Developing with the Power BI Service API

Setup Time: 40 minutes

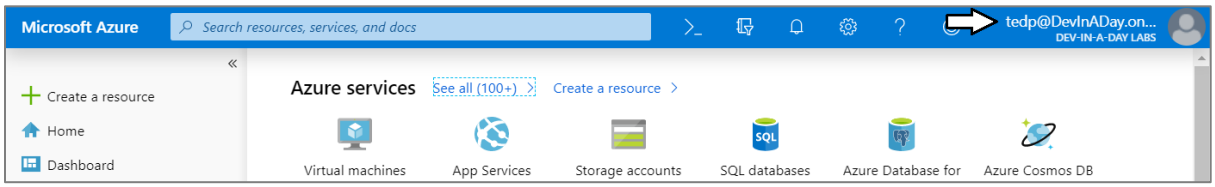
Lab Folder: C:\Student\Modules\04_PowerBiServiceApi\Lab

Overview: In this lab, you will create three different C# console applications which will allow you to experiment with the various ways in which you can authenticate and acquire access tokens from Azure Active Directory. You will learn how to register Azure AD applications by hand in the Azure portal and by using PowerShell scripts. Along the way, you will also gain experience authenticating and acquiring access tokens using both ADAL and MSAL and you will learn to program with the Power BI Service API using the Power BI .NET SDK.

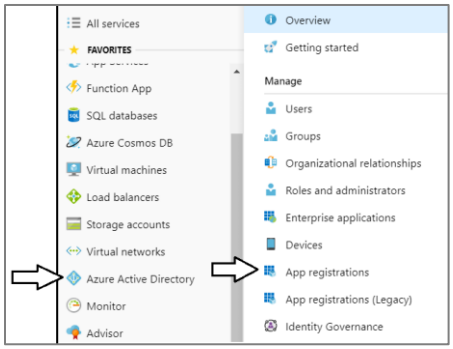
Exercise 1: Register a New Azure AD Application in the Azure Portal

In this exercise, you will create a new public 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
   a) In the browser, navigate to the Azure portal at https://portal.azure.com.
   b) When you are prompted to log in, provide the credentials to log in with your Office 365 user account name.
   c) 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.

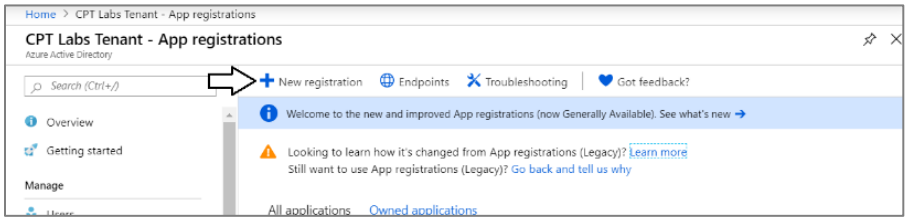


2. Register a new Azure AD application.
   a) In the left navigation, scroll down and click on the link for **Azure Active Directory**.
   b) Click the link for **App registration**.

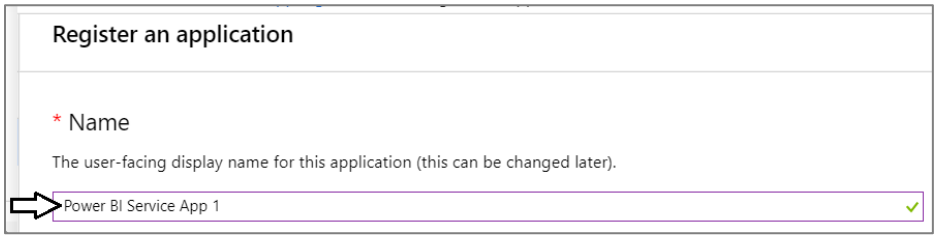


Note that the Azure portal user experience for creating and configuring Azure AD applications was updated in April 2019. If you start feeling nostalgic, you can get back to the old user experience by clicking the **App registrations (Legacy)** link.

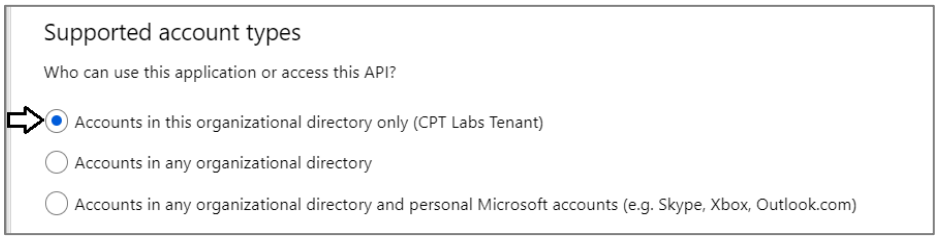
   c) Click **New registration**.



d) Enter a **Name** of **Power BI Service App 1**.



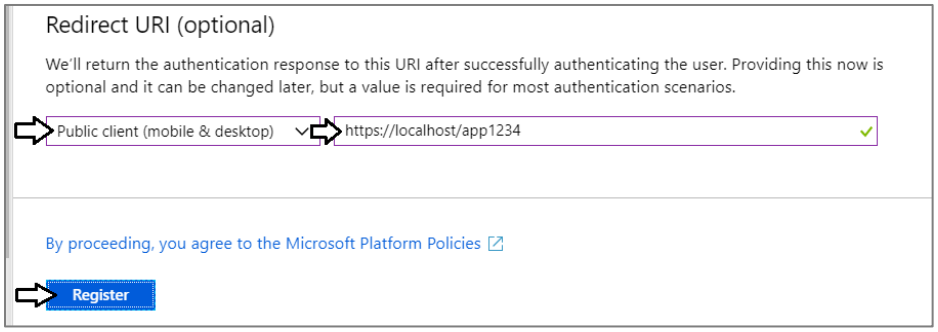
e) For the **Supported account types** option, leave the default value of **Accounts in this organizational directory only**.



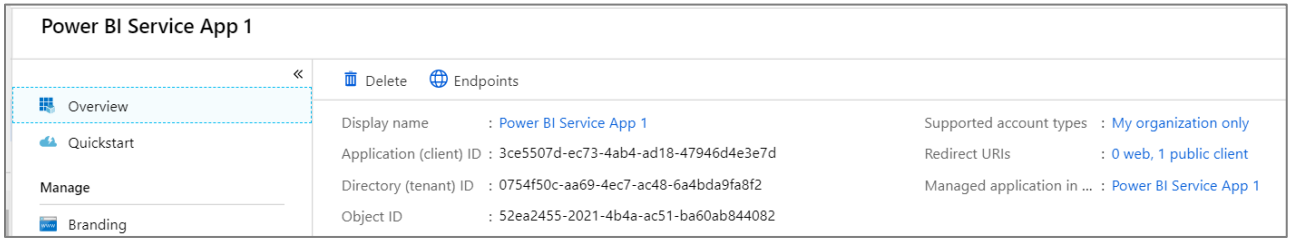
f) In the **Redirect URI** section, select **Public client (mobile or desktop)** in the left dropdown.

g) In the textbox to the right, as a **Redirect URL** of **https://localhost/app1234**.

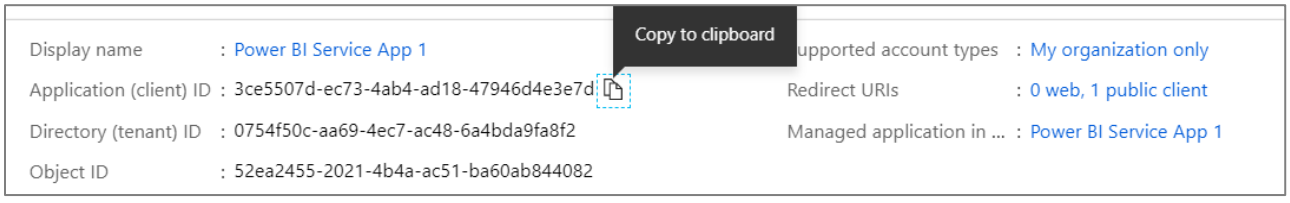
h) Click the **Register** button to create the new Azure AD application.



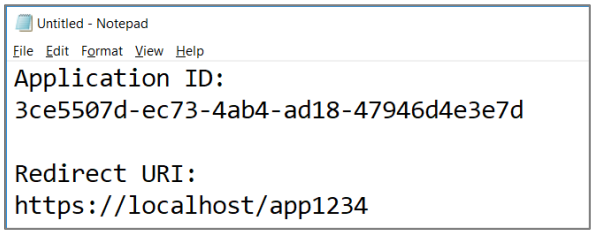
i) Once you've created the new application you should see the application summary view as shown in the following screenshot..



j) Copy the **Application ID** to the Windows clipboard.

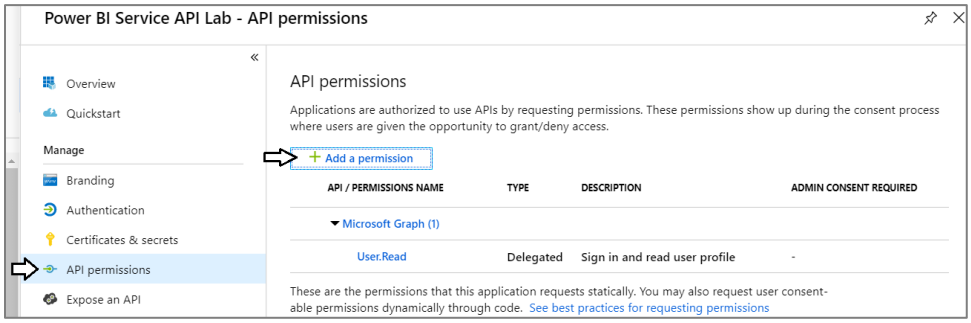


k) Launch Notepad and paste the **Application ID** into a new text file. Also add the value of the **Redirect URI**.

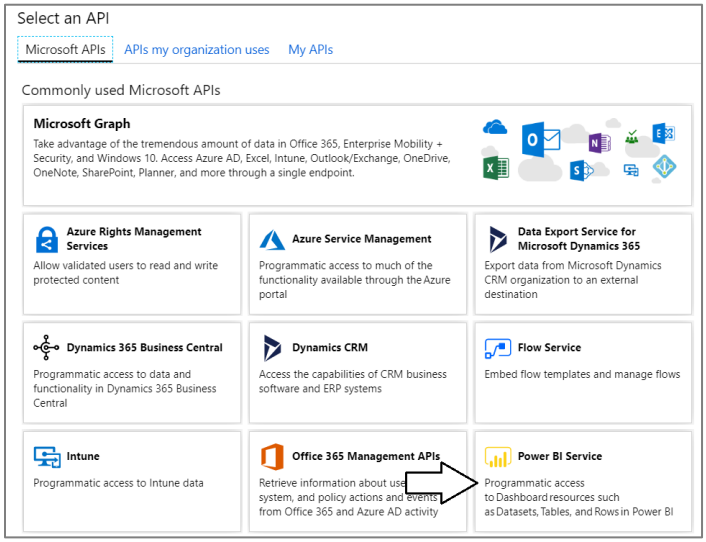


l) Click the **API permissions** link on the left.

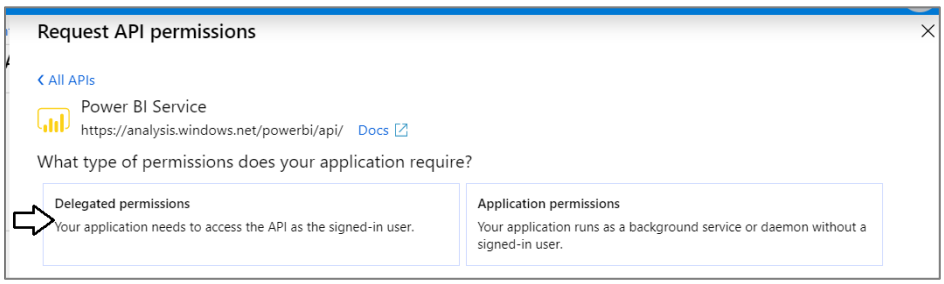
m) Click the **Add a permission** button in the **API permissions** section.



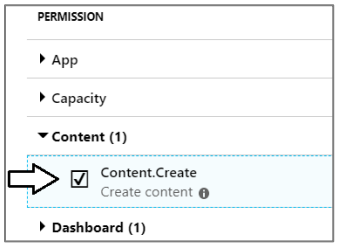
n) On the **Microsoft APIs** tab, click **Power BI Service**.



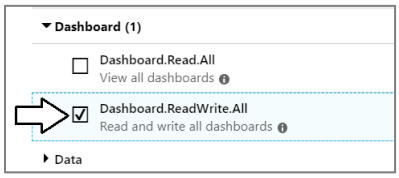
o) Click **Delegated Permissions**.



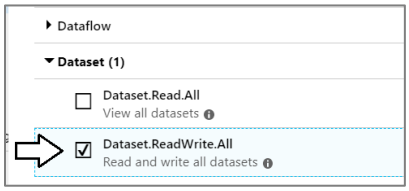
p) In the **PERMISSION** section, expand **Content** and select the **Content.Create** permission.



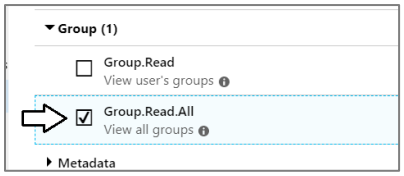
q) Expand **Dashboard** and select the **Dashboard.ReadWrite.All** permission.



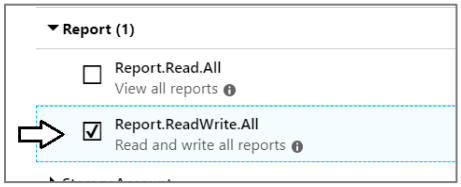
r) Expand **Dataset** and select the **Dataset.ReadWrite.All** permission.



s) Expand **Group** and select the **Group.Read.All** permission.

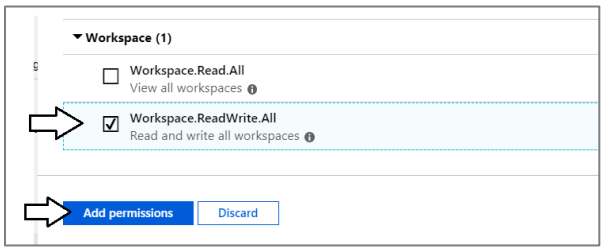


t) Expand **Report** and select the **Report.ReadWrite.All** permission.

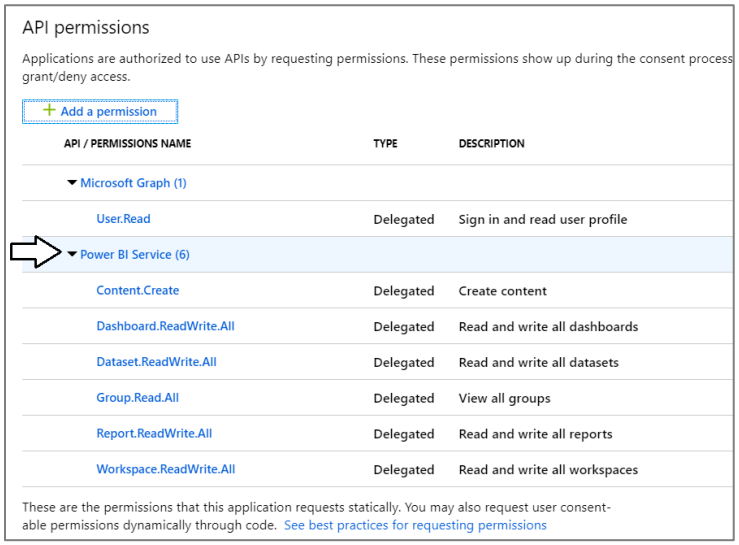


u) Expand **Workspace** and select the **Workspace.ReadWrite.All** permission.

v) Click **Add permissions** to save your changes.

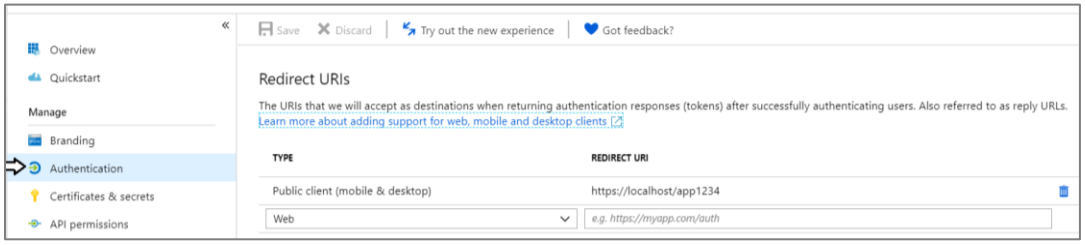


w) At this point, you should see that the Power BI Service permissions have been added to the **Required permissions** list.

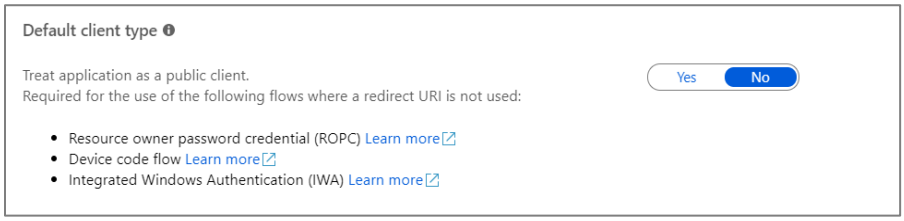


3. Change the application's **Default client type** setting to support the User Password Credential flow.

a) Click on the **Authentication** link on the left.



b) Scroll down and locate the section for the **Default client type**.



c) Update the setting for the **Default client type** to **Yes**.

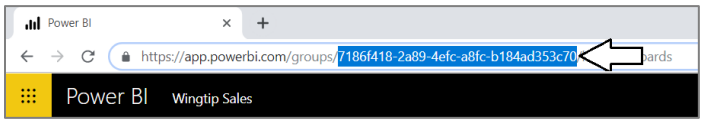
d) Click the **Save** button at the top of the page to save your changes.

You are now done registering your application with Azure AD.

Exercise 2: Call the Power BI Service API using ADAL and the Power BI .NET SDK

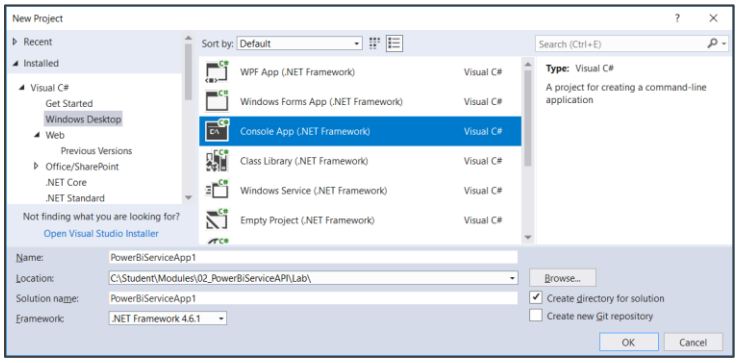
In this exercise, you will create a C# console application to authenticate with the Azure Active Directory Authentication Library (ADAL) and to call into the Power BI Service API. Before creating the Console application in Visual Studio, you will first record the GUID for the **Wingtip Sales** app workspace which will be needed later in this exercise.

1. Get the **app workspace ID** for the **Wingtip Sales** workspace.
   a) Navigate to the Power BI portal in the browser and then navigate to the **Wingtip Sales** app workspace you created in lab 1.
   b) Copy the GUID for the app workspace ID from the address bar which appears in the URL just after **groups/**.

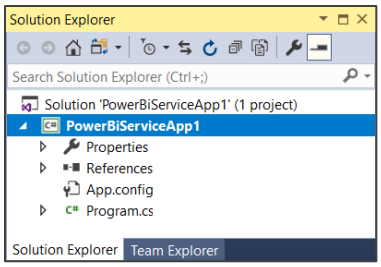


   c) Copy the **App Workspace ID** into the same text file you created earlier to hold the **Application ID** and the **Redirect URI**.

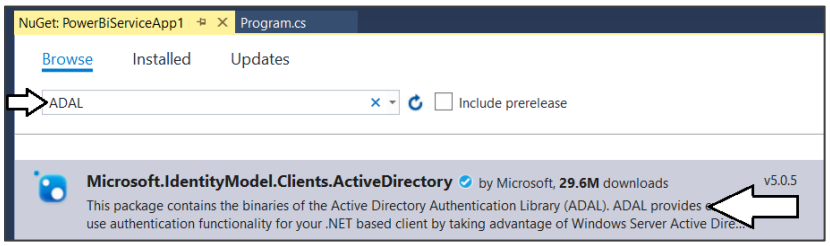
2. Create a new C# Console application in Visual Studio.
   a) Launch Visual Studio.
   b) Create a new project by running the **File > New Project** command.
   c) Select a project type of **Console App (.NET Framework)** from the **Visual C# > Windows Desktop** project templates.
   d) Give the project a **Name** of **PowerBiServiceApp1** and
   e) Give the project a **Location** of **C:\Student\Modules\04_PowerBiServiceAPI\Lab**.
   f) Click **OK** to create the new project.



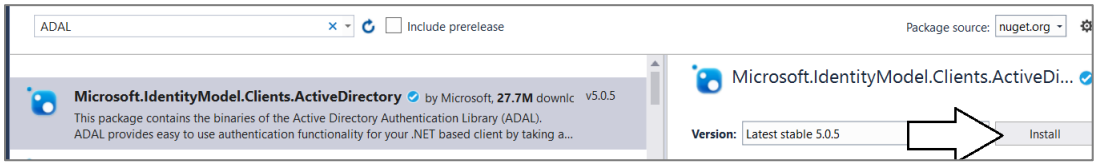
   g) You should now have a new C# console app project named **PowerBiServiceApp1**.



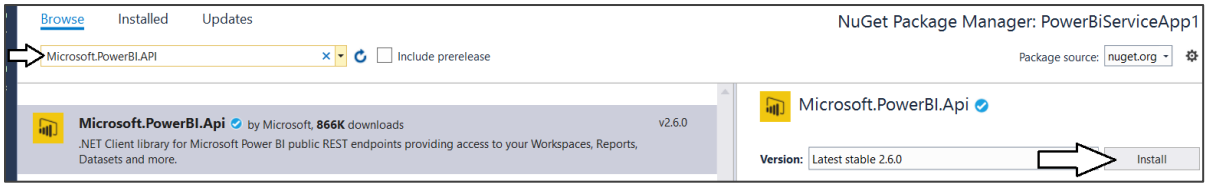
3. Add the NuGet packages to the project required to program the Power BI Service API using the Power BI .NET SDK.
   a) Right-click the top-level node for the **PowerBiServiceApp1** project and select **Manage NuGet Packages…**.
   b) Click the **Browse** tab and type **ADAL** into the search box.
   c) Locate the *Active Directly Authentication Library (ADAL)* package named **Microsoft.IdentityModel.Clients.ActiveDirectory**.



   d) Select and install **Microsoft.IdentityModel.Clients.ActiveDirectory**.

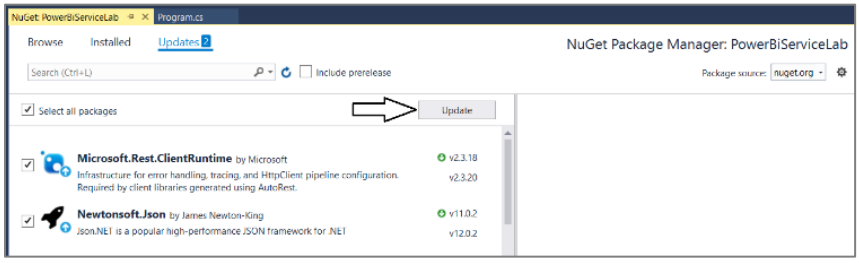


   e) When prompted about the licensing agreement, click **I Agree**.
   f) From the **Browse** tab, search for **Microsoft.PowerBI.Api** and then find and install the package **Microsoft.PowerBI.Api**.



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

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



   b) Close the window for the Nuget Package Manager.

5. Add the C# starter code to **program.cs**.
   a) Using Windows Explorer, locate the file **PowerBiServiceApp1_Starter.cs.txt** in the **Student** folder at the following path.

```
C:\Student\Modules\04_PowerBiServiceAPI\Lab\StarterFiles\PowerBiServiceApp1_Starter.cs.txt
```

   b) Open the file named **PowerBiServiceApp1_Starter.cs.txt** in Notepad and copy its contents into the Window clipboard.
   c) Return to the **PowerBiServiceApp1** project in Visual Studio.
   d) Open the source file named **program.cs**.
   e) Delete all the code inside **program.cs** and replace it with the content you copied into the Windows clipboard.

f) You should now have the basic C# code for a simple console application which accesses the Power BI Service API.

```
using System;
using System.Collections.Generic;
using System.IO;
using Microsoft.IdentityModel.Clients.ActiveDirectory;
using Microsoft.PowerBI.Api.V2;
using Microsoft.PowerBI.Api.V2.Models;
using Microsoft.Rest;

class Program {

  static string aadAuthorizationEndpoint = "https://login.windows.net/common";
  static string resourceUriPowerBi = "https://analysis.windows.net/powerbi/api";
  static string urlPowerBiRestApiRoot = "https://api.powerbi.com/";

  // enter the correct configuration values for your environment
  static string appWorkspaceId = "";
  static string clientId = "";
  static string redirectUrl = "https://localhost/app1234";

  static string GetAccessToken() ...

  static PowerBIClient GetPowerBiClient() ...

  static void Main() ...

  static void DisplayAppWorkspaceAssets() ...

}
```

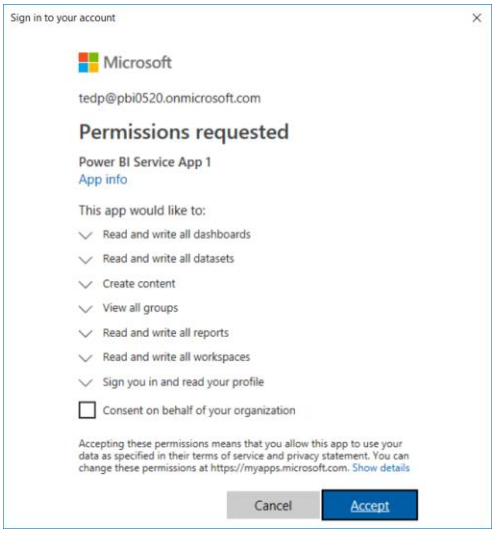
6. Update the code with your app workspace ID, the Azure AD application ID and Redirect URI.
   a) Locate the section of the code with the static properties named **appWorkspaceId**, **clientId** and **redirectUrl**.
   b) Replace these values with the values you copied into Notepad earlier.

```
static string appWorkspaceId = "7186f418-2a89-4efc-a8fc-b184ad353c70";
static string clientId = "3ce5507d-ec73-4ab4-ad18-47946d4e3e7d";
static string redirectUrl = "https://localhost/app1234";
```

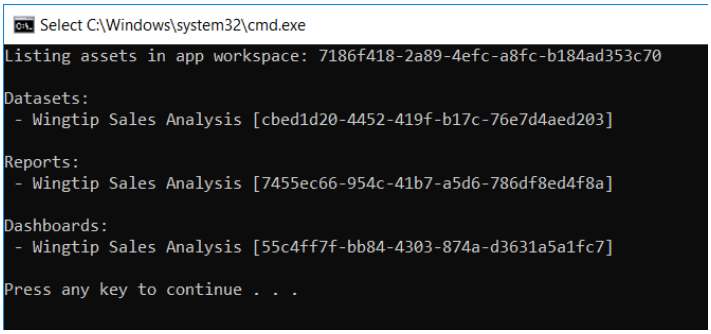
Always remember that **Application ID** and **Client ID** are just two different names for the same thing. There is a grand tradition in Azure AD development circles where we keep randomly switching back and forth between calling this identifier the application ID, the app ID and the client ID. We do this mainly to keep new developers and developer wanna-bees in a perpetual state of confusion. 😊

   c) Save your changes to **program.cs**.

7. Run the application to call to the Power BI Service API.
   a) Run the console application in the Visual Studio debugger by pressing the **CTRL** + **{F5}** keyboard combination.
   b) When promoted to sign in, log in using your Office 365 user account credentials.
   c) When prompted with the **Required permissions** dialog, click **Accept**.



d) The application should run and call into the Power BI Service API to retrieve data about the contents of the app workspace.



Since you will be running this program quite a few times as you write more code, it will make development less tedious if you modify the **GetAccessToken** method so it can run in an unattended fashion without requiring you to sign in interactively.

8. Modify the **GetAccessToken** method to acquire access tokens using the *User Password Credential* flow.

a) The following code listing shows the current implementation of the **GetAccessToken** method.

```
static string GetAccessToken() {

  // create new authentication context
  var authenticationContext = new AuthenticationContext(aadAuthorizationEndpoint);

  // use authentication context to trigger user sign-in and return access token
  var promptBehavior = new PlatformParameters(PromptBehavior.SelectAccount);
  var userAuthnResult = authenticationContext.AcquireTokenAsync(resourceUriPowerBi,
                                                                 clientId,
                                                                 new Uri(redirectUrl),
                                                                 promptBehavior).Result;

  // return access token to caller
  return userAuthnResult.AccessToken;
}
```

b) Replace the code in **GetAccessToken** with the following code which implements the *User Password Credential* flow.

```
static string GetAccessToken() {

  // create new authentication context
  var authenticationContext = new AuthenticationContext(aadAuthorizationEndpoint);

  // use authentication context to sign-in using User Password Credentials flow
  string masterUserAccount = "ACCOUNT_NAME_OF_MASTER_USER";
  string masterUserPassword = "PASSWORD_OF_MASTER_USER";
  UserPasswordCredential creds = new UserPasswordCredential(masterUserAccount, masterUserPassword);

  var userAuthnResult =
    authenticationContext.AcquireTokenAsync(resourceUriPowerBi, clientId, creds).Result;

  // return access token to caller
  return userAuthnResult.AccessToken;
}
```

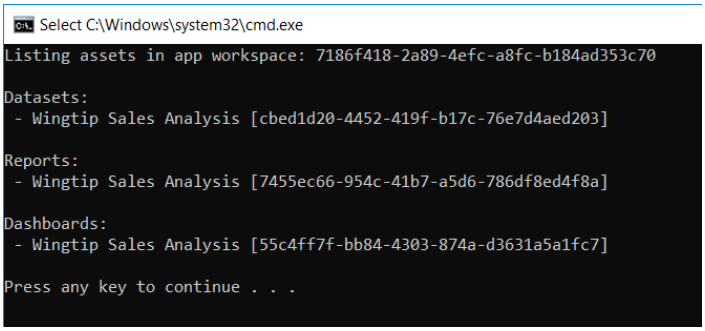
Note that the new implementation of **GetAccessToken** using the User Password Credential Flow does not use the **Redirect URI**.

c) Update the variables **masterUserAccount** and **masterUserPassword** with the credentials for your Office 365 account.

```
// use authentication context to sign-in using User Password Credentials flow
string masterUserAccount = "student@portlandembed.onMicrosoft.com";
string masterUserPassword = "pass@word1";
UserPasswordCredential creds = new UserPasswordCredential(masterUserAccount, masterUserPassword);
```

d) Save your changes to **program.cs**.

9. Run the application to call to the Power BI Service API.
   a) Run the console application in the Visual Studio debugger by pressing the **CTRL** + **{F5}** keyboard combination.
   b) The program should run as it did before but it should no longer require you to interactively enter a user name and password.



Note that **User Password Credential** flow would fail if you had not set the default client type to treat the application as a public client.

Exercise 3: Write C# Code to Create an App Workspace and Upload a PBIX Project File

In this exercise, you will update your C# console application to create new app workspaces and to publish PBIX project files.

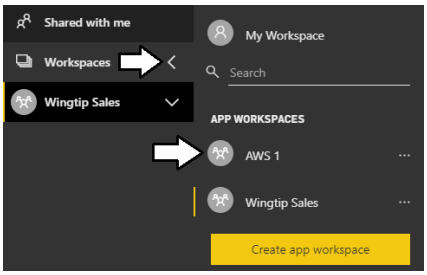
10. Add the code required to create a new app workspace.
    a) Add the static **CreateAppWorkspace** method to the bottom of the **Program** class in **program.cs**.

```
static string CreateAppWorkspace(string Name) {
  PowerBIClient pbiClient = GetPowerBiClient();
  // create new app workspace
  GroupCreationRequest request = new GroupCreationRequest(Name);
  Group aws = pbiClient.Groups.CreateGroup(request,workspaceV2: true);
  // return app workspace ID
  return aws.Id;
}
```

    b) Update the **Main** method to match the following code.

```
static void Main() {

  // DisplayAppWorkspaceAssets();
  CreateAppWorkspace("AWS 1");

}
```

11. Run the application to call to the Power BI Service API.
    a) Run the console application in the Visual Studio debugger by pressing the **CTRL** + **{F5}** keyboard combination.
    b) The program should run without any errors.
    c) After the program runs, you should be able to confirm that it created a new app workspace named **AWS 1**.



You'll have to refresh the Power BI portal page before you can see the new app workspace **AWS1** in the **Workspaces** flyout menu.

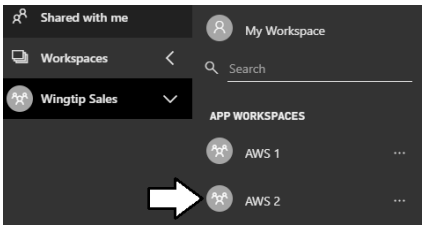
12. Add the code required to publish a PBIX project file to an app workspace.
    a) Add the static **PublishPBIX** method to the bottom of the **Program** class in **program.cs**.

```
static void PublishPBIX(string appWorkspaceId, string PbixFilePath, string ImportName) {
  Console.WriteLine("Publishing " + PbixFilePath);
  PowerBIClient pbiClient = GetPowerBiClient();
  FileStream stream = new FileStream(PbixFilePath, FileMode.Open, FileAccess.Read);
  var import = pbiClient.Imports.PostImportWithFileInGroup(appWorkspaceId, stream, ImportName);
  Console.WriteLine("Publishing process completed");
}
```

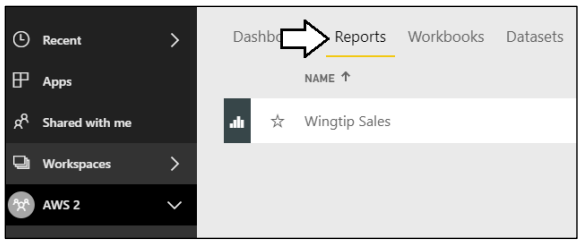
    b) Update the **Main** method to match the following code which uploads a PBIX file with an **Import** name of **Wingtip Sales**.

```
static void Main() {
  // DisplayAppWorkspaceAssets();
  // CreateAppWorkspace("AWS 1");
  string appWorkspaceId = CreateAppWorkspace("AWS 2");
  string pbixPath = @"C:\Student\PBIX\Wingtip Sales Analysis.pbix";
  string importName = "Wingtip Sales";
  PublishPBIX(appWorkspaceId, pbixPath, importName);
}
```

13. Run the application to call to the Power BI Service API.
    a) Press the **{F5}** key to begin a debugging session.
    b) The program should run without any errors.
    c) After the program runs, you should be able to confirm that it created a new app workspace named **AWS 2**.



    d) Navigate the **AWS 2** workspace and click the **Reports** tab.
    e) You should be able to verify that a report exists with the same **Import** name of **Wingtip Sales**.



Exercise 4: Write C# Code to Clone Power BI Content Across Workspaces

In this exercise, you will copy-and-paste a large piece of C# code for the **CloneAppWorkspace** method that clones content from a source app workspace to a target app workspace. Then you will test the code to make sure it works in your environment.

14. Copy and paste the code for the **CloneAppWorkspace** method.
    a) Using Windows Explorer, locate the file named **CloneAppWorkspace.cs.txt** in the **Student** folder at the following path.

```
C:\Student\Modules\02_PBIRestApi\Lab\StarterFiles\CloneAppWorkspace.cs.txt
```

    b) Open the file named **CloneAppWorkspace.cs.txt** in Notepad and copy its contents into the Window clipboard.
    c) Return to the **PowerBiServiceApp1** project in Visual Studio and the source file named **program.cs**.
    d) Place your inside the **Program** class at the bottom and paste in the content you copied into the Windows clipboard.

15. Take a moment to review the code inside **CloneAppWorkspace**.

a) The code begins by determining whether the source app workspace and target app workspace exist.

```
static void CloneAppWorkspace(string sourceAppWorkspaceName, string targetAppWorkpaceName) {

PowerBIClient pbiClient = GetPowerBiClient();
string sourceAppWorkspaceId = "";
string targetAppWorkspaceId = "";

var workspaces = pbiClient.Groups.GetGroups().Value;
foreach (var workspace in workspaces) {
  if (workspace.Name.Equals(sourceAppWorkspaceName)) {
    sourceAppWorkspaceId = workspace.Id;
  }
  if (workspace.Name.Equals(targetAppWorkpaceName)) {
    targetAppWorkspaceId = workspace.Id;
  }
}

if (sourceAppWorkspaceId == "") {
  throw new ApplicationException("Source Workspace does not exist");
}

if (targetAppWorkspaceId == "") {
  // create target app workspace if it doesn't exist
  Console.WriteLine("Creating app workspace named " + targetAppWorkpaceName);
  Console.WriteLine();
  GroupCreationRequest request = new GroupCreationRequest(targetAppWorkpaceName);
  Group AppWorkspace = pbiClient.Groups.CreateGroup(request);
  targetAppWorkspaceId = AppWorkspace.Id;
}
```

b) Next, the code exports PBIX files to clone the datasets and reports in the target workspace.

```
var reports = pbiClient.Reports.GetReportsInGroup(sourceAppWorkspaceId).Value;

string downloadPath = @"C:\Student\downloads\";

// create download folder if it doesn't exist
if (!Directory.Exists(downloadPath)) {
  Directory.CreateDirectory(downloadPath);
}

foreach (var report in reports) {

  var reportStream = pbiClient.Reports.ExportReportInGroup(sourceAppWorkspaceId, report.Id);
  string filePath = downloadPath + report.Name + ".pbix";
  Console.WriteLine("Downloading PBIX file for " + report.Name + "to " + filePath);
  FileStream stream1 = new FileStream(filePath, FileMode.Create, FileAccess.ReadWrite);
  reportStream.CopyToAsync(stream1).Wait();
  reportStream.Close();
  stream1.Close();
  stream1.Dispose();

  FileStream stream = new FileStream(filePath, FileMode.Open, FileAccess.Read);
  Console.WriteLine("Publishing " + filePath + " to " + targetAppWorkpaceName);
  var import = pbiClient.Imports.PostImportWithFileInGroup(targetAppWorkspaceId, stream, report.Name);

  Console.WriteLine("Deleing file " + filePath);
  stream.Close();
  stream.Dispose();
  File.Delete(filePath);

  Console.WriteLine();
}

Console.WriteLine("Export/Import process completed");
```

When this code runs, you will be able to see PBIX files created in **C:\Student\downloads** folder for a short period of time.

c) At the end of **CloneAppWorkspace**, there is code to clone dashboard tiles from one app workspace to another.

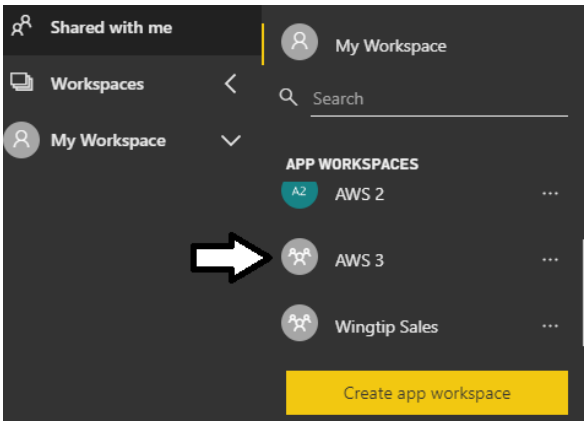
```
var dashboards = pbiClient.Dashboards.GetDashboardsInGroup(sourceAppWorkspaceId).Value;

foreach (var sourceDashboard in dashboards) {

  // create the target dashboard
  Console.WriteLine();
  Console.WriteLine("Creating Dashboard named " + sourceDashboard.DisplayName);
  AddDashboardRequest addReq = new AddDashboardRequest(sourceDashboard.DisplayName);
  Dashboard targetDashboard = pbiClient.Dashboards.AddDashboardInGroup(targetAppWorkspaceId, addReq);

  // clone tiles
  IList<Tile> sourceTiles =
              pbiClient.Dashboards.GetTilesInGroup(sourceAppWorkspaceId, sourceDashboard.Id).Value;

  foreach (Tile sourceTile in sourceTiles) {
    Console.WriteLine("Adding dashboard tile with title of " + sourceTile.Title);
    var sourceDatasetID = sourceTile.DatasetId;
    var sourceDatasetName =
        pbiClient.Datasets.GetDatasetByIdInGroup(sourceAppWorkspaceId, sourceDatasetID).Name;
    var targetWorkspaceDatasets = pbiClient.Datasets.GetDatasetsInGroup(targetAppWorkspaceId).Value;
    string targetDatasetId = "";
    foreach (var ds in targetWorkspaceDatasets) {
      if (ds.Name.Equals(sourceDatasetName)) {
        targetDatasetId = ds.Id;
      }
    }
    if (targetDatasetId.Equals("")) throw new ApplicationException("An error occured!");

    var sourceReportId = sourceTile.ReportId;
    var sourceReportName =
        pbiClient.Reports.GetReportInGroup(sourceAppWorkspaceId, sourceReportId).Name;

    var targetWorkspaceReports = pbiClient.Reports.GetReportsInGroup(targetAppWorkspaceId).Value;
    string targetReportId = "";

    foreach (var r in targetWorkspaceReports) {
      if (r.Name.Equals(sourceReportName)) {
        targetReportId = r.Id;
      }
    }

    CloneTileRequest addReqTile =
      new CloneTileRequest(targetDashboard.Id, targetAppWorkspaceId, targetReportId, targetDatasetId);

    pbiClient.Dashboards.CloneTileInGroup(sourceAppWorkspaceId,
                                          sourceDashboard.Id,
                                          sourceTile.Id,
                                          addReqTile);
  }
}
```

d) Update the **Main** method to match the following code which uploads a PBIX file with an **Import** name of **Wingtip Sales**.

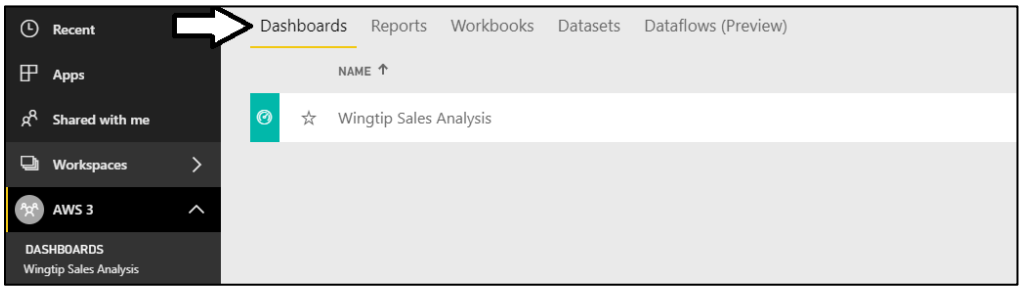
```
static void Main() {
  // DisplayAppWorkspaceAssets();
  // CreateAppWorkspace("AWS 1");
  // string appWorkspaceId = CreateAppWorkspace("AWS 2");
  // string pbixPath = @"C:\Student\PBIX\Wingtip Sales Analysis.pbix";
  // string importName = "Wingtip Sales";
  // PublishPBIX(appWorkspaceId, pbixPath, importName);

  CloneAppWorkspace("Wingtip Sales", "AWS 3");

}
```

In the following step you will run the program one more time to test your implementation of **CloneAppWorkspace**. When you test **CloneAppWorkspace**, the implementation of **CloneAppWorkspace** will clone the datasets and reports by first exporting the reports from the source workspace and then by importing the downloaded PBIX files into the target workspace.

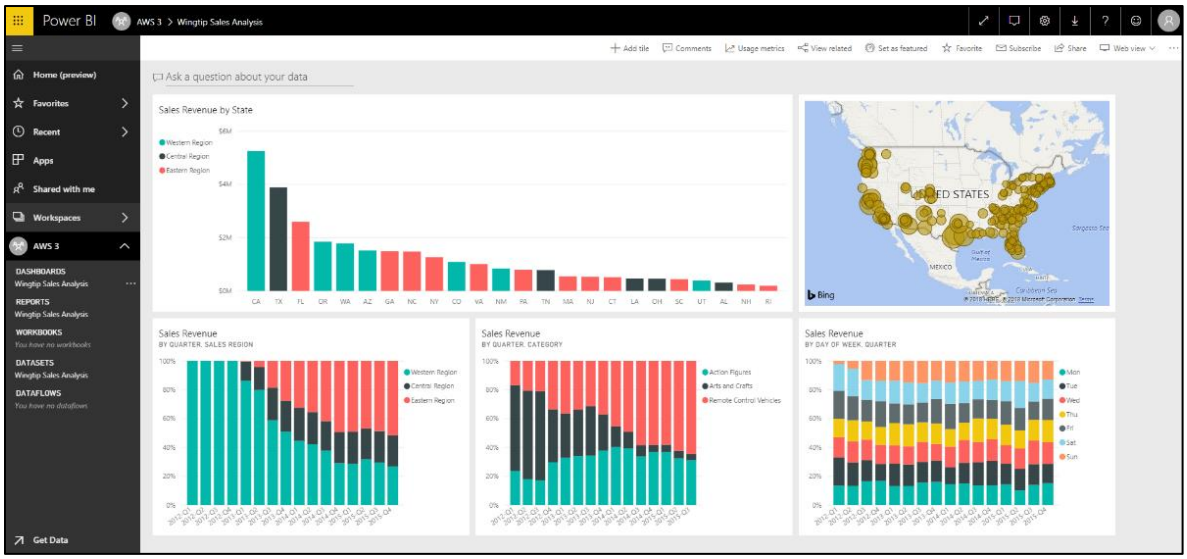
16. Run the application to call to the Power BI Service API.
    a) Run the console application in the Visual Studio debugger by pressing the **CTRL** + **{F5}** keyboard combination.
    b) The program should run without any errors.
    c) After the program runs, you should be able to confirm that it created a new app workspace named **AWS 3**.



    d) Navigate the **AWS 3** workspace and click the **Dashboards** tab.
    e) You should be able to verify that the dashboards from the **Wingtip Sales** workspace have been clones in **AWS 3**.



    f) Open the **Wingtip Sales Analysis** dashboard to verify the tiles have all been cloned correctly.



You have now successfully cloned the content in an app workspace using the Power BI Service API.

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.

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

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

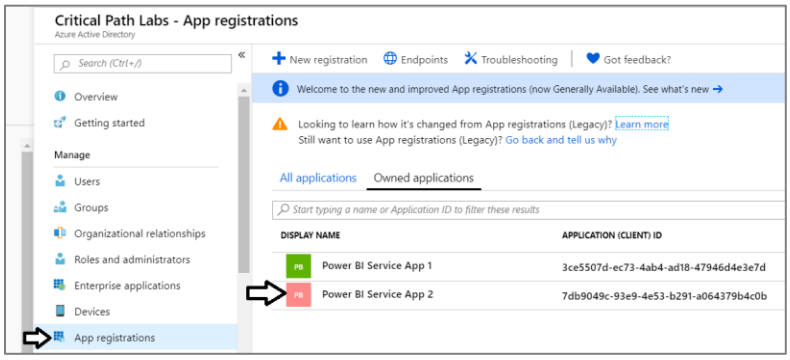
    c) Update the variables named **$userName** and **$password** with the credentials for your Office 365 user account.

```
PowerBiServiceApp2.ps1*
1  # log into Azure AD
2  $userName = "student@helavaworkshop.onMicrosoft.com"
3  $password = "myCAT$rightLEG"
```

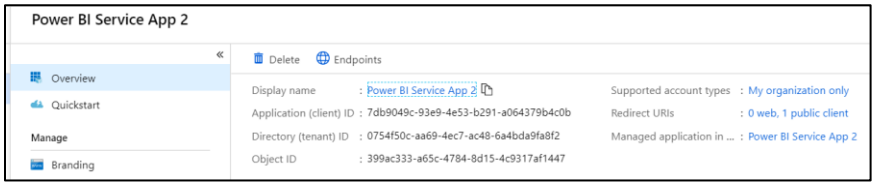
    d) Save you changes to **PowerBiServiceApp2.ps1** and run the script.
    e) When the script runs, it will create a new public client application and display application details in a text file shown in Notepad.

```
PowerBiServiceApp2.txt - Notepad
File Edit Format View Help
--- Public Client App Info for PowerBiServiceApp2 ---
ClientId: 7db9049c-93e9-4e53-b291-a064379b4c0b
ReplyUrl: https://localhost/app1234
TenantName: pbi0520.onmicrosoft.com
```

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



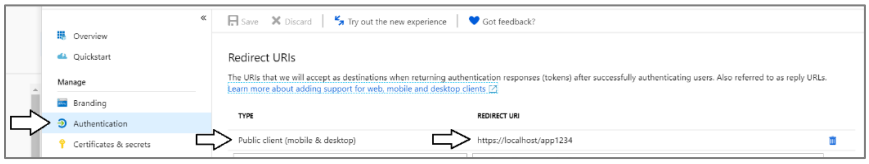
    e) You should now see the summary page for **Power BI Service App 2**.



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.

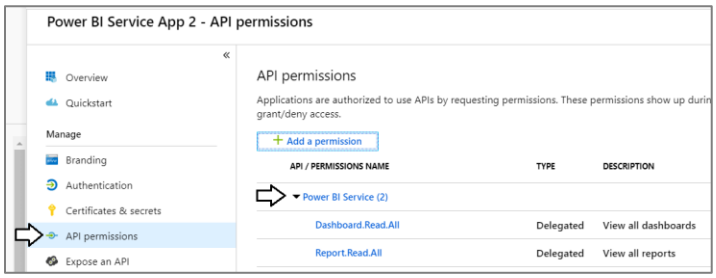
f) Click the **Authentication** link on the left.

g) You should be able to verify that the **TYPE** is set to **Public client** and **REDIRECT URI** is set to **https://localhost/app1234**.



h) Click the **API Permissions** link on the left.

i) 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).

19. Create a new C# Console application in Visual Studio.

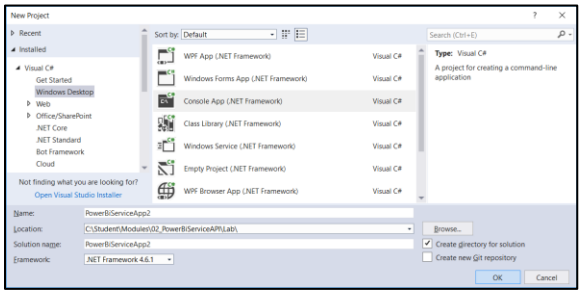
a) Launch a new instance of Visual Studio.

b) Create a new project by running the **File > New Project** command.

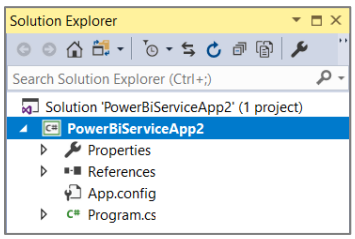
c) Select a project type of **Console App (.NET Framework)** from the **Visual C# > Windows Desktop** project templates.

d) Give the project a **Name** of **PowerBiServiceApp2**.

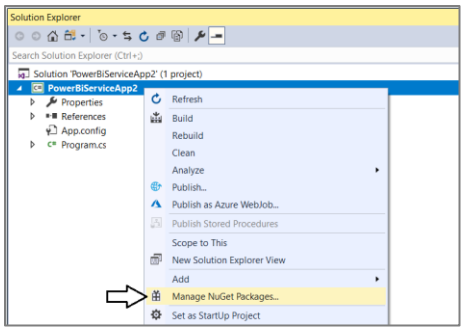
e) Give the project a **Location** of **C:\Student\Modules\04_PowerBiServiceAPI\Lab**. and click **OK** to create the new project.



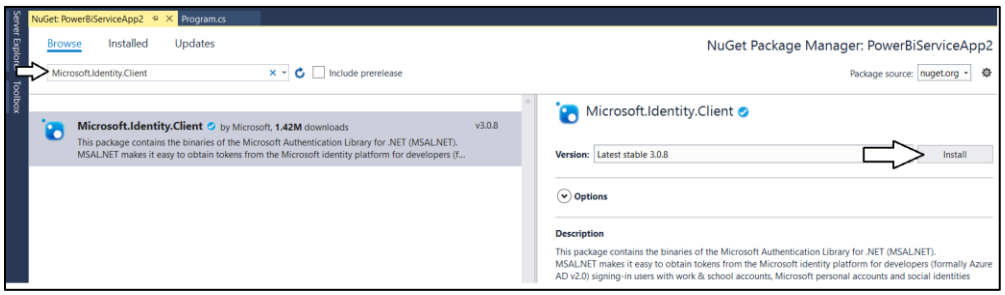
f) You should now have a new project named **PowerBiServiceApp2**.



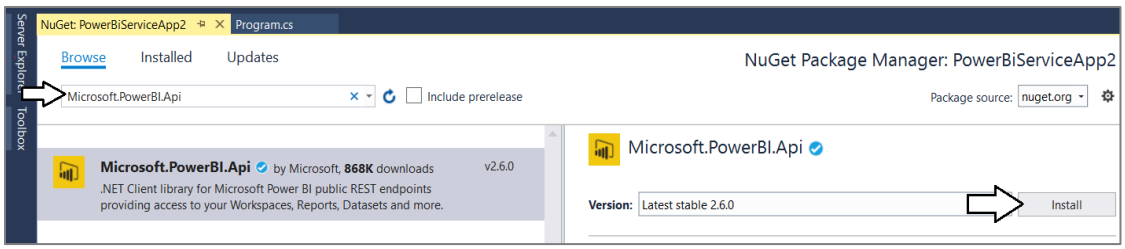
20. Add the NuGet packages to the project required to program the Power BI Service API using the Power BI .NET SDK.
    a) Right-click the top-level node for the **PowerBiServiceApp2** project and select **Manage NuGet Packages…**.



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

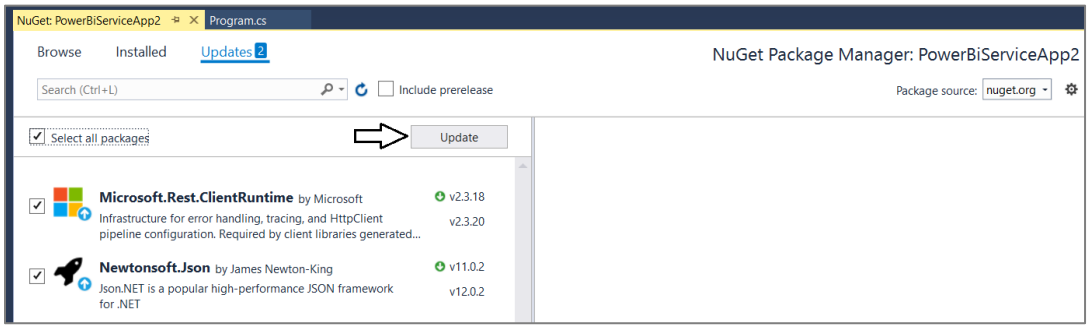


    d) If you are prompted about **Preview Changes**, click **OK**.
    e) When prompted about **License Acceptance**, click **I Agree**.
    f) Search for and install the package named **Microsoft.PowerBI.Api** to add the Power BI .NET SDK to your project.

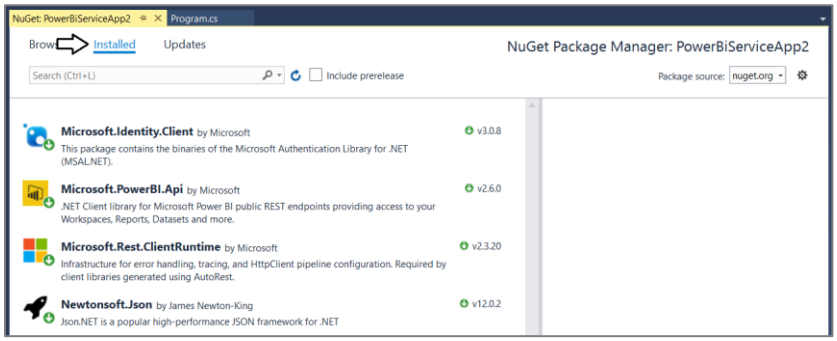


    g) When prompted about the licensing agreement, click **I Agree**.

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



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



c) Close the window for the Nuget Package Manager.

22. Add the C# starter code to **program.cs**.

a) 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
```

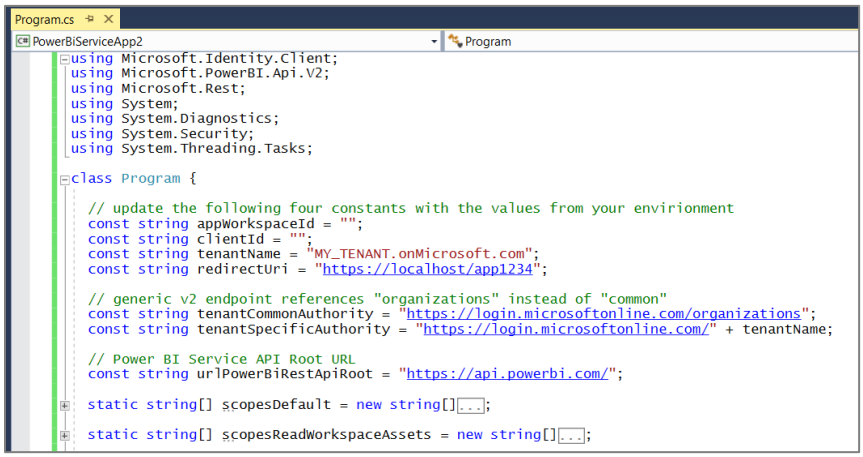
b) Open the file named **PowerBiServiceApp2_Starter.cs.txt** in Notepad and copy its contents into the Window clipboard.

c) Return to the **PowerBiServiceApp2** project in Visual Studio.

d) Open the source file named **program.cs**.

e) Delete all the code inside **program.cs** and replace it with the content you copied into the Windows clipboard.

f) You should now have the basic C# code for a simple console application which access the Power BI Service API.



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

```
// update the following four constants with the values from your envirionment
const string appWorkspaceId = "";
const string clientId = "";
const string tenantName = "MY_TENANT.onMicrosoft.com";
const string redirectUri = "https://localhost/app1234";
```

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

```
// update the following four constants with the values from your envirionment
const string appWorkspaceId = "7186f418-2a89-4efc-a8fc-b184ad353c70";
const string clientId = "7db9049c-93e9-4e53-b291-a064379b4c0b";
const string tenantName = "pbi0520.onMicrosoft.com";
const string redirectUri = "https://localhost/app1234";
```

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

```
// generic v2 endpoint references "organizations" instead of "common"
const string tenantCommonAuthority = "https://login.microsoftonline.com/organizations";
const string tenantSpecificAuthority = "https://login.microsoftonline.com/" + tenantName;

// Power BI Service API Root URL
const string urlPowerBiRestApiRoot = "https://api.powerbi.com/";

static string[] scopesDefault = new string[]...;

static string[] scopesReadWorkspaceAssets = new string[]...;

static string[] scopesReadUserApps = new string[]...;

static string[] scopesManageWorkspaceAssets = new string[]...;

static string[] scopesKitchenSink = new string[]...;
```

    c) 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;
}
```

    d) 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();
    }
```

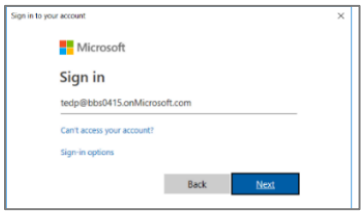
e) Code in **DisplayAppWorkspaceAssets** calls **GetAccessTokenInteractive** passing a parameter value of **scopesDefault**.

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

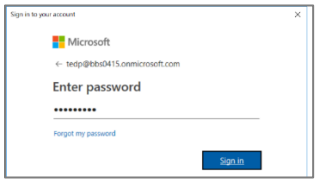
The first time you run the program, Azure AD will prompt you to consent to the default permissions configured for the application.

24. Run the application to test your work.

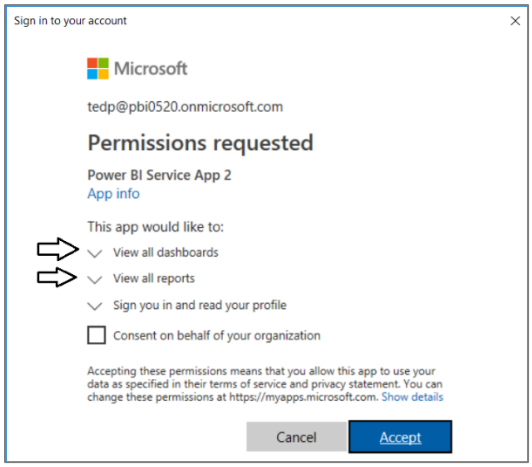
a) Run the console application in the Visual Studio debugger by pressing the **CTRL** + **{F5}** keyboard combination.



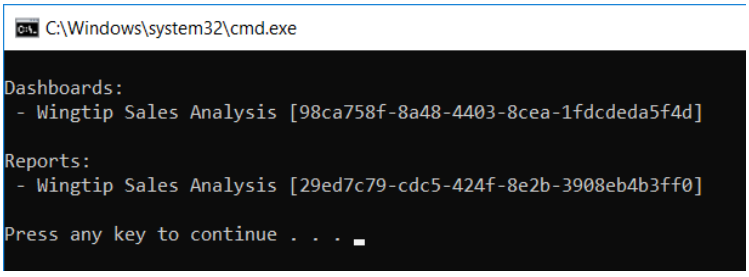
b) When prompted to sign in, enter your user name and password.



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



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



25. Try running the console application again after uncommenting the code to retrieve information about datasets

a) Locate the commented code at the bottom of the **DisplayAppWorkspaceAssets** method and uncomment it.

```
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("Datasets:");
//var datasets = pbiClient.Datasets.GetDatasetsInGroup(appWorkspaceId).Value;
//foreach (var dataset in datasets) {
//  Console.WriteLine(" - " + dataset.Name + " [" + dataset.Id + "]");
//}

Console.WriteLine();
```

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

```
    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("Datasets:");
    var datasets = pbiClient.Datasets.GetDatasetsInGroup(appWorkspaceId).Value;
    foreach (var dataset in datasets) {
      Console.WriteLine(" - " + dataset.Name + " [" + dataset.Id + "]");
    }

    Console.WriteLine();
```

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

c) Run the console application in the Visual Studio debugger by pressing the **CTRL** + **{F5}** keyboard combination.

d) When prompted, sign in.

e) The program should run and then fail with an ***Unauthorized*** exception when it attempts to call **GetDatasetsInGroup**.

```
C:\Windows\system32\cmd.exe

Dashboards:
 - Wingtip Sales Analysis [98ca758f-8a48-4403-8cea-1fdcdeda5f4d]

Reports:
 - Wingtip Sales Analysis [29ed7c79-cdc5-424f-8e2b-3908eb4b3ff0]

Datasets:

Unhandled Exception: Microsoft.Rest.HttpOperationException: Operation returned an invalid status code 'Unauthorized'
   at Microsoft.PowerBI.Api.V2.Datasets.<GetDatasetsInGroupWithHttpMessagesAsync>d__27.MoveNext()
--- End of stack trace from previous location where exception was thrown ---
   at System.Runtime.CompilerServices.TaskAwaiter.ThrowForNonSuccess(Task task)
   at System.Runtime.CompilerServices.TaskAwaiter.HandleNonSuccessAndDebuggerNotification(Task task)
```

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

26. Acquire an access token interactively with the required scopes to successfully call **DisplayAppWorkspaceAssets**.

a) 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"
};
```

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

```
static void DisplayAppWorkspaceAssets() {

  string AccessToken = GetAccessTokenInteractive(scopesDefault);
  var pbiClient = new PowerBIClient(new Uri(urlPowerBiRestApiRoot),
                                    new TokenCredentials(AccessToken, "Bearer"));
```

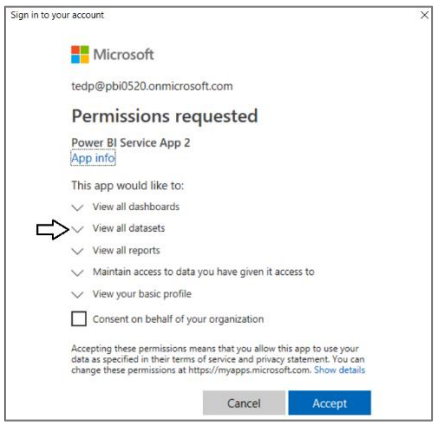
c) In the call to **DisplayAppWorkspaceAssets**, replace **scopesDefault** with **scopesReadWorkspaceAssets**.

```
static void DisplayAppWorkspaceAssets() {

  string AccessToken = GetAccessTokenInteractive(scopesReadWorkspaceAssets);
  var pbiClient = new PowerBIClient(new Uri(urlPowerBiRestApiRoot),
                                    new TokenCredentials(AccessToken, "Bearer"));
```

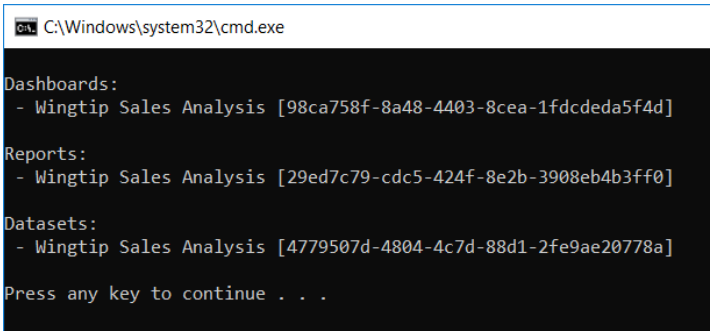
d) Run the console application again by pressing the **CTRL** + **{F5}** keyboard combination.

e) When prompted, sign in.

f) After signing in, you should be prompted to consent to permissions including **View all datasets**.



g) The program should now succeed when calling **GetDatasetsInGroup**.



27. Authenticate using *(the dreaded) User Credential Password* flow.

a) 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;
}
```

b) Locate the two variables inside **GetAccessTokenWithUserPassword** named **username** and **userPassword**.

c) 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";
```

d) 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"));
```

e) 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"));
```

f) 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*.

28. Authenticate the user using the *Device Code Authentication* flow.

a) Review the method named **GetAccessTokenWithDeviceCode**.

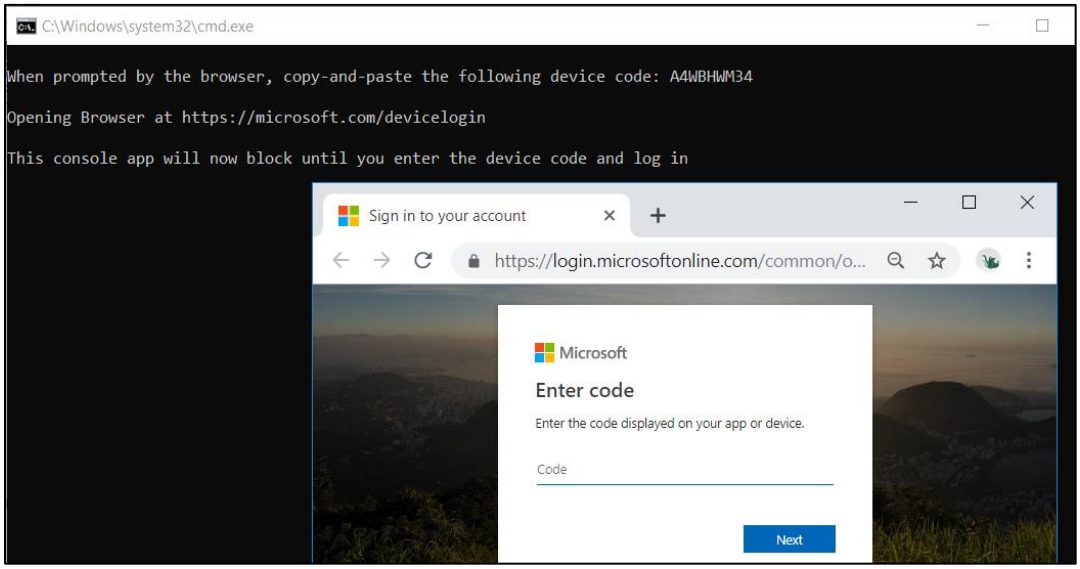
```
static string GetAccessTokenWithDeviceCode(string[] scopes) {

  // 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;
}
```

b) Return to the method named **DisplayAppWorkspaceAssets** and find the call to **GetAccessTokenWithUserPassword**.

c) Replace the call to **GetAccessTokenWithUserPassword** with a call to **GetAccessTokenWithDeviceCode**.

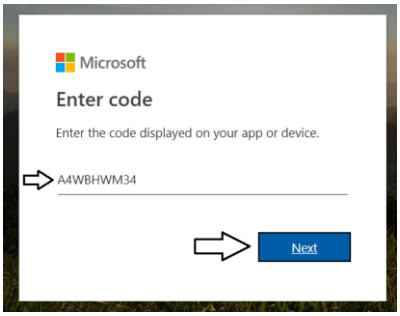
```
string AccessToken = GetAccessTokenWithDeviceCode(scopesReadWorkspaceAssets);
var pbiClient = new PowerBIClient(new Uri(urlPowerBiRestApiRoot),
                                  new TokenCredentials(AccessToken, "Bearer"));
```

29. Run the program to test the device code authentication flow.
    a) Run the console application in the Visual Studio debugger by pressing the **CTRL** + **{F5}** keyboard combination.
    b) 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.

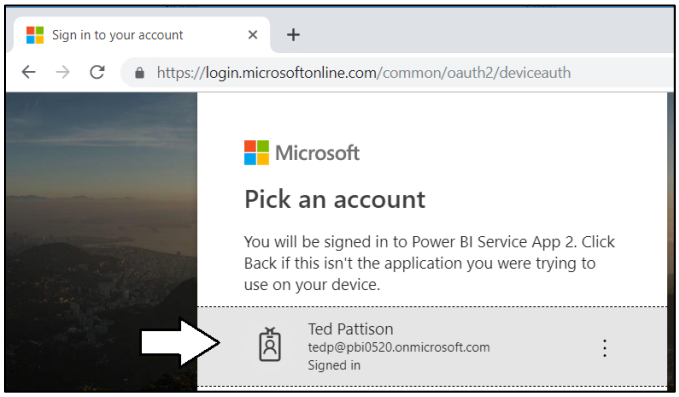


Note that the program has blocked on the call to **AcquireTokenWithDeviceCode**. The program will continue to block until you have logged in using the device code and your Office 365 user account credentials. Once you have logged in, the call to **AcquireTokenWithDeviceCode** will return and provide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

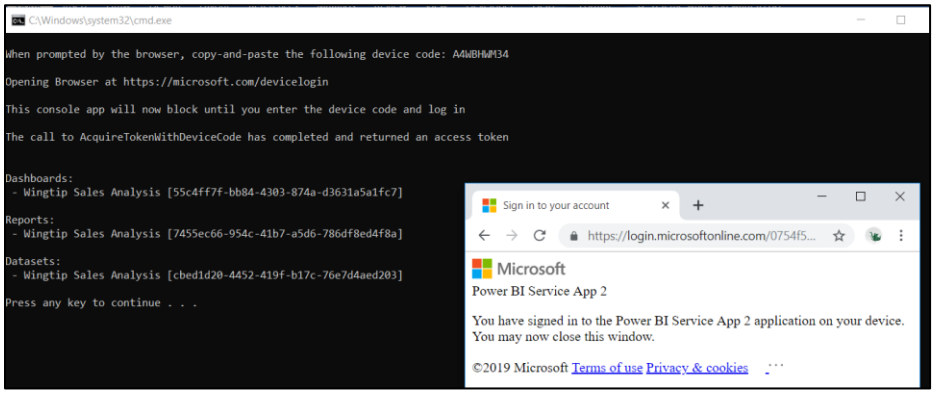
    c) Copy the device code from the console window and paste it into the browser and then click **Next**.



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



e) 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.

30. Acquire an access token with every available delegated permission supported by the Power BI Service API.

a) Return to the **PowerBiServiceApp2** project Visual Studio and make sure the **program.cs** file in open in an editor window.

b) 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"
};
```

c) Inspect the **DisplayAllWorkspacesInTenant** method and see how it acquires an access token using **scopesKitchenSink**.

d) You should also notice that the **DisplayAllWorkspacesInTenant** method calls the Admin API function **GetGroupsAsAdmin**.

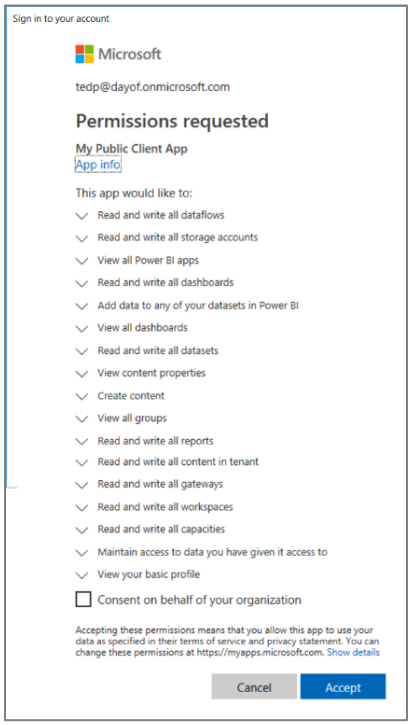
```
static void DisplayAllWorkspacesInTenant() {

  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();

}
```

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

```
  static void Main() {
    // DisplayAppWorkspaceAssets();
    DisplayAllWorkspacesInTenant();
  }
```

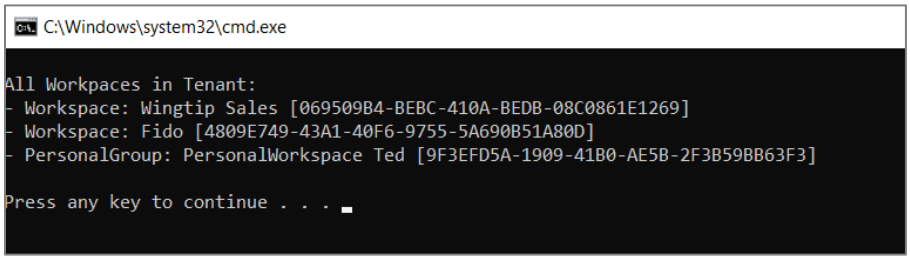
f) Run the console application in the Visual Studio debugger by pressing the **CTRL** + **{F5}** keyboard combination.

g) When you sign in, you should now be prompted to consent to a humongous delegated permission set.



h) Click **Accept** to continue running the code for the program.

i) 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.

31. Use a PowerShell script to create a new Azure AD group.

a) Open **CreateADGroupForPowerBiApps.ps1** at the following path.

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

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

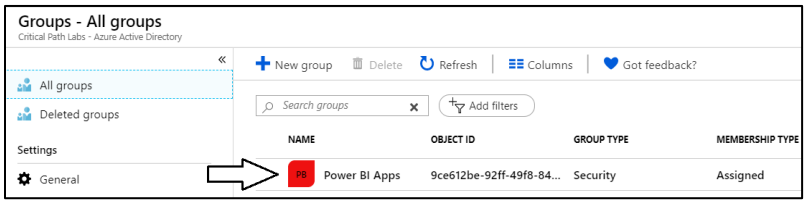
```
# log into Azure AD
$userName = "student@helavaworkshop.onMicrosoft.com"
$password = "myCAT$rightLEG"
```

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

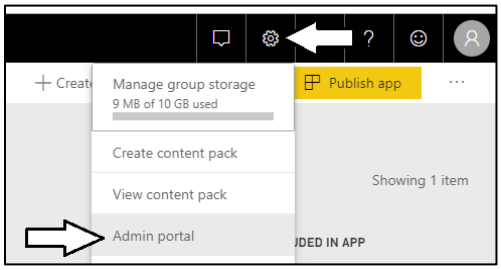
32. Use the Azure portal to verify the new Azure AD group named **Power BI Apps** has been created.
    a) Navigate to the **All groups** blade of the Azure AD portal using the following URL.

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

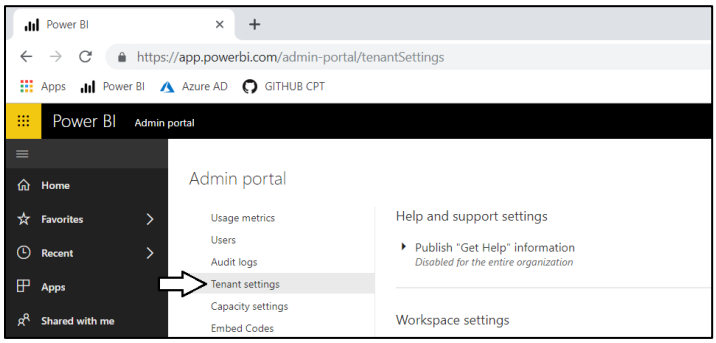
    b) Verify that you can see the new Azure AD security group named **Power BI Apps**.



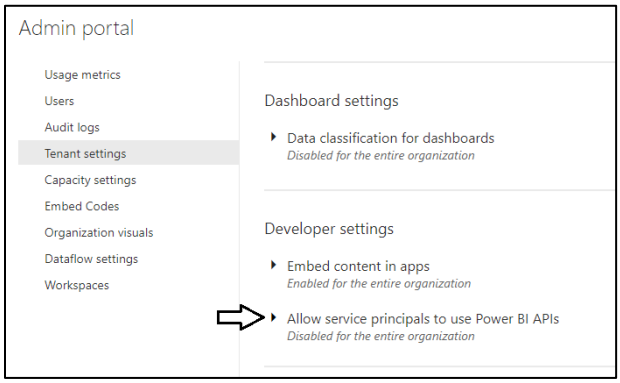
33. Enable the **Allow service principals to use Power BI APIs** setting and configure it with the **Power BI Apps** security group.
    a) Navigate to the Power BI portal at https://app.powerbi.com.
    b) Drop down the **Settings** menu and select the navigation command for the **Admin portal**.



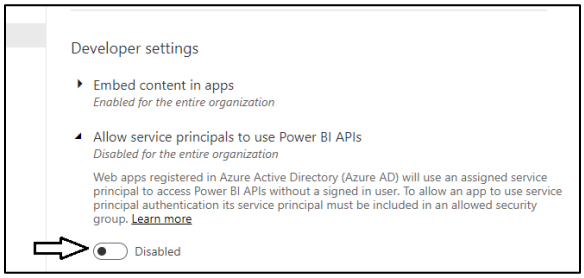
    c) In the Power BI Admin portal, click the **Tenant settings** link on the left.



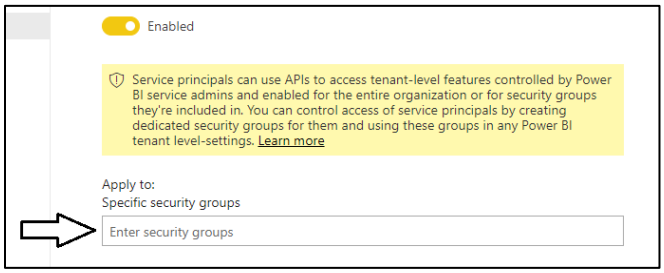
    d) Move down in the **Developer settings** section and expand the **Allow service principals to use Power BI APIs** section.



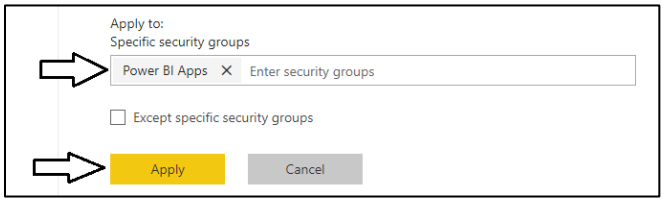
e) Note that the **Allow service principals to use Power BI APIs** setting is initially set to **Disabled**.



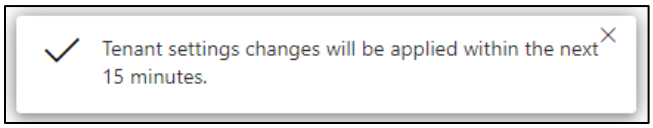
f) Change the setting to **Enabled** and place your cursor inside the **Apply to: Specific security groups** textbox.



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



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



34. Run a PowerShell script to create new confidential client application in Azure AD with a client secret.
    a) Open the PowerShell script named **RegisterPowerBiServiceApp3.ps1**.
    b) Update the two PowerShell variables named **$username** and **$userPassword**.and save your changes.
    c) Review what this script does when it runs.
        i) Connects to your tenant in Azure AD with global tenant admin permissions
        ii) Generates a new GUID and uses it generate a new password credential which serves as the client secret.
        iii) Creates a new confidential client application with the password credential and a display name of **Power BI Service App 3**.
        iv) Configures your Office 365 user account to be an owner of the new Azure AD application.
        v) Assigns the service principal for the **Power BI Service App 3** application as member of the **Power BI Apps** security group.
        vi) 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!*

d) Run the PowerShell script named **RegisterPowerBiServiceApp3.ps1**.

e) The script should create a new confidential application and display a text file with the info you will need in your application.

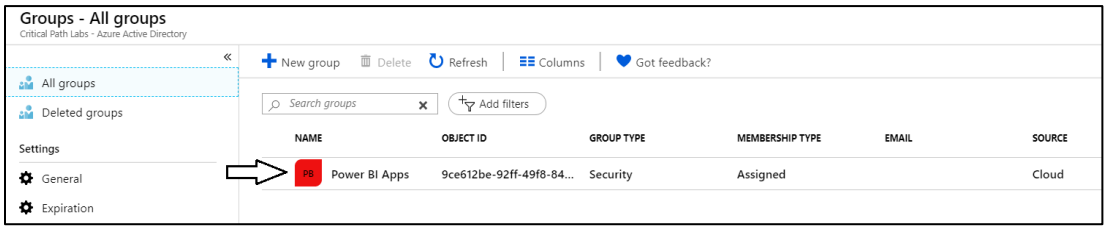
Leave this text file open as you will need to copy and paste the **ClientId** and **ClientSecret** into your application source code.

35. Inspect the group membership for **Power BI Apps** security group.

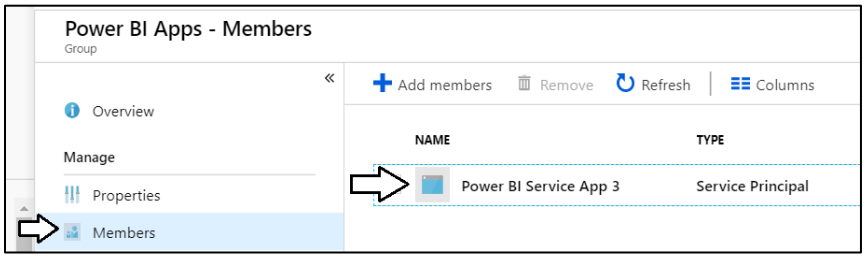
a) Navigate to the **All groups** blade of the Azure AD portal using the following URL.

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

b) Click on the link for the **Power BI Apps** group.



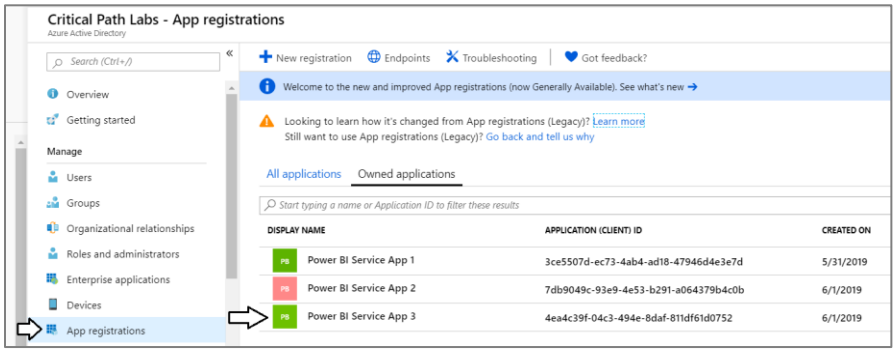
c) Click the **Members** link on the left and verify that **Power BI Service App 3** has been added as a member.



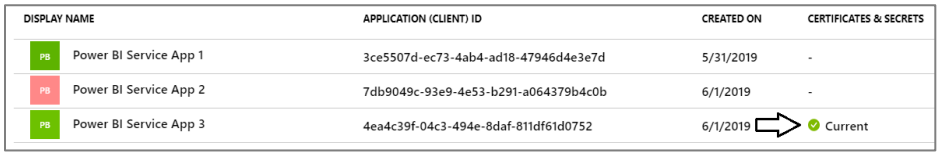
36. Inspect the confidential client application named **Power BI Service App 3** in the Azure portal

a) Navigate to the Azure AD **App registrations** blade in the Azure portal.

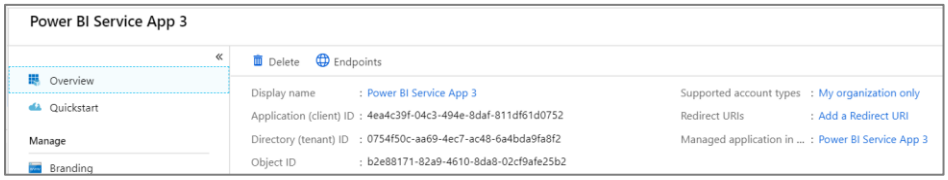
b) Locate the new Azure AD application named **Power BI Service App 3**.



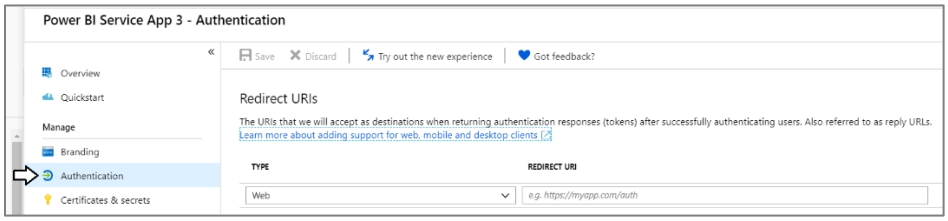
c) You should be able to see that **Power BI Service App 3** is configured with a client secret which is active (*e.g. Current*).



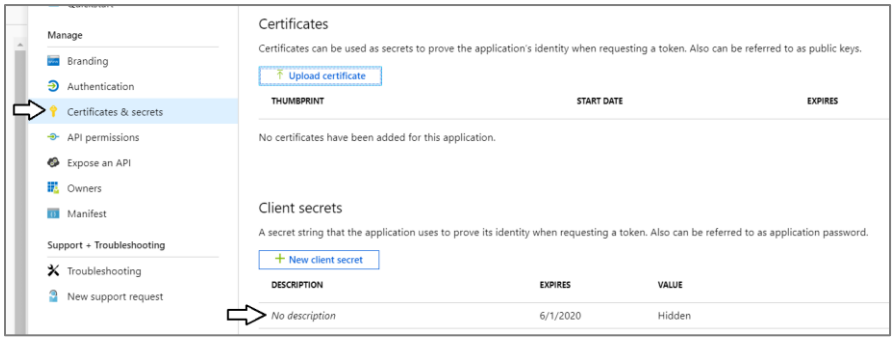
d) Click on the link for **Power BI Service App 3** to navigate to it summary page.



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

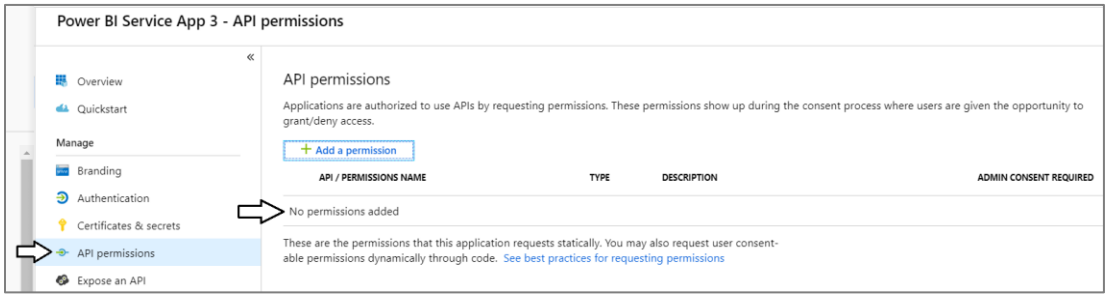


f) Click on the **Certificates & secrets** link and verify that the application has a client secret that expires in a year.



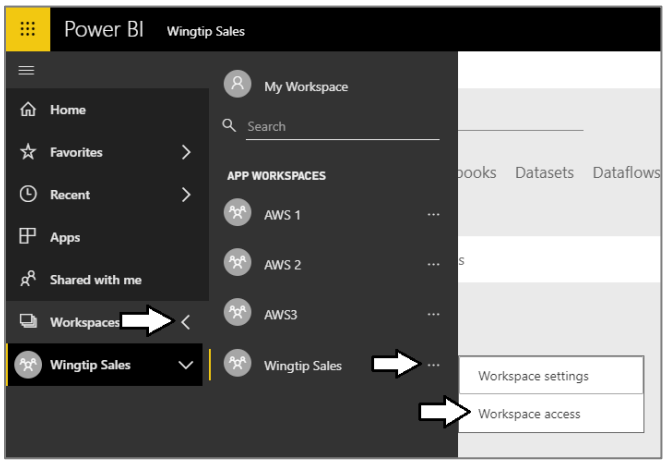
Note that it's not currently possible to assign a **Description** to a client secret created with PowerShell. However, it will still work fine.

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

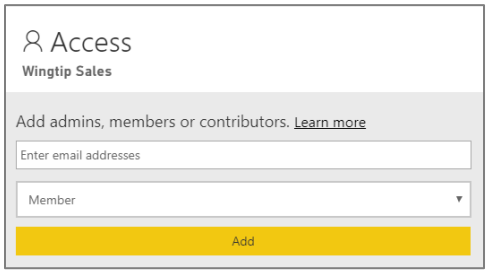


Remember that Power BI Service API permissions are not used on calls made with an app-only token. Instead, access is configured on a workspace-by-workspace basis by adding the confidential application's service principal as a workspace admin.

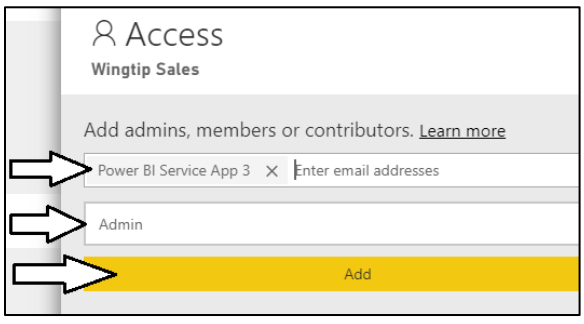
37. Configure the service principal for **Power BI Service App 3** as an admin for the app workspace named **Wingtip Sales**.
    a) Navigate to the Power BI portal.
    b) Expand the **Workspaces** flyout menu.
    c) Click the **Wingtip Sales** workspace context menu (**…**) and select **Workspace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



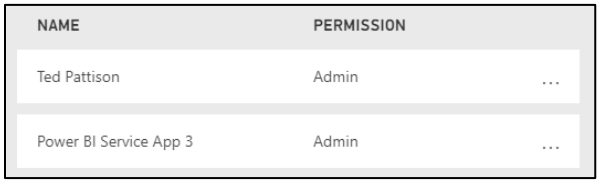
    d) On the right of the page, you should see the **Access** pane for the **Wingtip Sales** workspace.



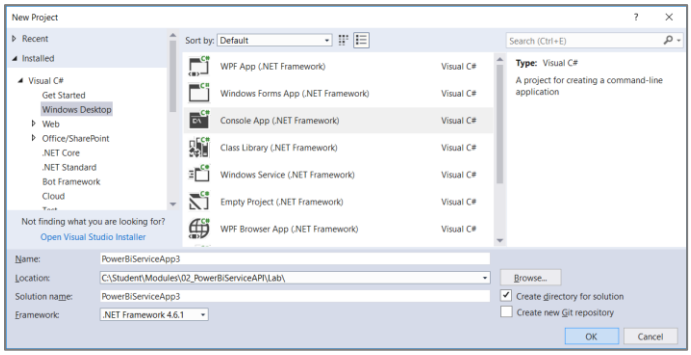
    e) Place the cursor into the *Enter email address* textbox and type **Power BI Service App 3**.
    f) Change the member type from **Member** to **Admin**.
    g) Click to **Add** button.



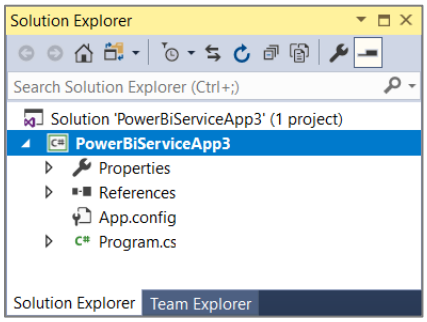
    h) Verify that **Power BI Service App 3** has been added as a workspace member with **Admin** permissions.



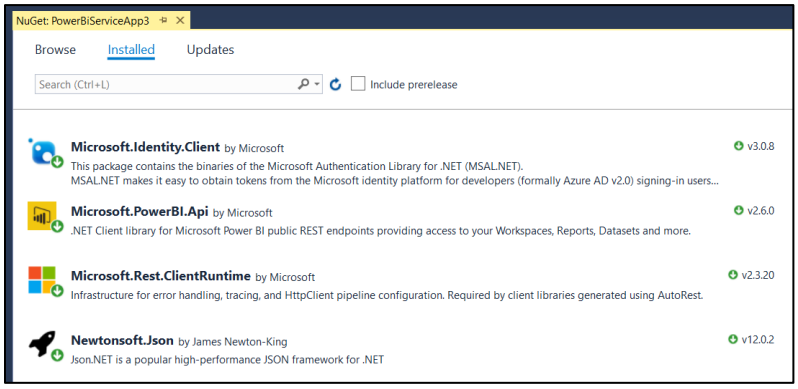
38. Create a new C# Console application in Visual Studio to access the Power BI Service API using an app-only access token.
    a) Return to Visual Studio and create a new project by running the **File > New Project** command.
    b) Select a project type of **Console App (.NET Framework)** from the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



    f) You should now have a new project named **PowerBiServiceApp3**.



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



    f) Close the window for the Nuget Package Manager.

40. Add the C# starter code to **program.cs**.

a) Using Windows Explorer, locate the file **PowerBiServiceApp3_Starter.cs.txt** in the **Student** folder at the following path.

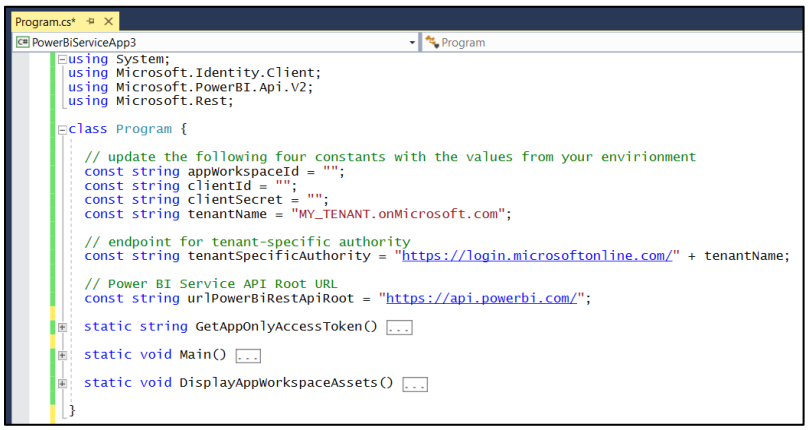
```
C:\Student\Modules\02_PBIRestApi\Lab\StarterFiles\PowerBiServiceApp3_Starter.cs.txt
```

b) Open the file named **PowerBiServiceApp3_Starter.cs.txt** in Notepad and copy its contents into the Window clipboard.

c) Return to the **PowerBiServiceApp3** project in Visual Studio.

d) Open the source file named **program.cs**, delete all the code inside and replace it with the content in the Windows clipboard.

e) You should now have the basic code for a simple C# console application which accesses the Power BI Service API.



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

```
// update the following four constants with the values from your envirionment
const string appWorkspaceId = "7186f418-2a89-4efc-a8fc-b184ad353c70";
const string clientId = "eb2e4b17-7c64-44b4-90cb-5229c30ff0d1";
const string clientSecret = "NjEwNDZiZjAtNzNjMyOOZTRkLWIwNzAtOTgwYWI2ODQyNmIz=";
const string tenantName = "pbi0520.onMicrosoft.com";
```

g) 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;
}
```

h) You can observe that the **DisplayAppWorkspaceAssets** method calls the **GetAppOnlyAccessToken** method.

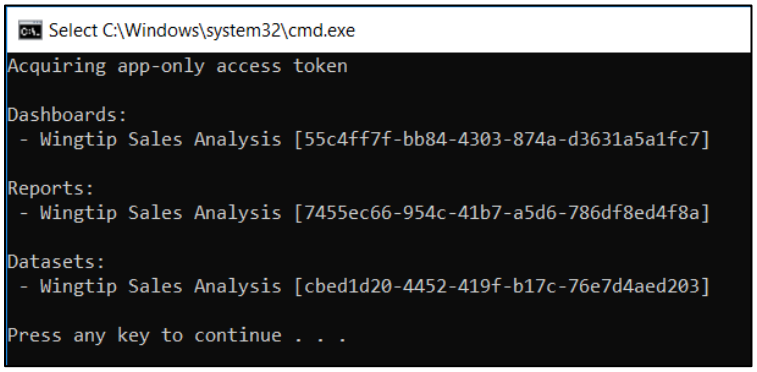
```
static void DisplayAppWorkspaceAssets() {

    string AccessToken = GetAppOnlyAccessToken();
    var pbiClient = new PowerBIClient(new Uri(urlPowerBiRestApiRoot),
                                      new TokenCredentials(AccessToken, "Bearer"));
```

i) You can observe that the **Main** method calls the **DisplayAppWorkspaceAssets** method.

```
static void Main() {
  DisplayAppWorkspaceAssets();
}
```

j) Run the app in the Visual Studio debugger by pressing the **CTRL** + **{F5}** keyboard combination.

k) The application should run successfully and display the following output in the console window.



l) 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.