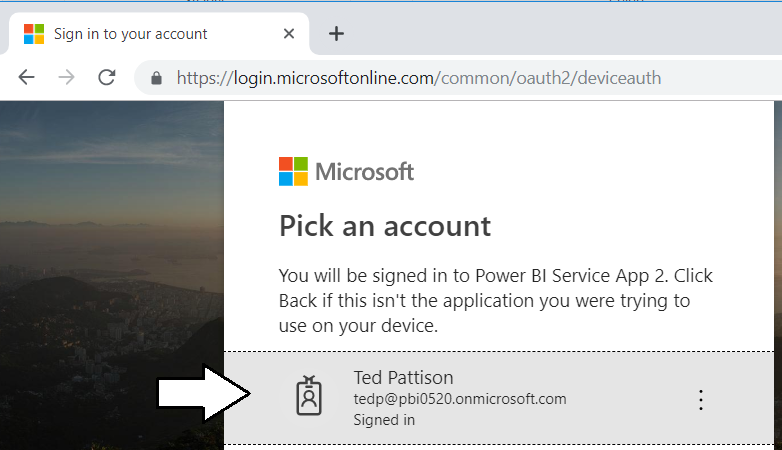


Note that the program has blocked on the call to **AcquireTokenWithDeviceCode**. The program will continue to block until you have logged in using the device code and your Office 365 user account credentials. Once you have logged in, the call to **AcquireTokenWithDeviceCode** will return and provide your application with an access token.

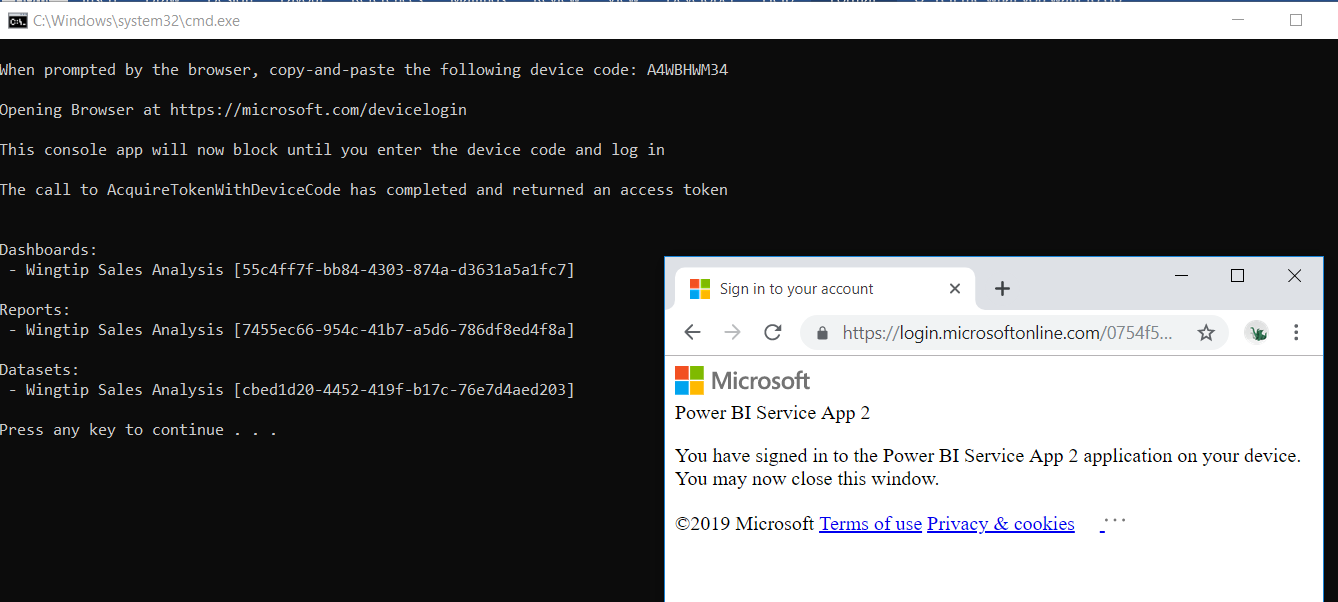
* 1. Copy the device code from the console window and paste it into the browser and then click **Next**.



* 1. When prompted, log in using the credentials for your primary Office 365 user account.



* 1. Once you have logged in, return to the console window and verify that program has completed successfully.



You have now learned how to authenticate using the *Device Code Authentication* flow.

1. Acquire an access token with every available delegated permission supported by the Power BI Service API.
   1. Return to the **PowerBiServiceApp2** project Visual Studio and make sure the **program.cs** file in open in an editor window.
   2. Inspect the static field named **scopesKitchenSink** and the delegated permission scopes defined inside.

static string[] scopesKitchenSink = new string[] {

"https://analysis.windows.net/powerbi/api/Tenant.ReadWrite.All", // requires admin

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

"https://analysis.windows.net/powerbi/api/Capacity.ReadWrite.All",

"https://analysis.windows.net/powerbi/api/Content.Create",

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

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

"https://analysis.windows.net/powerbi/api/Data.Alter\_Any",

"https://analysis.windows.net/powerbi/api/Dataflow.ReadWrite.All",

"https://analysis.windows.net/powerbi/api/Dataset.ReadWrite.All",

"https://analysis.windows.net/powerbi/api/Gateway.ReadWrite.All",

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

"https://analysis.windows.net/powerbi/api/Metadata.View\_Any",

"https://analysis.windows.net/powerbi/api/Report.ReadWrite.All",

"https://analysis.windows.net/powerbi/api/StorageAccount.ReadWrite.All",

"https://analysis.windows.net/powerbi/api/Workspace.ReadWrite.All"

};

* 1. Inspect the **DisplayAllWorkspacesInTenant** method and see how it acquires an access token using **scopesKitchenSink**.
  2. You should also notice that the **DisplayAllWorkspacesInTenant** method calls the Admin API function **GetGroupsAsAdmin**.

static void DisplayAllWorkspacesInTenant() {

string AccessToken = GetAccessTokenInteractive(scopesKitchenSink);

var pbiClient = new PowerBIClient(new Uri(urlPowerBiRestApiRoot),

new TokenCredentials(AccessToken, "Bearer"));

Console.WriteLine();

Console.WriteLine("All Workpaces in Tenant:");

var workspaces = pbiClient.Groups.GetGroupsAsAdmin(top: 100).Value;

foreach (var workspace in workspaces) {

Console.WriteLine("- " + workspace.Type + ": " + workspace.Name + " [" + workspace.Id + "] ");

}

Console.WriteLine();

}

* 1. Update the static **Main** method to call **DisplayAllWorkspacesInTenant** instead of **DisplayAppWorkspaceAssets**.

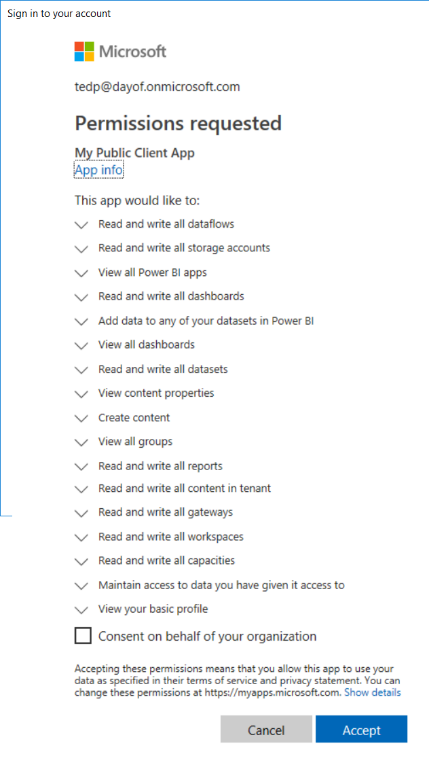
static void Main() {

// DisplayAppWorkspaceAssets();

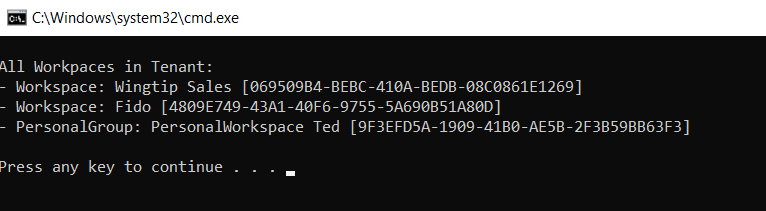
DisplayAllWorkspacesInTenant();

}

* 1. Run the console application in the Visual Studio debugger by pressing the **CTRL** + **{F5}** keyboard combination.
  2. When you sign in, you should now be prompted to consent to a humongous delegated permission set.



* 1. Click **Accept** to continue running the code for the program.
  2. The program should display a list of all the Power BI workspaces in your Azure AD tenant including personal workspaces.



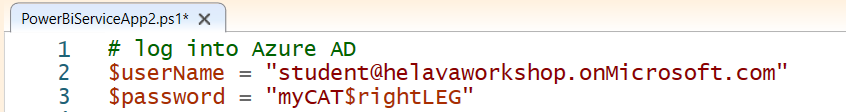
### Exercise 6: Call the Power BI Service API using an App-only Access Token

In this final exercise, you will move through the steps required to call the Power BI Service API with an app-only access token. You will begin by create a new Azure AD security group to enable calling the Power BI Service API using the identity of a service principal.

1. Use a PowerShell script to create a new Azure AD group.
   1. Open **CreateADGroupForPowerBiApps.ps1** at the following path.

C:\Student\Modules\04\_PowerBiServiceAPI\Lab\Scripts\CreateADGroupForPowerBiApps.ps1

* 1. Update the variables named **$userName** and **$password** with the credentials for your Office 365 user account.

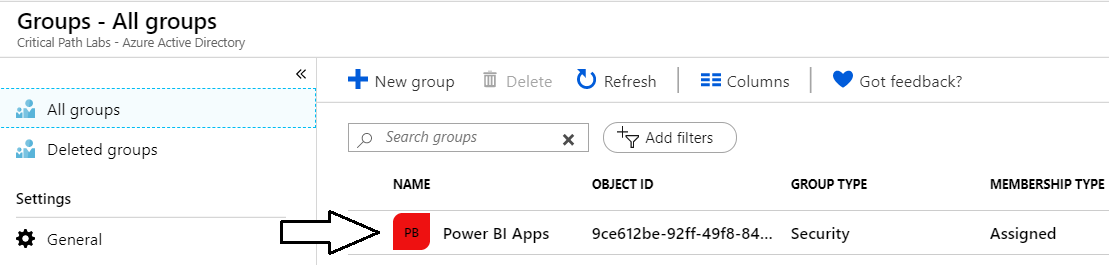


* 1. Save your changes to **CreateADGroupForPowerBiApps.ps1** and run the script.

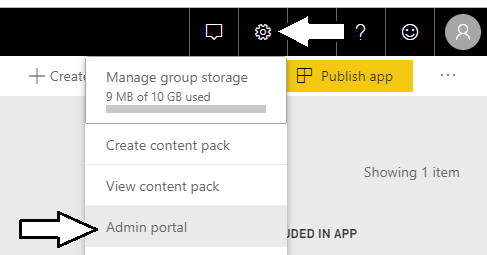
1. Use the Azure portal to verify the new Azure AD group named **Power BI Apps** has been created.
   1. Navigate to the **All groups** blade of the Azure AD portal using the following URL.

<https://portal.azure.com/#blade/Microsoft_AAD_IAM/GroupsManagementMenuBlade/AllGroups>

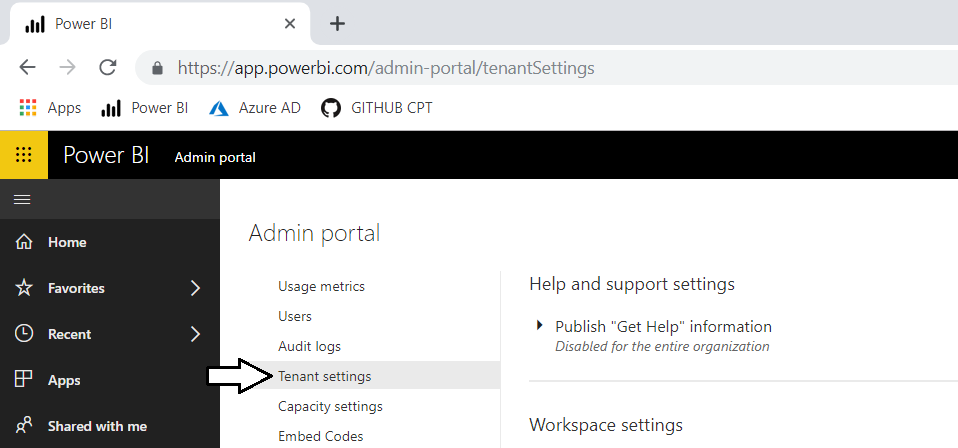
* 1. Verify that you can see the new Azure AD security group named **Power BI Apps**.



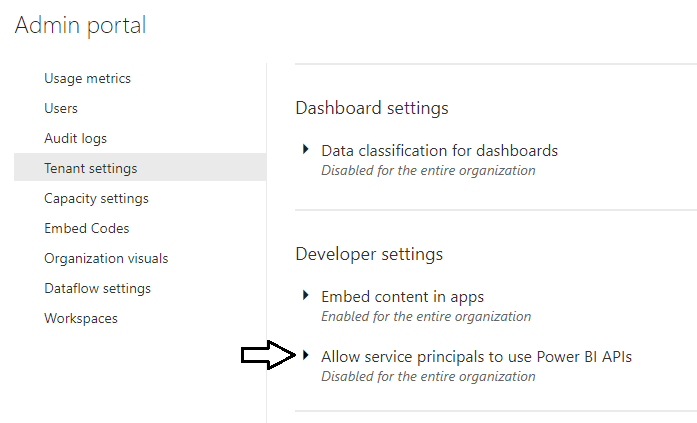
1. Enable the **Allow service principals to use Power BI APIs** setting and configure it with the **Power BI Apps** security group.
   1. Navigate to the Power BI portal at <https://app.powerbi.com>.
   2. Drop down the **Settings** menu and select the navigation command for the **Admin portal**.



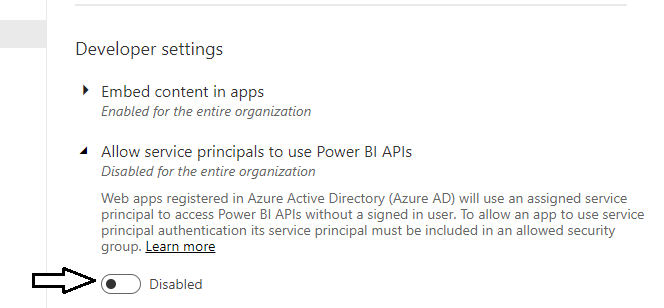
* 1. In the Power BI Admin portal, click the **Tenant settings** link on the left.



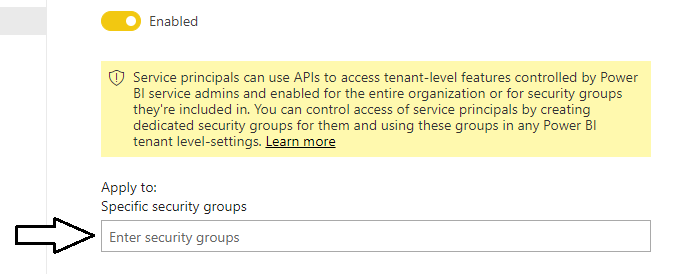
* 1. Move down in the **Developer settings** section and expand the **Allow service principals to use Power BI APIs** section.



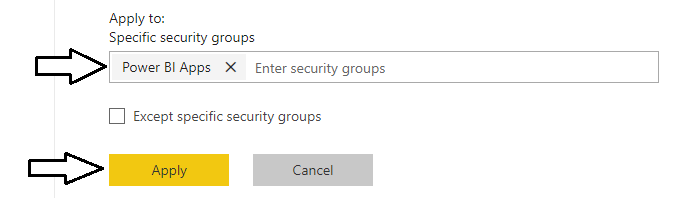
* 1. Note that the **Allow service principals to use Power BI APIs** setting is initially set to **Disabled**.



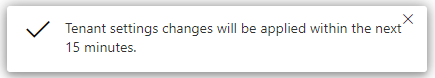
* 1. Change the setting to **Enabled** and place your cursor inside the **Apply to: Specific security groups** textbox.



* 1. Type in **Power BI Apps** to resolve the Azure AD group and then click **Apply** to save your configuration changes.



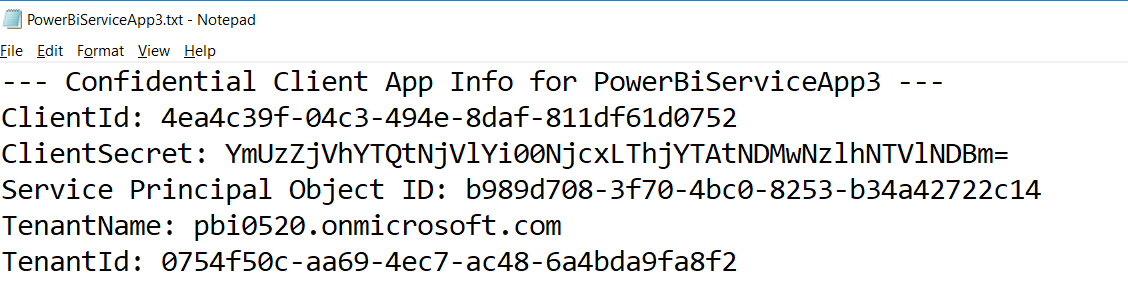
* 1. You will see a notification indicating it may take up to 15 minutes until your tenant recognizes your configuration changes.



1. Run a PowerShell script to create new confidential client application in Azure AD with a client secret.
   1. Open the PowerShell script named **RegisterPowerBiServiceApp3.ps1**.
   2. Update the two PowerShell variables named **$username** and **$userPassword**.and save your changes.
   3. Review what this script does when it runs.
      1. Connects to your tenant in Azure AD with global tenant admin permissions
      2. Generates a new GUID and uses it generate a new password credential which serves as the client secret.
      3. Creates a new confidential client application with the password credential and a display name of **Power BI Service App 3**.
      4. Configures your Office 365 user account to be an owner of the new Azure AD application.
      5. Assigns the service principal for the **Power BI Service App 3** applicationas member of the **Power BI Apps** security group.
      6. Opens a text file in Notepad with the configuration information you will need when you create your next C# Console app.

Note that this Azure AD application is being created without a Reply URL. While OAuth2 uses Reply URLs to add extra security protection for interactive authentication flows to acquire user-based access tokens, Reply URLs do not factor into app-only authentication. That means if an attacker can discover your app secret, this attackers can use it to authenticate and acquire app-only access token from anywhere on the Internet. *You have been warned!*

* 1. Run the PowerShell script named **RegisterPowerBiServiceApp3.ps1**.
  2. The script should create a new confidential application and display a text file with the info you will need in your application.

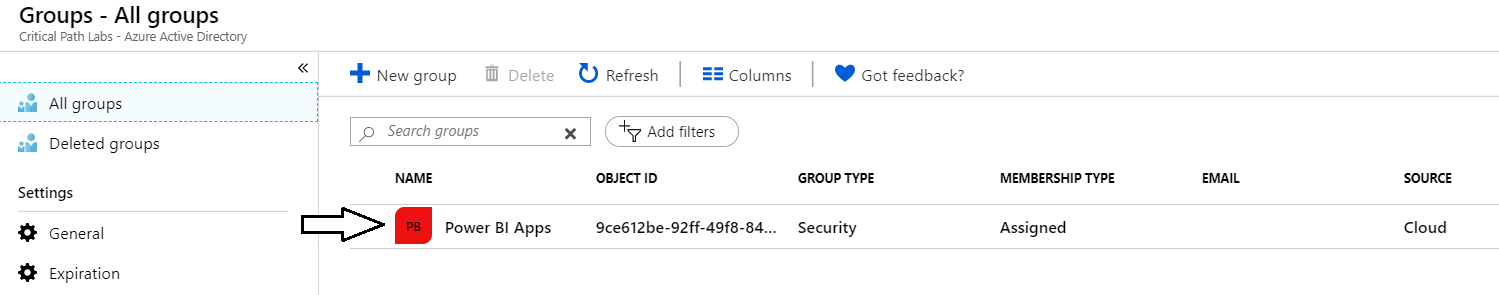


Leave this text file open as you will need to copy and paste the **ClientId** and **ClientSecret** into your application source code.

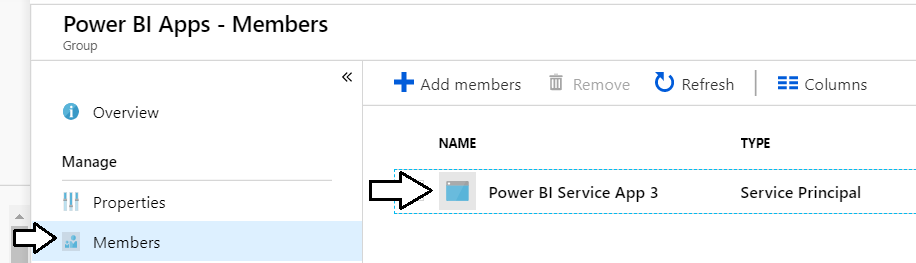
1. Inspect the group membership for **Power BI Apps** security group.
   1. Navigate to the **All groups** blade of the Azure AD portal using the following URL.

<https://portal.azure.com/#blade/Microsoft_AAD_IAM/GroupsManagementMenuBlade/AllGroups>

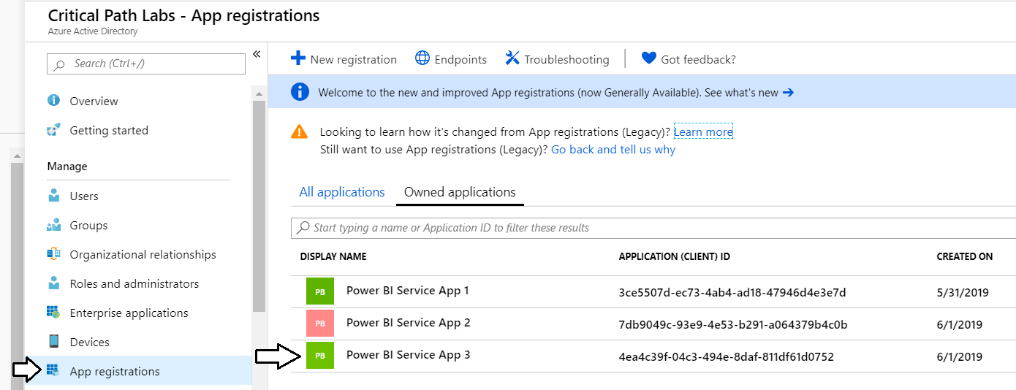
* 1. Click on the link for the **Power BI Apps** group.



* 1. Click the **Members** link on the left and verify that **Power BI Service App 3** has been added as a member.



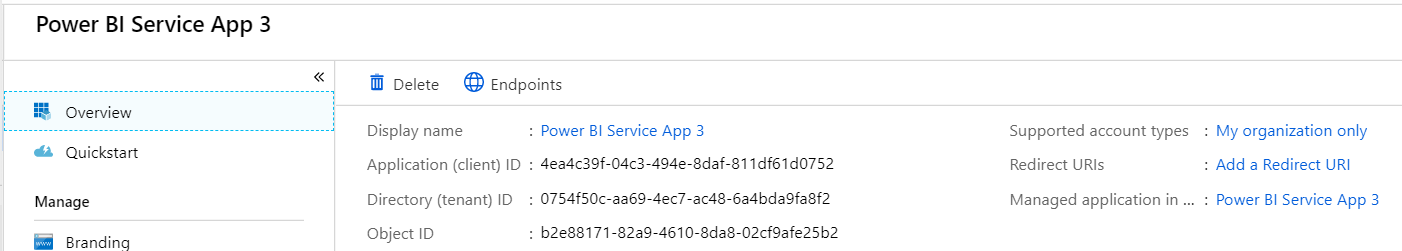
1. Inspect the confidential client application named **Power BI Service App 3** in the Azure portal
   1. Navigate to the Azure AD **App registrations** blade in the Azure portal.
   2. Locate the new Azure AD application named **Power BI Service App 3**.



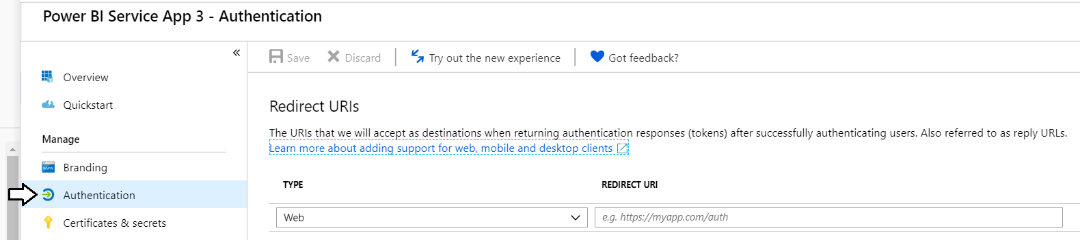
* 1. You should be able to see that **Power BI Service App 3** is configured with a client secret which is active (*e.g. Current*).



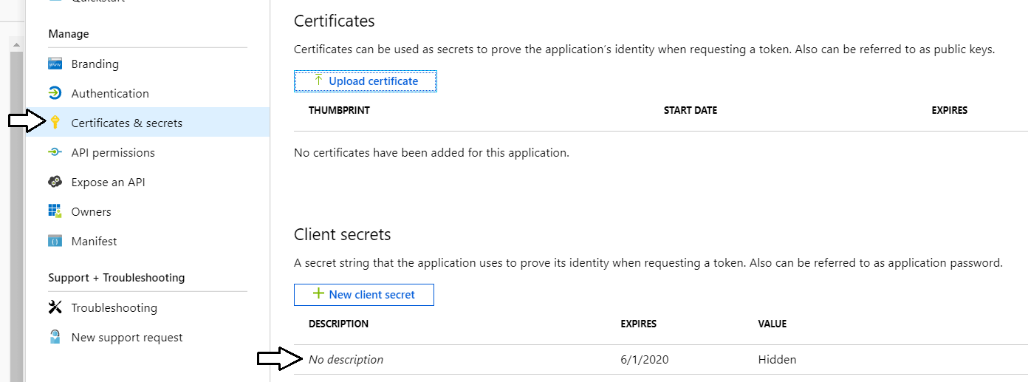
* 1. Click on the link for **Power BI Service App 3** to navigate to it summary page.



* 1. Click on the **Authentication** link on the left and verify that the application has been configured without any **Redirect URIs**.

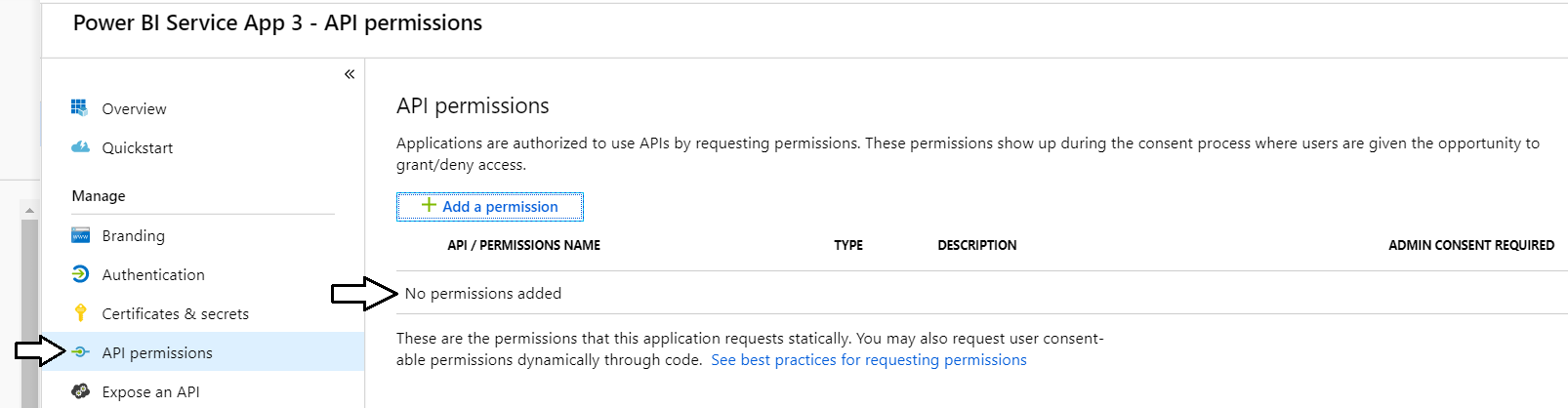


* 1. Click on the **Certificates & secrets** link and verify that the application has a client secret that expires in a year.



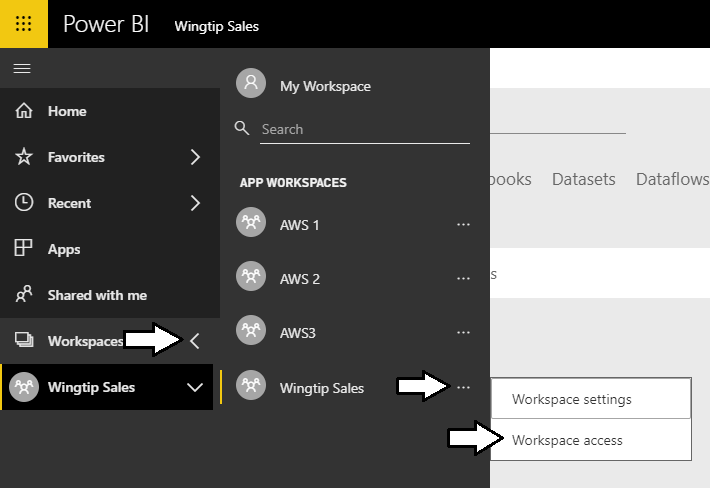
Note that it's not currently possible to assign a **Description** to a client secret created with PowerShell. However, it will still work fine.

* 1. Click on the **API permissions** link on the left and verify that the application has been configured with no **API** **permissions**.

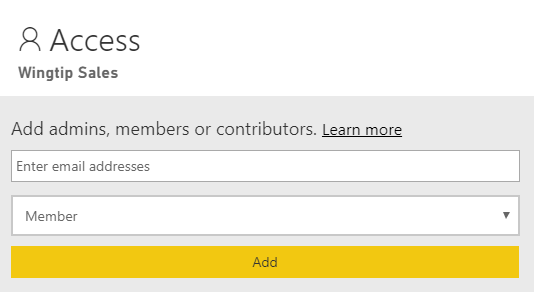


Remember that Power BI Service API permissions are not used on calls made with an app-only token. Instead, access is configured on a workspace-by-workspace basis by adding the confidential application's service principal as a workspace admin.

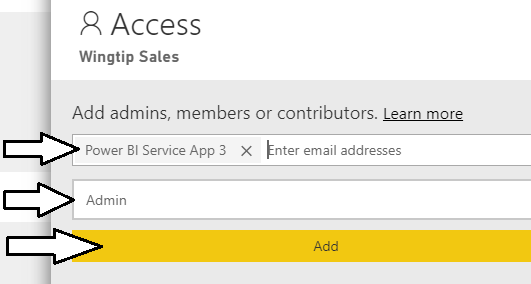
1. Configure the service principal for **Power BI Service App 3** as an admin for the app workspace named **Wingtip Sales**.
   1. Navigate to the Power BI portal.
   2. Expand the **Workspaces** flyout menu.
   3. Click the **Wingtip Sales** workspace context menu (**…**) and select **Workspace access**.



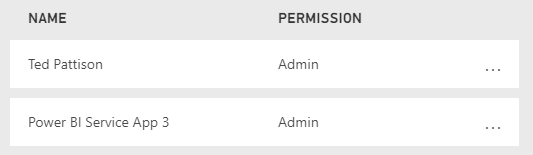
* 1. On the right of the page, you should see the **Access** pane for the **Wingtip Sales** workspace.



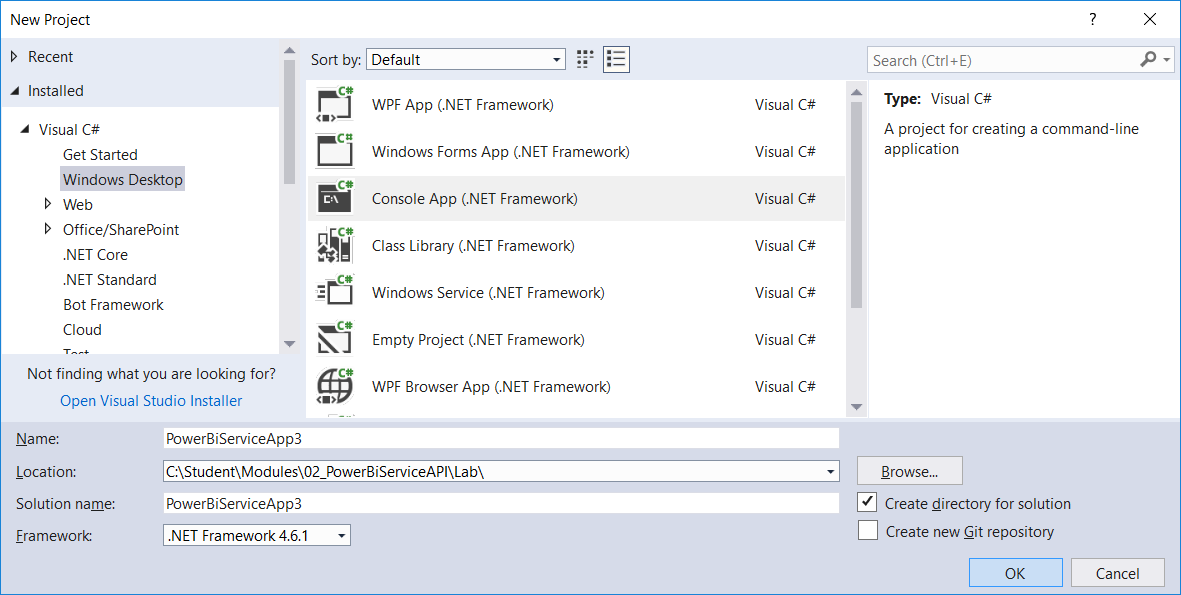
* 1. Place the cursor into the *Enter email address* textbox and type **Power BI Service App 3**.
  2. Change the member type from **Member** to **Admin**.
  3. Click to **Add** button.



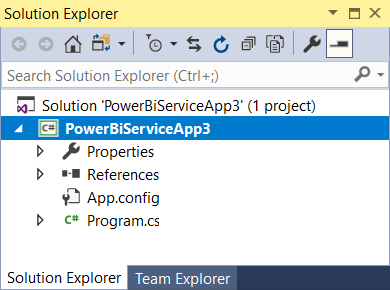
* 1. Verify that **Power BI Service App 3** has been added as a workspace member with **Admin** permissions.



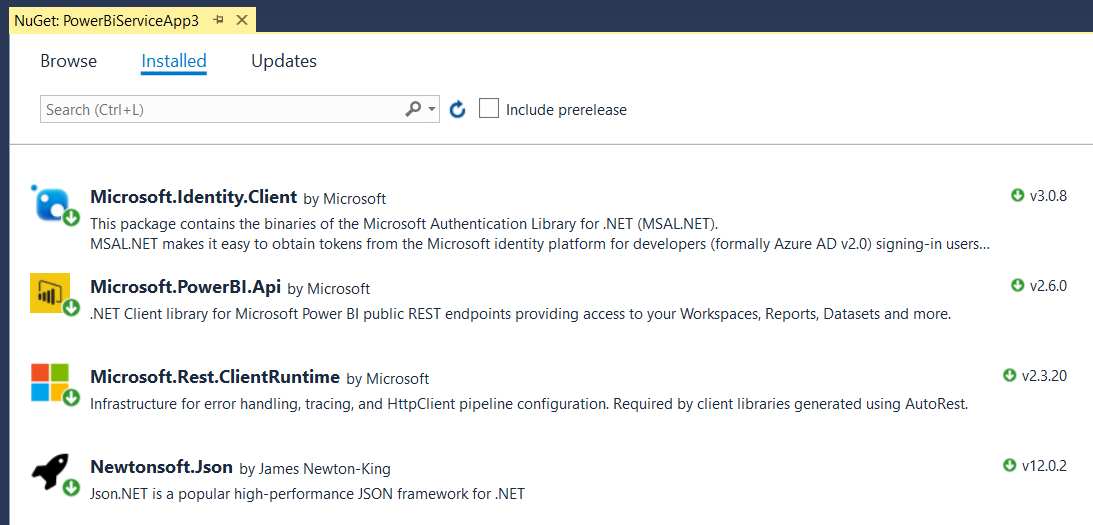
1. Create a new C# Console application in Visual Studio to access the Power BI Service API using an app-only access token.
   1. Return to Visual Studio and create a new project by running the **File > New Project** command.
   2. Select a project type of **Console App (.NET Framework)** from the **Visual C# > Windows Desktop** project templates.
   3. Give the project a **Name** of **PowerBiServiceApp3**.
   4. Give the project a **Location** of **C:\Student\Modules\04\_PowerBiServiceAPI\Lab**.
   5. Click **OK** to create the new project.



* 1. You should now have a new project named **PowerBiServiceApp3**.



1. Add the NuGet packages to the project required to program the Power BI Service API using the Power BI .NET SDK.
   1. Right-click the top-level node for the **PowerBiServiceApp3** project and select **Manage NuGet Packages…**.
   2. Locate and install the package **Microsoft.Identity.Client** for the *Microsoft Authentication library (MSAL)*.
   3. Search for and install the **Microsoft.PowerBI.Api** package to add the Power BI .NET SDK to your project.
   4. Navigate to the **Update** tab and update any packages that have updates available.
   5. Click on the **Installed** tab and ensure you have the following four packages installed.

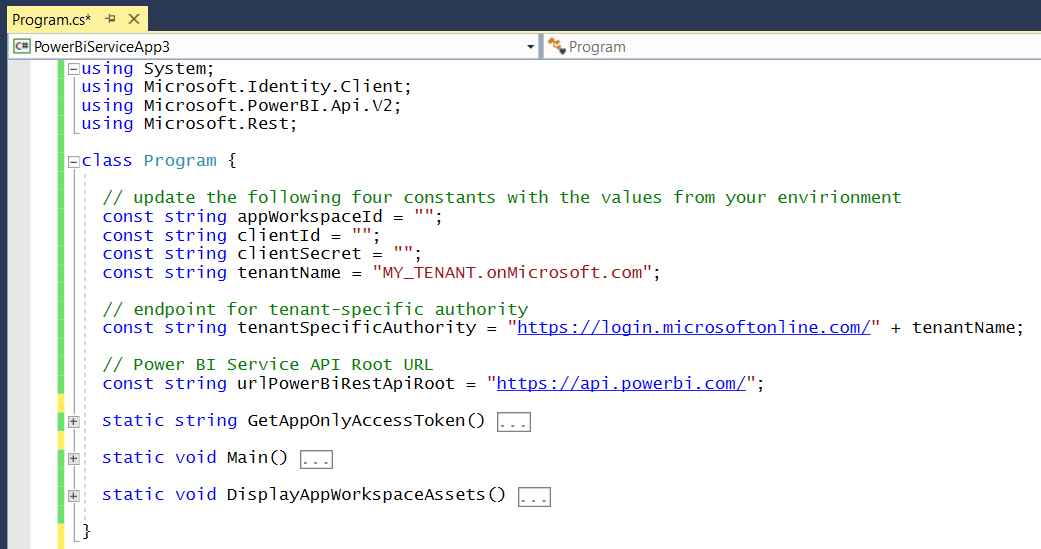


* 1. Close the window for the Nuget Package Manager.

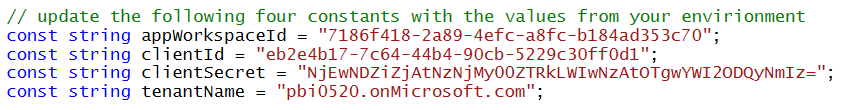
1. Add the C# starter code to **program.cs**.
   1. Using Windows Explorer, locate the file **PowerBiServiceApp3\_Starter.cs.txt** in the **Student** folder at the following path.

C:\Student\Modules\02\_PBIRestApi\Lab\StarterFiles\PowerBiServiceApp3\_Starter.cs.txt

* 1. Open the file named **PowerBiServiceApp3\_Starter.cs.txt** in Notepad and copy its contents into the Window clipboard.
  2. Return to the **PowerBiServiceApp3** project in Visual Studio.
  3. Open the source file named **program.cs**, delete all the code inside and replace it with the content in the Windows clipboard.
  4. You should now have the basic code for a simple C# console application which accesses the Power BI Service API.



* 1. Update the four constants at the top of **program.cs** with values for your environment.



* 1. Move down inside **program.cs** and examine the code inside the **GetAppOnlyAccessToken** method.

static string GetAppOnlyAccessToken() {

Console.WriteLine("Acquiring app-only access token");

var appConfidential = ConfidentialClientApplicationBuilder.Create(clientId)

.WithClientSecret(clientSecret)

.WithAuthority(tenantSpecificAuthority)

.Build();

string[] scopesDefault = new string[] { "https://analysis.windows.net/powerbi/api/.default" };

var authResult = appConfidential.AcquireTokenForClient(scopesDefault).ExecuteAsync().Result;

return authResult.AccessToken;

}

* 1. You can observe that the **DisplayAppWorkspaceAssets** method calls the **GetAppOnlyAccessToken** method.

static void DisplayAppWorkspaceAssets() {

string AccessToken = GetAppOnlyAccessToken();

var pbiClient = new PowerBIClient(new Uri(urlPowerBiRestApiRoot),

new TokenCredentials(AccessToken, "Bearer"));

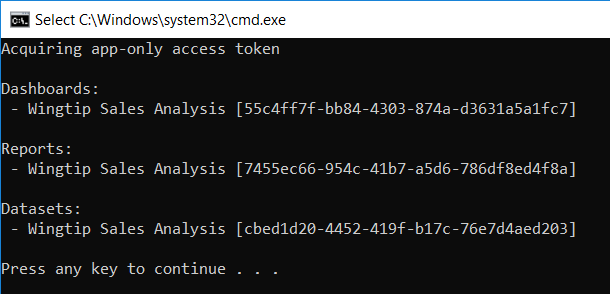
* 1. You can observe that the **Main** method calls the **DisplayAppWorkspaceAssets** method.

static void Main() {

DisplayAppWorkspaceAssets();

}

* 1. Run the app in the Visual Studio debugger by pressing the **CTRL** + **{F5}** keyboard combination.
  2. The application should run successfully and display the following output in the console window.



* 1. If you run your program and it fails, it might be due to a timing issue where Power BI has not yet synchronized the tenant-level settings for enabling service principals in the **Power BI Apps** security group to access the Power BI Service API. If that is the case, double-check that you completed all the steps in this exercise correctly and then try again every 5 minutes or so to see if your code can successfully acquire an app-only token and use it to call the Power BI Service API.

Once your program runs successfully, you are done with this lab.

You have now learned about the some of the primary differences between authenticating with ADAL versus authentication with MSAL. You also learned how to call into the Power BI API as a standard user, as an admin user and with an app-only access token. In the lab exercises for the next module you will move beyond C# console application programming and begin to program against the Power BI Service API from a web application that can be deployed to a well-known URL on the Internet.