Developing with the Power BI Service API

Setup Time: 60 minutes

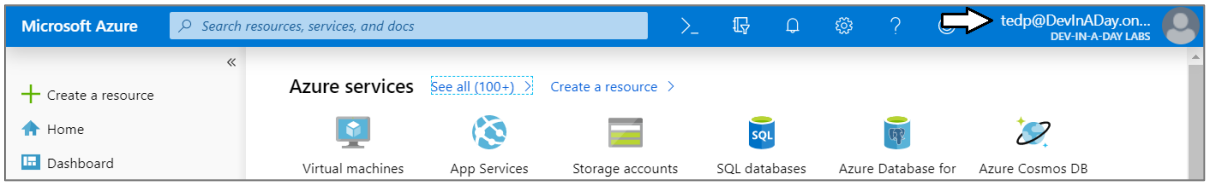
Lab Folder: C:\Student\Modules\02_PowerBiServiceAPI\Lab

Overview: In this lab, you will begin by creating a new Azure AD application that allows you to call the Power BI Service API. After that, you will use Visual Studio to create a new C# console application that programs using the Power BI SDK.

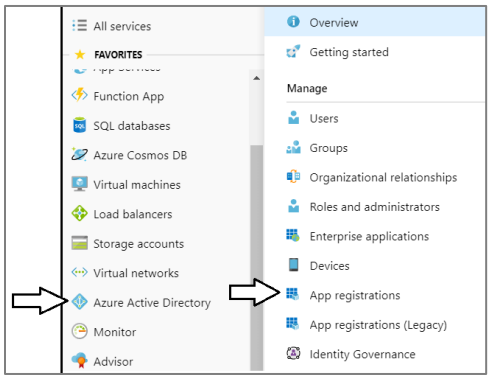
Exercise 1: Register a New Application with Azure Active Directory

In this exercise, you will register a new application with Azure AD and you will configure the application's required permissions to access 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 are log into the Azure portal, check the email address in the login menu in the upper right to make sure you are logged in the Azure portal with the correct identity.

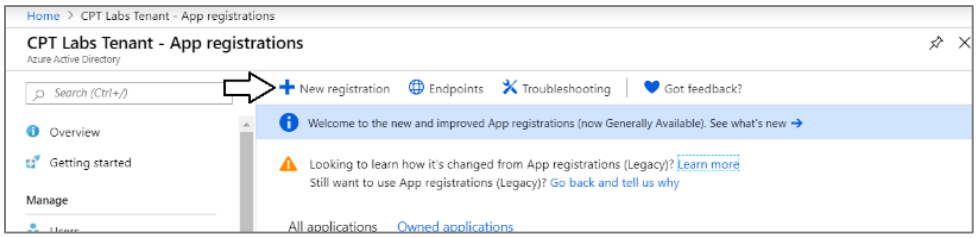


2. Register a new Azure AD application.
   a) In the left navigation, scroll down and click on the link for **Azure Active Directory**.
   b) Click the link for **App registration**.

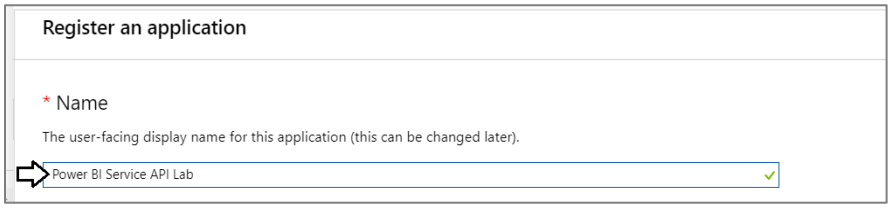


Note that the Azure portal user experience for creating and configuring Azure AD applications was updated in April 2019. You can get to the old user experience by clicking the **App registrations (Legacy)** link. In this lab exercise, we want you to use the new user experience to become comfortable with it.

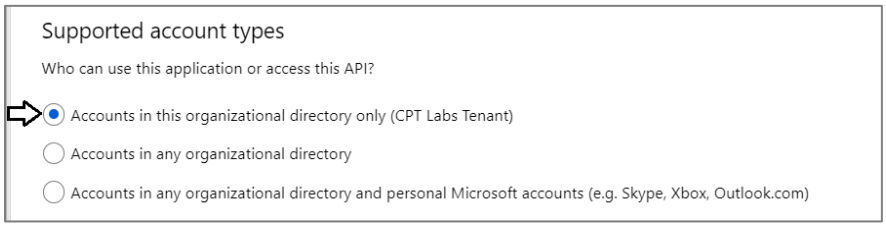
   c) Click **New application registration**.



d) Enter a **Name** of **Power BI Service API Lab**.



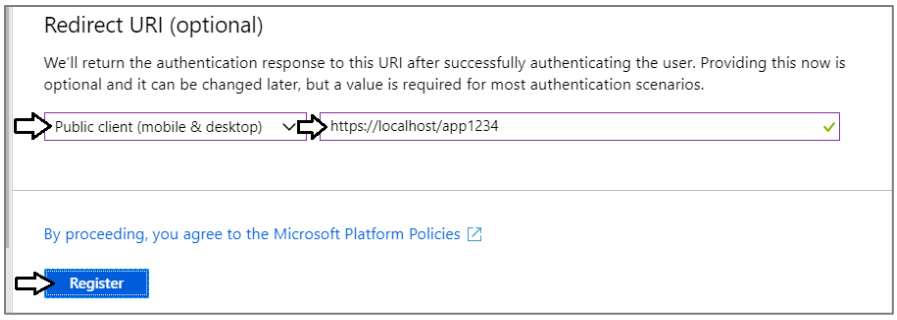
e) For the **Supported account types** option, leave the default value of **Accounts in this organizational directory only**.



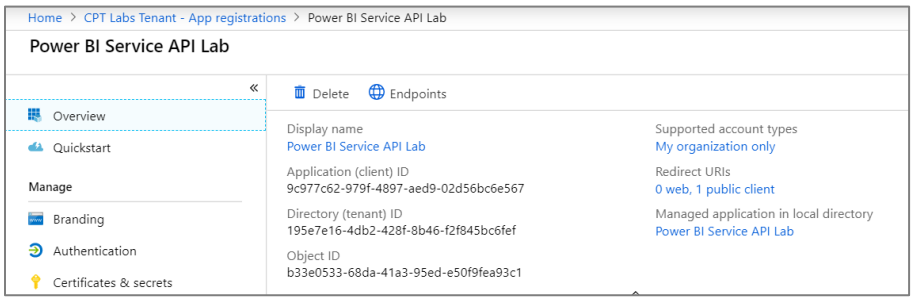
f) In the **Redirect URI** section, select **Public client (mobile or desktop)** in the left dropdown.

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

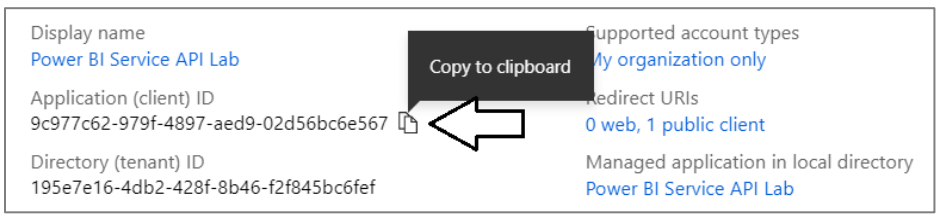
h) Click the **Register** button to create the new Azure AD application.



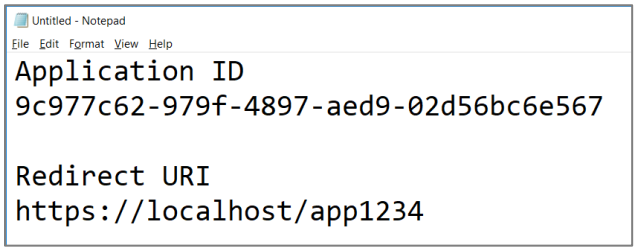
i) Once you've created the new application you should see the application summary view as shown in the following screenshot..



j) Copy the Application ID to the Windows clipboard.

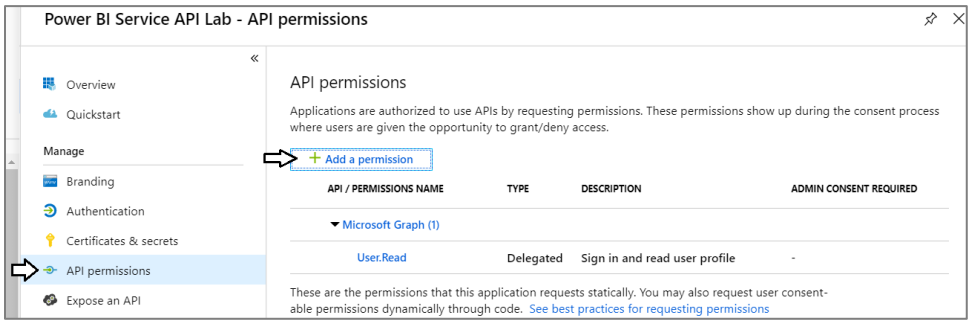


k) Launch Notepad and paste the Application ID into a new text file. Also add the value of the Redirect URI.

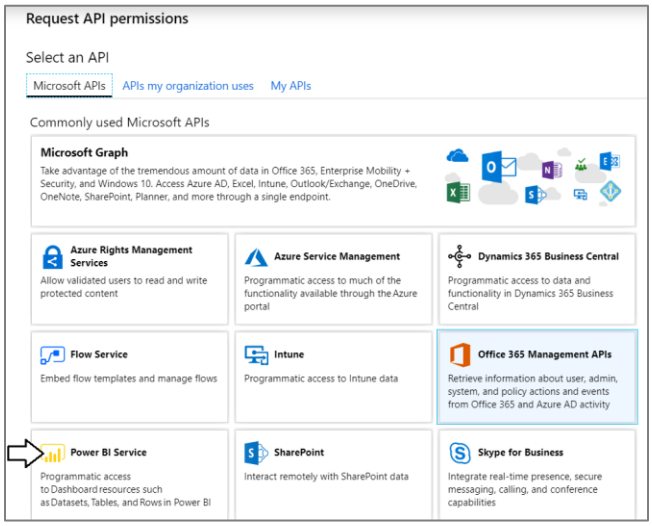


l) Click the **API permissions** link on the left.

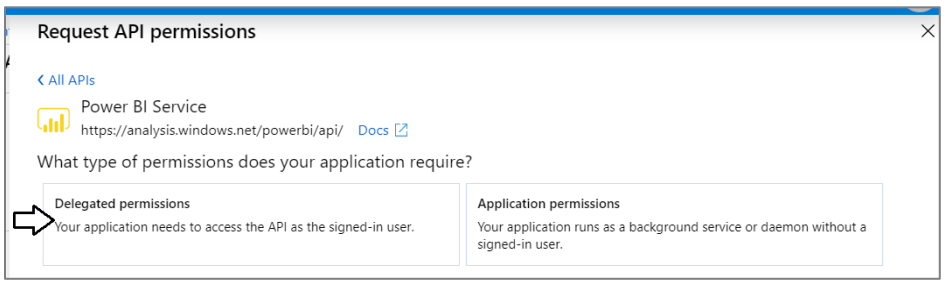
m) Click the **Add a permission** button in the **API permissions** section.



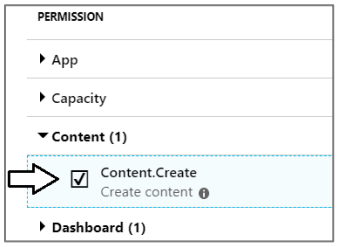
n) On the **Microsoft APIs** tab, click **Power BI Service**.



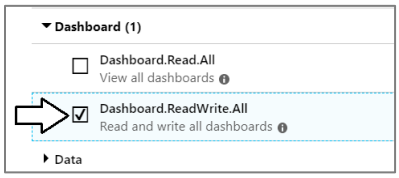
o) Click **Delegated Permissions**.



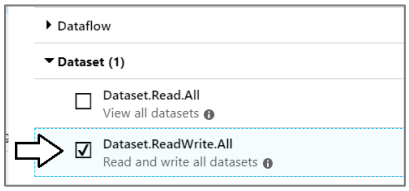
p) In the **PERMISSION** section, expand **Content** and select the **Content.Create** permission.



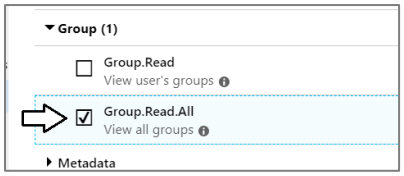
q) Expand **Dashboard** and select the **Dashboard.ReadWrite.All** permission.



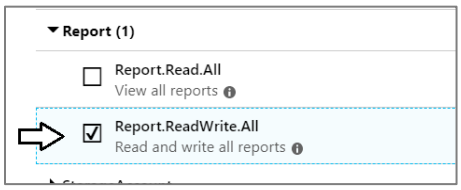
r) Expand **Dataset** and select the **Dataset.ReadWrite.All** permission.



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

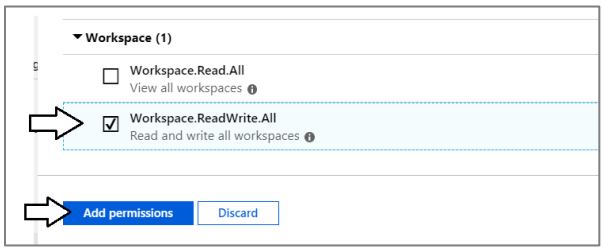


t) Expand **Report** and select the **Report.ReadWrite.All** permission.

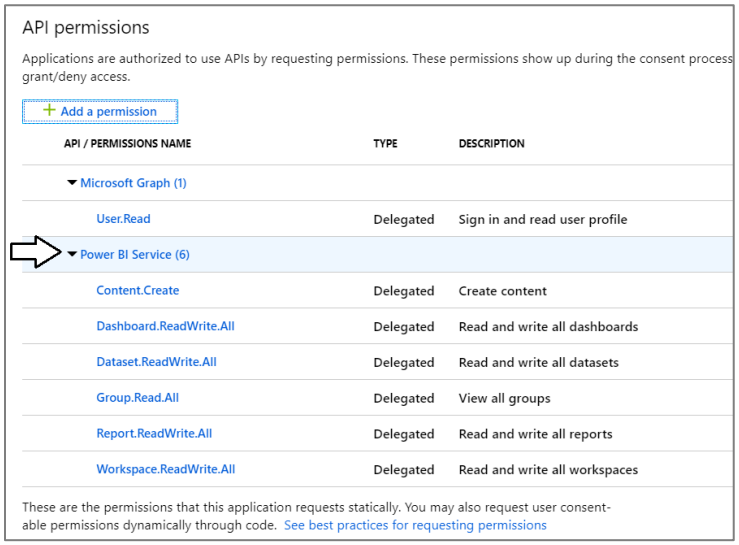


u) Expand **Workspace** and select the **Workspace.ReadWrite.All** permission.

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

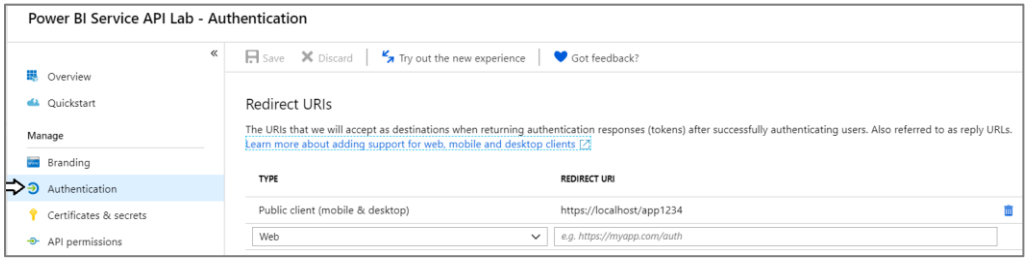


w) At this point, you should be able to verify that the Power BI Service has been added to the **Required permissions** list.

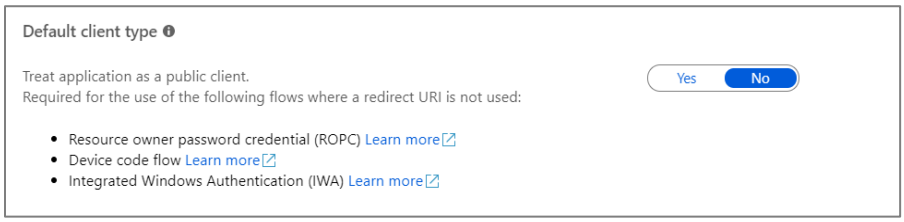


3. Change the application's **Default client type** setting to support the User Password Credential flow.

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**.

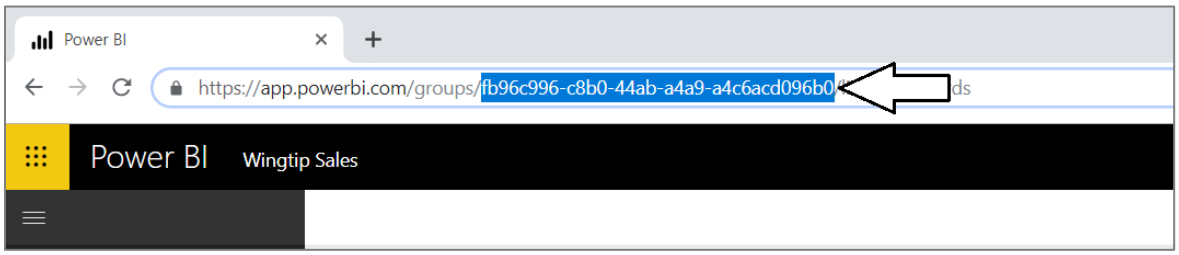
Click the **Save** button at the top of the page to save your changes.

You are now done registering your application with Azure AD.

Exercise 2: Call the Power BI Service API using the Power BI SDK

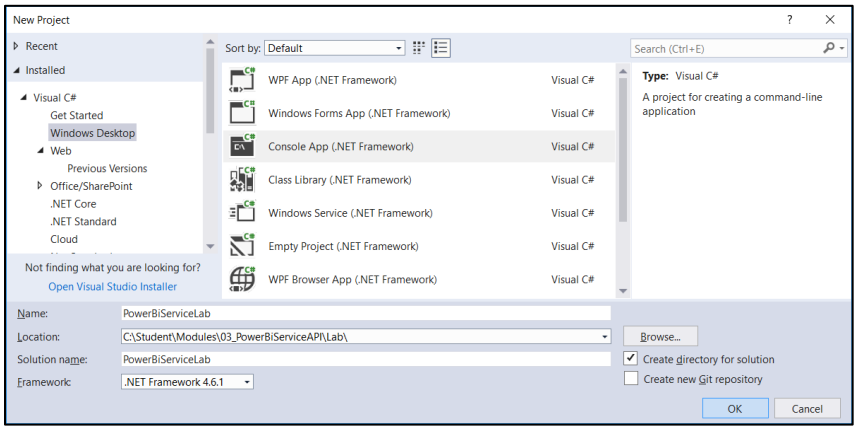
In this exercise, you will create a C# console application to call into the Power BI Service API. Before creating the Console application in Visual Studio, you will first record the GUID for the **Wingtip Sales** app workspace which will be needed later in this exercise.

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/**.

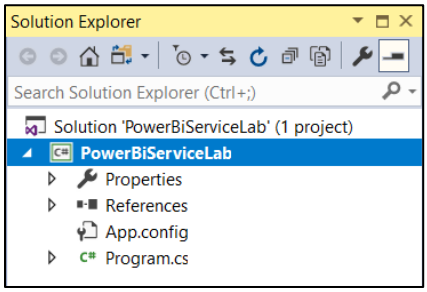


   c) Copy the **App Workspace ID** into the same text file you created earlier to hold the **Application ID** and the **Redirect URI**.

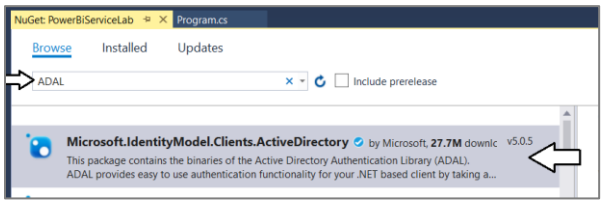
1. Create a new C# Console application in Visual Studio.
   a) Launch Visual Studio.
   b) Create a new project by running the **File > New Project** command.
   c) Select a project type of Console App from the Visual C# project templates.
   d) Give the project a **Name** of **PowerBiServiceLab** and
   e) Give the project a **Location** of **C:\Student\Modules\02_PowerBiServiceAPI\Lab**.
   f) Click **OK** to create the new project.



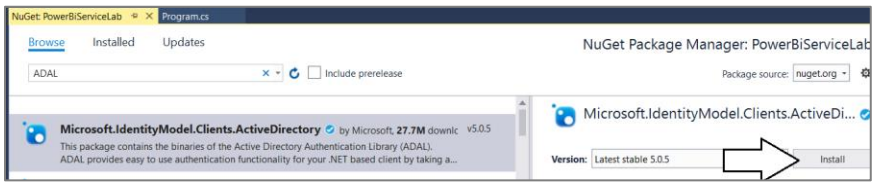
g) You should now have a new project named **PowerBiServiceLab**.



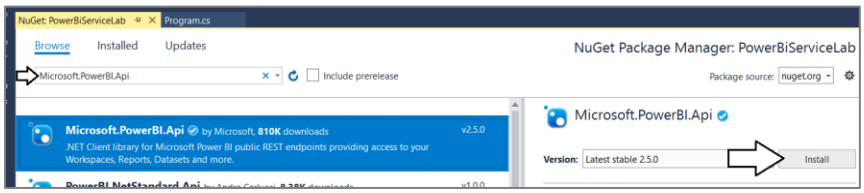
2. Add the NuGet packages to the project required to program the Power BI Service API using the Power BI SDK.
   a) Right-click the top-level node for the **PowerBiServiceLab** project and select **Manage NuGet Packages…**.
   b) Click the Browse tab and type **ADAL** into the search box.
   c) Locate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



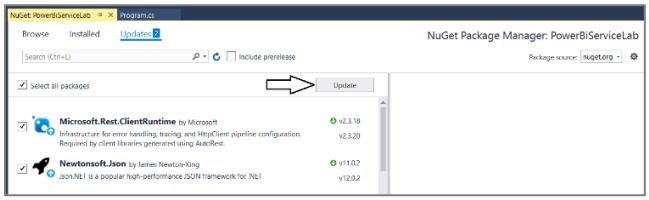
   d) Select and install **Microsoft.IdentityModel.Clients.ActiveDirectory**.



   e) When prompted about the licensing agreement, click **I Agree**.
   f) Search for Power BI and then find and install the **Microsoft.PowerBI.Api**.



   g) When prompted about the licensing agreement, click **I Agree**.
3. 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.

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

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

   b) Open the file named **ProgramStarter.cs.txt** in Notepad and copy its contents into the Window clipboard.
   c) Return to the **PowerBiServiceLab** 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 code for a simple C# console application which access the Power BI Service API.

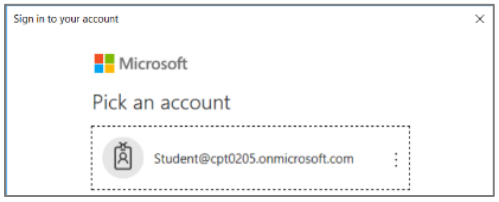
```
using System;
using Microsoft.IdentityModel.Clients.ActiveDirectory;
using Microsoft.PowerBI.Api.V2;
using Microsoft.Rest;

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 DisplayPersonalWorkspaceAssets() ...

}
```

5. 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.

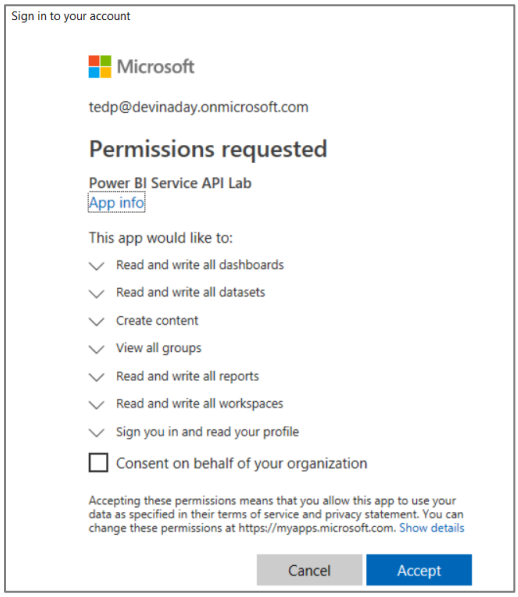
```
// enter the correct configuration values for your environment
static string appWorkspaceId = "dfe5e680-a85a-4731-8c89-963fa5c6c86e";
static string clientId = "0e0dd766-4db3-4dab-bc27-baecab09864d";
static string redirectUrl = "https://localhost/app1234";
```

Remember that **Application ID** and **Client ID** are two names that mean the same thing.

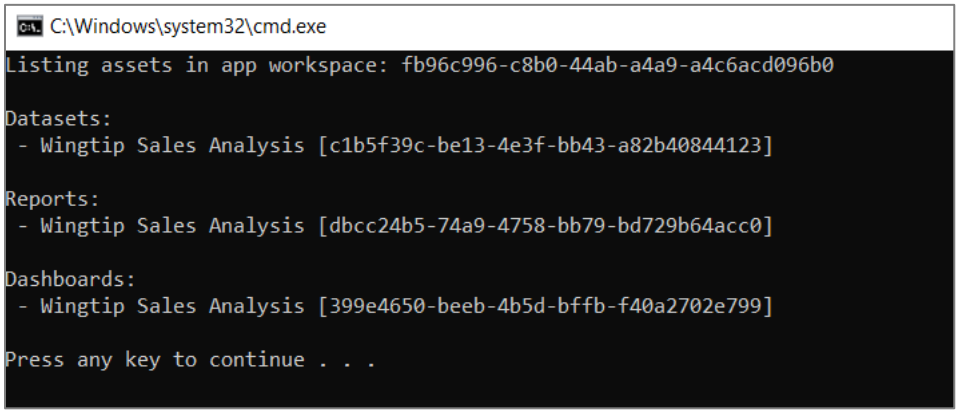
   c) Save your changes to **program.cs**.
6. Run the application to call to the Power BI Service API.
   a) Press the **{F5}** key to begin a debugging session.
   b) When promoted to sign in, log in using your Office 365 user account credentials.



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.

7. 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 Credentials 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.

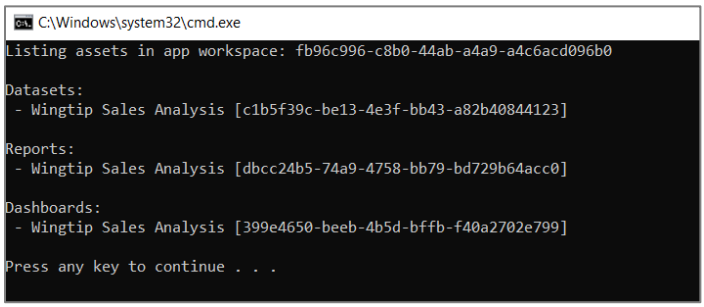
```
// use authentication context to sign-in using User Password Credentials flow
string masterUserAccount = "student@portlandembed.onMicrosoft.com";
string masterUserPassword = "pass@word1";
UserPasswordCredential creds = new UserPasswordCredential(masterUserAccount, masterUserPassword);
```

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

8. 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 as it did before but it should no longer require you to interactively enter a user name and password.



Note the 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 the a C# Console application to create app workspaces and publish PBIX project files.

1. Add the code required to create a new app workspace.

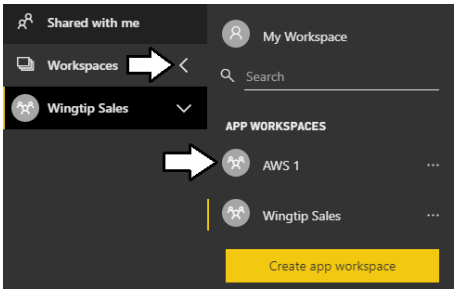
a) Add the static **CreateAppWorkspace** method to the bottom of the **Program** class in **program.cs**.

```
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() {

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

}
```

2. 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 1**.



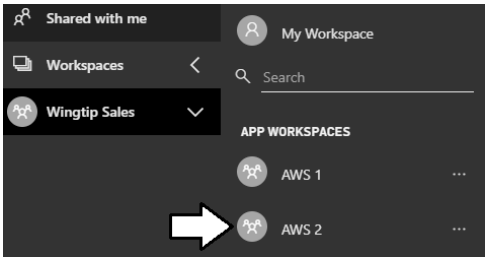
3. Add the code required to publish a PBIX project file to an app workspace.
   a) Add the static **PublishPBIX** method to the bottom of the **Program** class in **program.cs**.

```
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**.

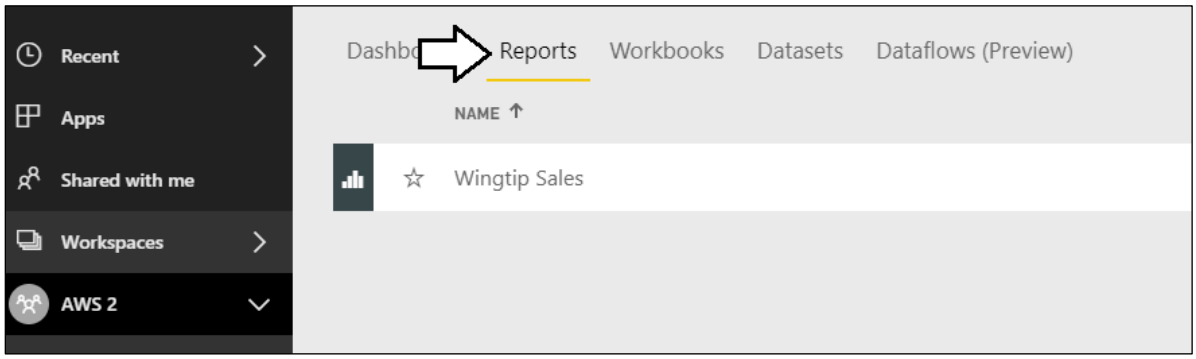
```
static void Main() {
  //DisplayPersonalWorkspaceAssets();
  //CreateAppWorkspace("AWS 1");

  string appWorkspaceId = CreateAppWorkspace("AWS 2");
  string pbixPath = @"C:\Student\PBIX\Wingtip Sales Analysis.pbix";
  string importName = "Wingtip Sales";
  PublishPBIX(appWorkspaceId, pbixPath, importName);
}
```

4. 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 which is **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 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.

5. 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 **PowerBiServiceLab** project in Visual Studio.

d) Return to the source file named **program.cs**.

e) Place you cursor at the bottom of the **Program** class and paste in the content you copied into the Windows clipboard.

f) The **Program** class should now contain a method named **CloneAppWorkspace**.

```
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 = "dfe5e680-a85a-4731-8c89-963fa5c6c86e";
  static string clientId = "0e0dd766-4db3-4dab-bc27-baecab09864d";
  static string redirectUrl = "https://localhost/app1234";

  static string GetAccessToken() ...

  static PowerBIClient GetPowerBiClient() ...

  static void Main() ...

  static void DisplayPersonalWorkspaceAssets() ...

  static string CreateAppWorkspace(string Name) ...

  static void PublishPBIX(string appWorkspaceId, string PbixFilePath, string ImportName) ...

  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)) {
```

6. 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");
```

You will be able to see the PBIX file created in **C:\Student\downloads** folder when the program runs.

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);
  }
}
```

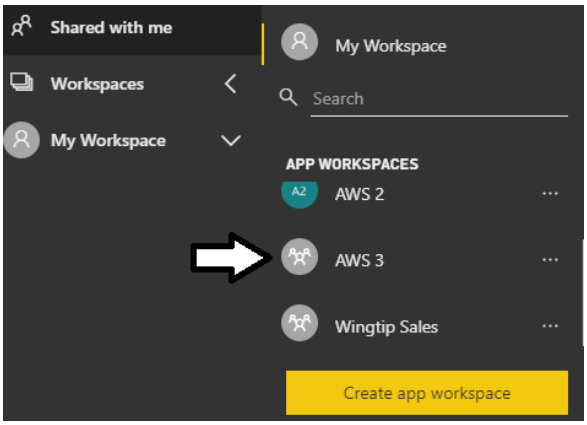
d) Update the **Main** method to match the following code which uploads a PBIX file with an Import name of **Wingtip Sales**.

```
static void Main() {
  //DisplayPersonalWorkspaceAssets();
  //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");

}
```

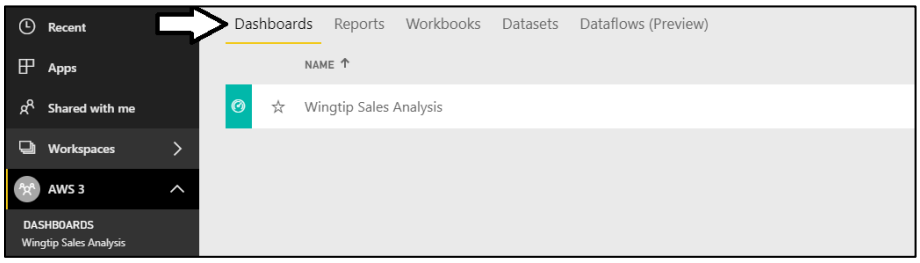
In the following step you will run the program once more to test you implementation of **CloneAppWorkspace**.

When you test **CloneAppWorkspace**, the program will clone the dataset and report by exporting then and importing a PBIX file.

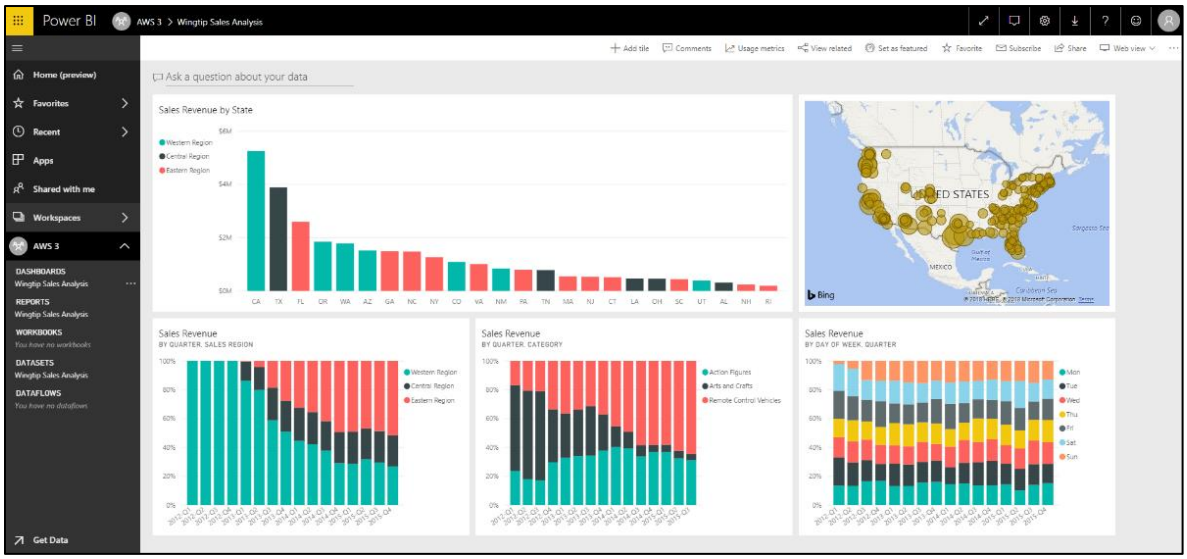
7. Run the application to call to the Power BI Service API.
   a) Press the **{F5}** key to begin a debugging session.
   b) The program should run without any errors.
   c) After the program runs, you should be able to confirm that it created a new app workspace named **AWS 3**.



   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 clone 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 a second console application which will use the C# Power BI SDK to call the Power BI Service API. This console application will be different from the one you created earlier in the lab 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 to test how Power BI Service API permissions can be incrementally expanded over time.

1. Use a PowerShell script to create a new Azure AD application.
   a) Open a PowerShell script editor such as the PowerShell ISE or Visual Studio Code.
   b) Open the PowerShell script at the following path.

```
C:\Student\Scripts\RegisterPublicClientApp.ps1
```

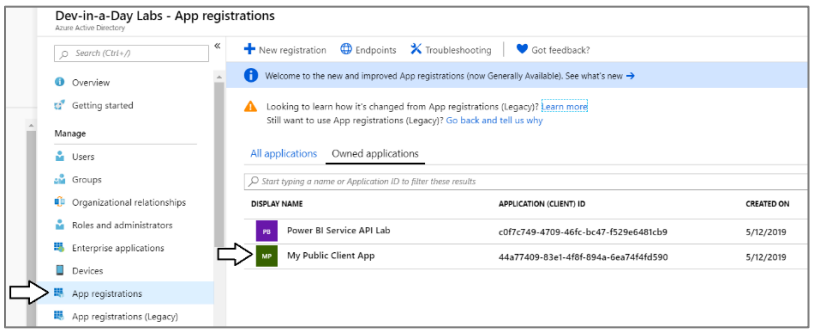
   c) Update the variables named **$userName** and **$password** with the credentials for your Office 365 user account.

```
RegisterPublicClientApp.ps1
# log into Azure AD
$userName = "student@hellovaworkshop.onMicrosoft.com"
$password = "myCat$rightLEG"
```

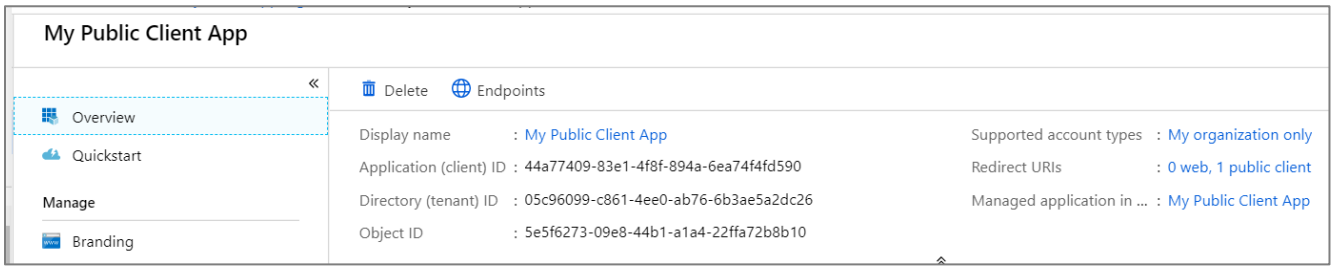
   d) Save you changes to **RegisterPublicClientApp.ps1** and run the script.
   e) When the script runs, it will create an Azure AD application and display the details in a text file as shown in the this screenshot.

```
PublicClientAppInfo.txt - Notepad
File Edit Format View Help
--- New Azure AD Public Client App Info ---
AppId: 44a77409-83e1-4f8f-894a-6ea74f4fd590
ReplyUrl: https://localhost/app1234
```

2. Inspect the new application named **My Public Client App** 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 **My Public Client App**.



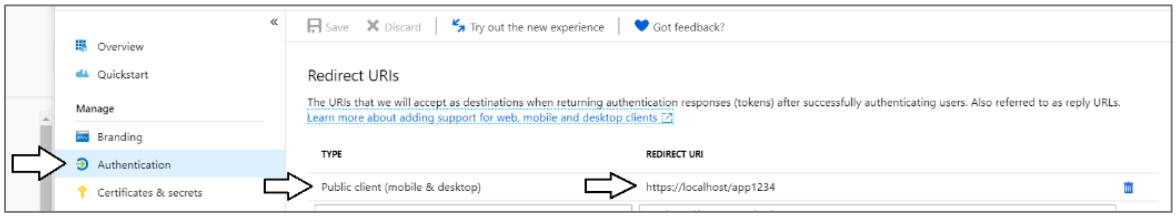
   e) You should now see the summary page for **My Public Client App**.



Note that you do not need to modify the app because the PowerShell script was able to configure the app with all the required settings. However, you will now examine a few settings for the app that were configured by the PowerShell script.

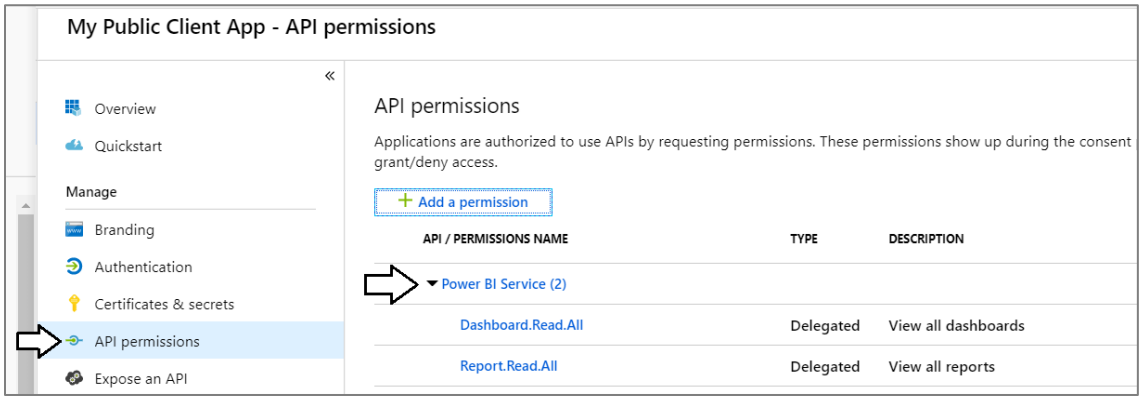
f) Click the **Authentication** link on the left.

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).

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

a) Launch a new instance of Visual Studio.

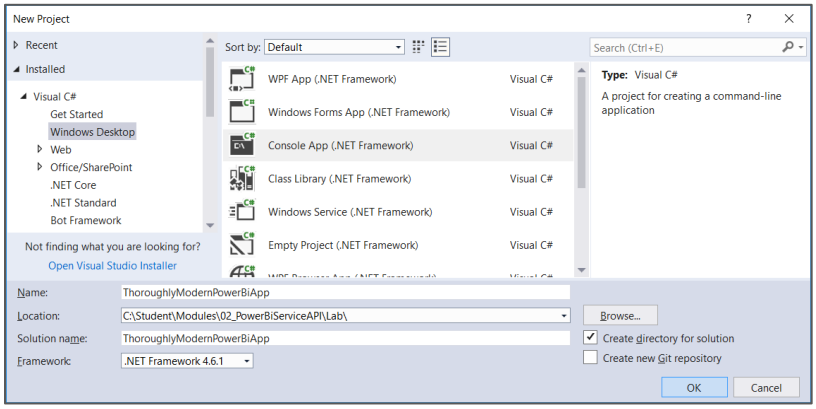
b) Create a new project by running the **File > New Project** command.

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

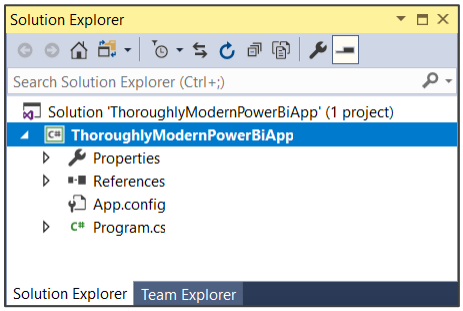
d) Give the project a **Name** of **ThoroughlyModernPowerBiApp**.

e) Give the project a **Location** of **C:\Student\Modules\02_PowerBiServiceAPI\Lab**.

f) Click **OK** to create the new project.

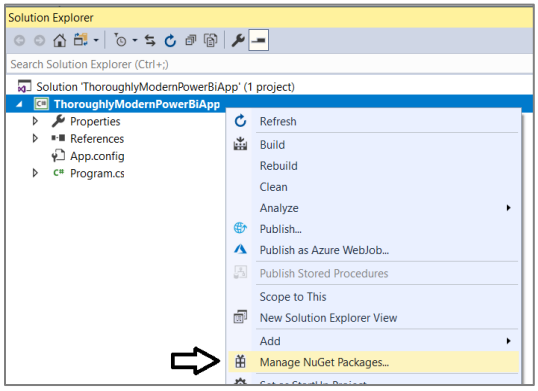


g) You should now have a new project named **ThoroughlyModernPowerBiApp**.



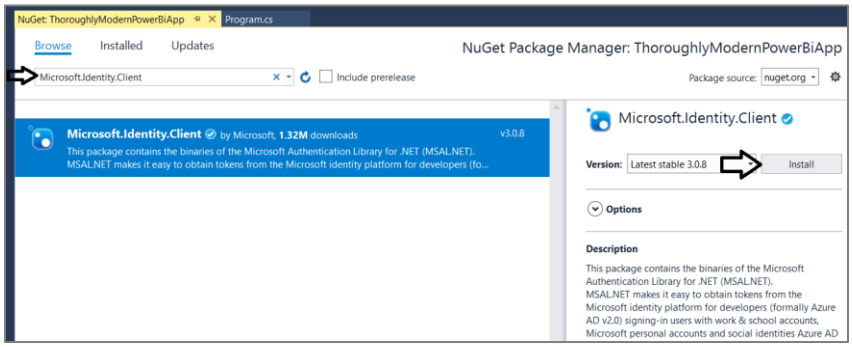
4. Add the NuGet packages to the project required to program the Power BI Service API using the Power BI SDK.

a) Right-click the top-level node for the **PowerBiServiceLab** project and select **Manage NuGet Packages…**.

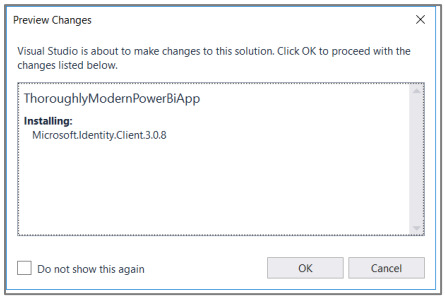


b) Click the Browse tab and type **MSAL** into the search box.

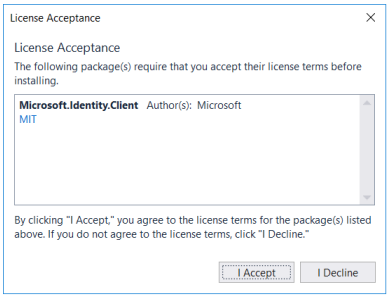
c) Locate and install the package **Microsoft.Identity.Client**. This is the package for the *Microsoft Authentication library (MSAL)*.



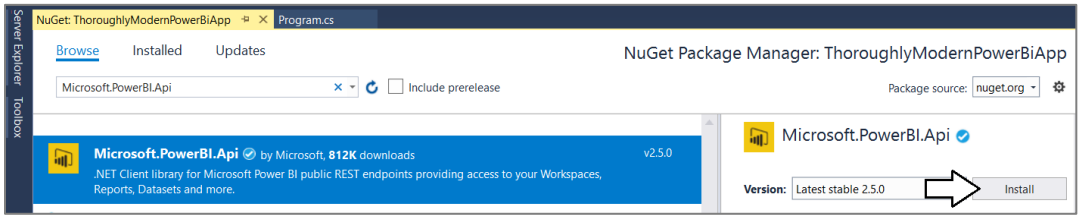
d) If you are prompted about **Preview Changes**, click **OK**.



e) When prompted about **License Acceptance**, click **I Agree**.



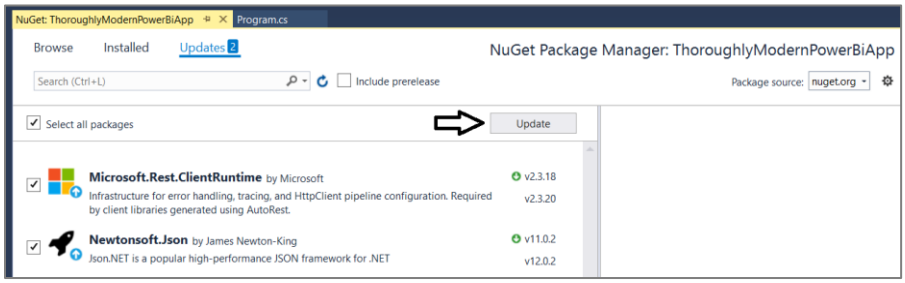
f) Search for Power BI and then find and install the **Microsoft.PowerBI.Api**.



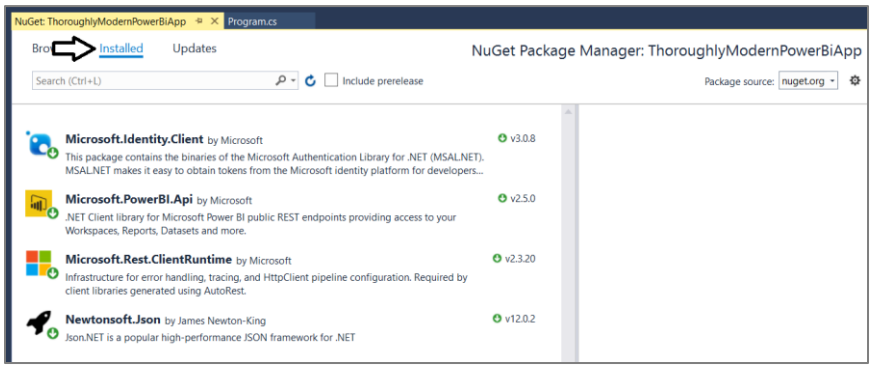
g) When prompted about the licensing agreement, click **I Agree**.

5. 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.

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

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

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

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

c) Return to the **ThoroughlyModernPowerBiApp** 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 code for a simple C# console application which access the Power BI Service API.

```
Program.cs
ThoroughlyModernPowerBiApp        ThoroughlyModernPowerBiApp.Program
using System;
using System.Security;
using Microsoft.Identity.Client;
using Microsoft.PowerBI.Api.V2;
using Microsoft.Rest;

namespace ThoroughlyModernPowerBiApp {

  class Program {

    // update the following three constants with values from your envirionment
    const string appWorkspaceId = "";
    const string clientIdPublicApp = "";
    const string redirectUri = "https://localhost/app1234";

    // generic v2 endpoint references "organizations" instead of "common"
    const string tenantAuthrority = "https://login.microsoftonline.com/organizations";

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

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

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

g) At the top of the **Program** class, you will see three constants named **appWorkspaceId**, **clientIdPublicApp** and **redirectUri**.

```
// update the following three constants with values from your envirionment
const string appWorkspaceId = "";
const string clientIdPublicApp = "";
const string redirectUri = "https://localhost/app1234";
```

h) Modify these constants with the values for your development environment.

```
// update the following three constants with values from your envirionment
const string appWorkspaceId = "6c221139-962e-4ec8-9174-4be003fe6688";
const string clientIdPublicApp = "810f63bb-8068-483c-9835-5d9b5404f4af";
const string redirectUri = "https://localhost/app1234";
```

7. Review the pre-provided code inside **Program.cs**.

a) Below in **Program.cs**, you will see two more constants named **tenantAuthrority** and **urlPowerBiRestApiRoot**.

b) There are several static string array fields whose names start with **scopes**. Each of these fields contains a set of permissions.

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

// 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) {

  PublicClientApplicationOptions options = new PublicClientApplicationOptions();

  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();
    }
```

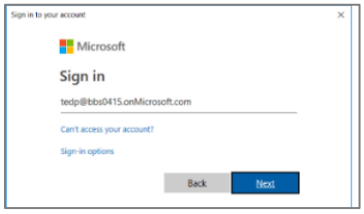
e) Code in **DisplayAppWorkspaceAssets** calls **GetAccessTokenInteractive** passing a parameter value of **scopesDefault**.

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

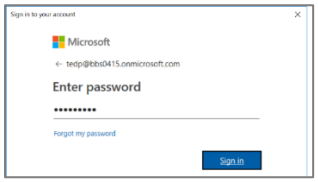
The first time you run the program, Azure AD will prompt for consent to the default permissions configured in the Azure AD application.

8. Run the application to test your work.

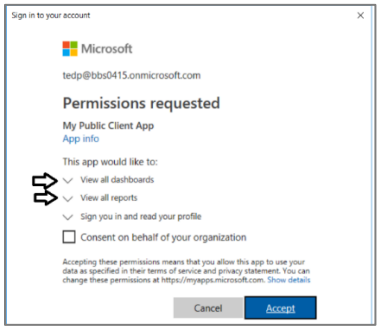
a) Press the **CTRL + {F5}** keyboard to run the program in the Visual Studio debugger.



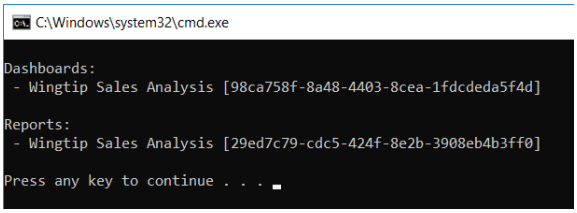
b) When prompted to sign in, enter your user name and password.



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.



9. 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**.

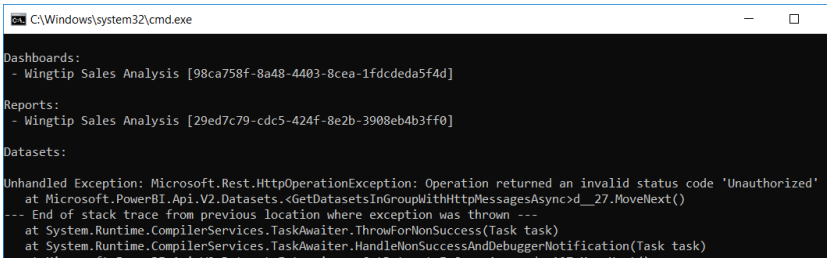
```
Console.WriteLine();
Console.WriteLine("Reports:");
var reports = pbiClient.Reports.GetReportsInGroup(appWorkspaceId).Value;
foreach (var report in reports) {
  Console.WriteLine(" - " + report.Name + " [" + report.Id + "]");
}

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 permissions to call **GetDatasetsInGroup**.

c) Press the **CTRL** + **{F5}** keyboard combination to run the **ThoroughlyModernPowerBiApp** program again.

d) When prompted, sign in.

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



10. Acquire an access token interactively with the required scopes.

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

```
static void DisplayAppWorkspaceAssets() {

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

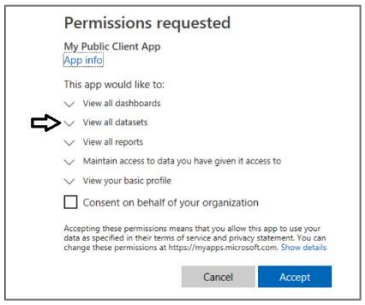
c) In the call to **DisplayAppWorkspaceAssets**, replace **scopesDefault** with **scopesReadWorkspaceAssets**.

```
static void DisplayAppWorkspaceAssets() {

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

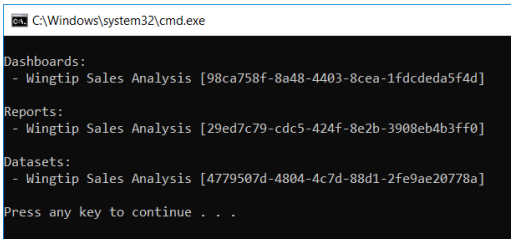
d) Press the **CTRL** + **{F5}** keyboard combination to run the **ThoroughlyModernPowerBiApp** program again.

e) When prompted, sign in.

f) After signing in, you should be prompted to consent to permissions including



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



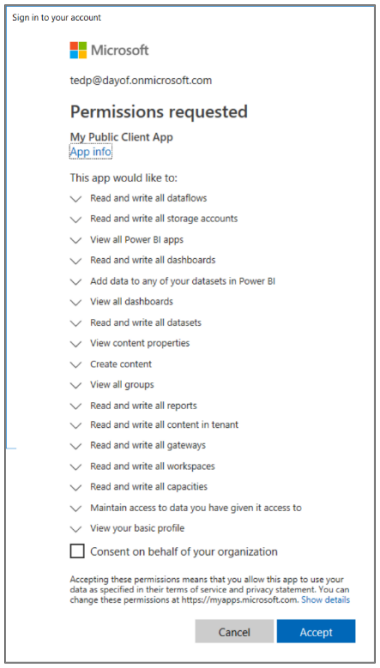
11. Acquire an access token with all the available Power BI Service API permissions.
    a) Inspect the static field named **scopesKitchenSink** and the scoped defined inside.

```
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"
};
```

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

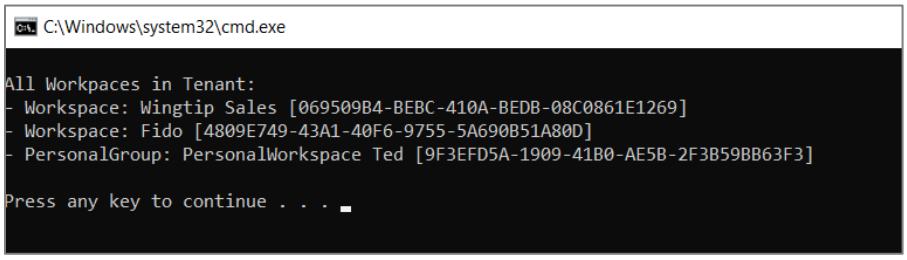
```
static void DisplayAllWorkspacesInTenant() {

  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();

}
```

    d) Run the program and again and sign in. You should now be prompted to consent to a large set of permissions.



e) Click accept to continue.

f) You should see that the program is able to see all the workspaces in the tenant including personal workspaces.



You have now completed this lab.