Embedding Power BI Reports and Dashboards

Setup Time: 45 minutes

Lab Folder: C:\Student\Modules\03_PowerBiEmbedding\Lab

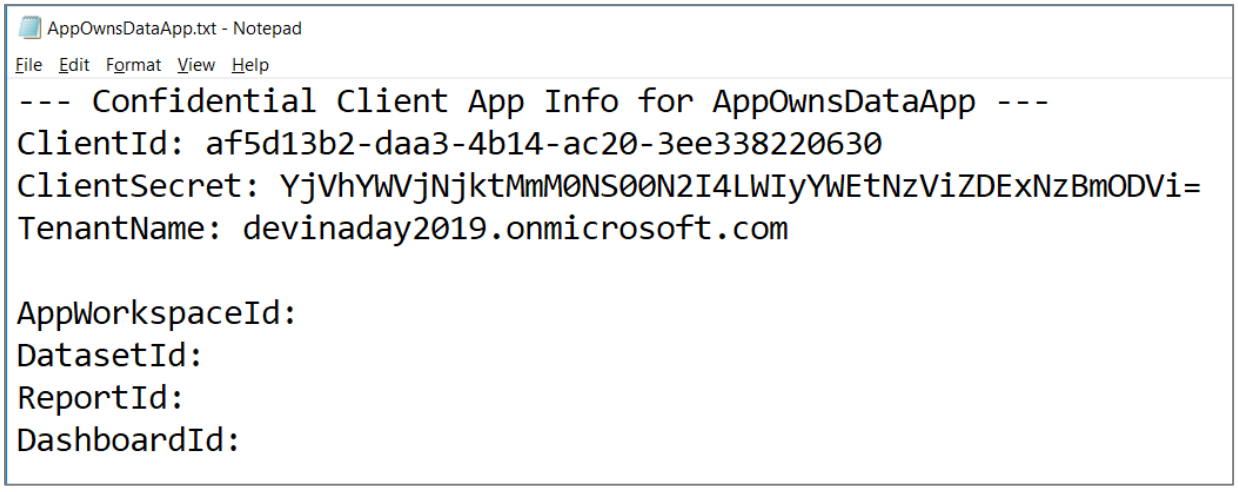
Overview: In this lab you will work through the steps to develop a custom web application that uses the App-Owns-Data model to embed Power BI reports and dashboards. First, you will first register a confidential client application with Azure AD and configure the new application's service principal as an Admin member of the **Wingtip Sales** workspace. After that, you will create an ASP.NET MVC web application using Visual Studio. As you move through the exercises of this lab, you will program against both the Power BI Service API and the Power BI JavaScript API to implement the required steps to embed Power BI reports and dashboards into your custom web application using third-party embedding.

Prerequisite: This lab assume you have already completed **Exercise 6** in the previous labs titled **Call the Power BI Service API using an App-only Access Token** in which you created the **Power BI Apps** group in Azure AD and used that group to configured the **Allow service principals to use Power BI APIs** setting in your Power BI tenant. If you have not yet completed **Exercise 6: Call the Power BI Service API using an App-only Access Token**, you should go back and complete steps 1 through 3 of this lab exercise before continuing with the exercises in this lab.

Exercise 1: Create a new Confidential Client Application in Azure AD

In this exercise, you will register a new application with Azure AD and you will configure the delegated permissions required to call the Power BI Service API and retrieve the data required to embed reports and dashboards.

1. Run a PowerShell script to register new confidential client application in Azure AD with a client secret.
   a) Open the PowerShell script named **RegisterAppOwnsDataApp.ps1**.
   b) Update the two PowerShell variables named **$username** and **$userPassword** and save your changes.
   c) Review what he script does.
      i) Connects to your tenant in Azure AD with global tenant admin permissions
      ii) Creates a new GUID and uses it to generate a new password credential which will acts the as the client secret
      iii) Creates a new confidential client app with password credentials and a display name of **App-Owns-Data Embedding App**.
      iv) Assigns your Office 365 user account as an owner of the new Azure AD application.
      v) Assigns the service principal ID of **App-Owns-Data App** as member of the Azure AD group **Power BI Apps**.
      vi) Opens a text file in Notepad with the configuration information you will need when you create your next C# console app.
   d) Run the PowerShell script named **RegisterAppOwnsDataApp.ps1**.
   e) The script should create the new confidential client application and display text with some configuration data you'll need later.

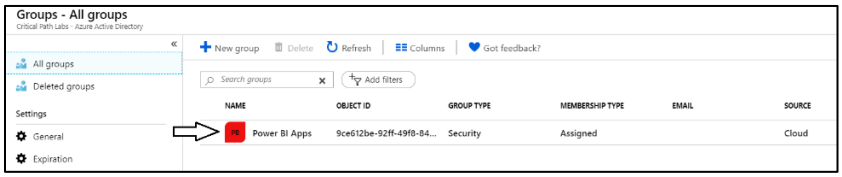


Leave this text file open as you will need to copy and paste the **ClientId** and **ClientSecret** into your application source code. You must also obtain the GUID-based ID values for the **Wingtip Sales** app workspace along with the IDs for a dataset, report and dashboard.

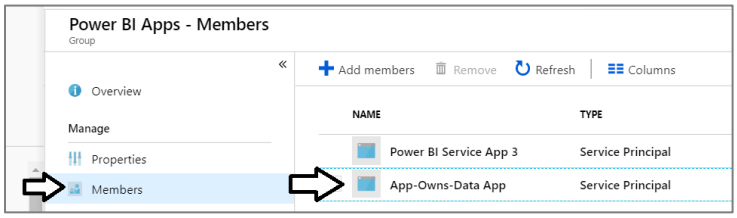
2. Inspect the group membership for **Power BI Apps** group.
   a) Navigate to the **All groups** blade of the Azure AD portal using the following URL.

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

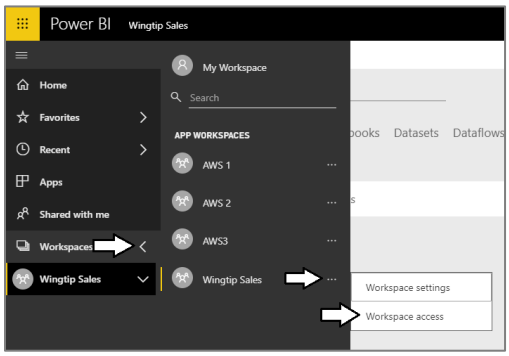
   b) Click on the link for the **Power BI Apps** group.



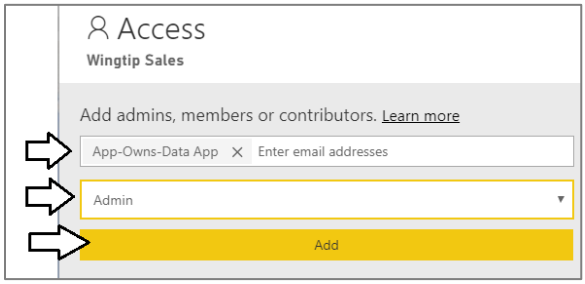
   c) Click the **Members** link on the left and verify that **App-Owns-Data Embedding App** has been added as a group member.



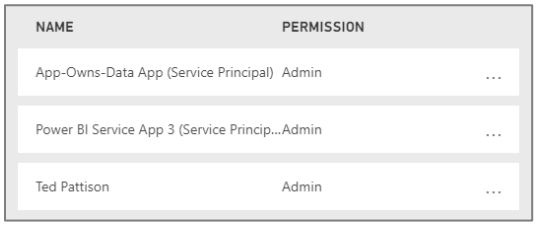
3. Configure the service principal for **App-Owns-Data App** as an admin for the app workspace named **Wingtip Sales**.
   a) Navigate to the Power BI portal.
   b) Expand the **Workspaces** flyout menu.
   c) Click the **Wingtip Sales** workspace context menu (**…**) and select **Workspace access**.



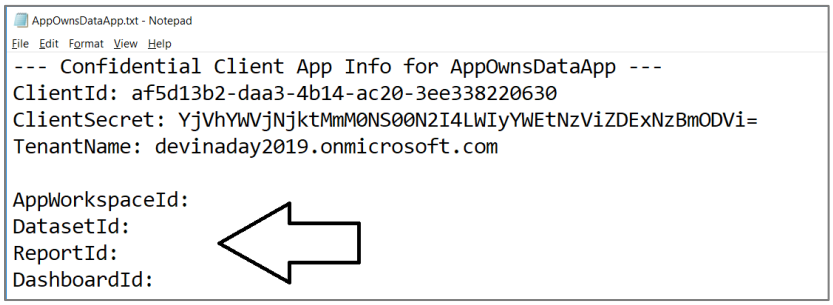
   d) On the right of the page, you should see the **Access** pane for the **Wingtip Sales** workspace.
   e) Place the cursor into the *Enter email address* textbox and type **App-Owns-Data App**.
   f) Change the member type from **Member** to **Admin**.
   g) Click to **Add** button.



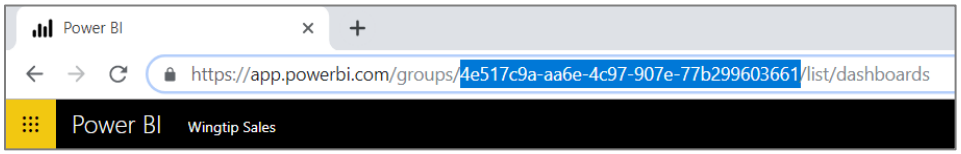
h) Verify that **App-Owns-Data App** has been added as a workspace member with **Admin** permissions.



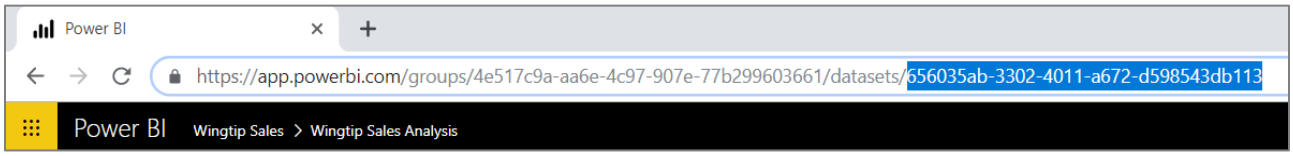
4. Gather the configuration data you will need for your Power BI embedding application.
   a) Return to Notepad and the text file where you added the **ClientID**, **ClientSecret and TenantName**.
   b) Locate the placeholders for **AppWorkspaceId**, **DatasetId**, **ReportId** and **DashboardId**.



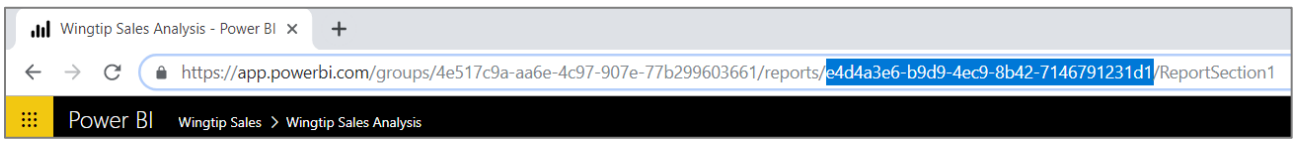
1. Retrieve the GUID-based IDs for the **Wingtip Sales** app workspace and the embeddable resources inside.
   a) Navigate to the **Wingtip Sales** app workspace in the Power BI portal.
   b) Locate and copy the app workspace ID from the URL by copying the GUID that comes after **/groups/**.



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



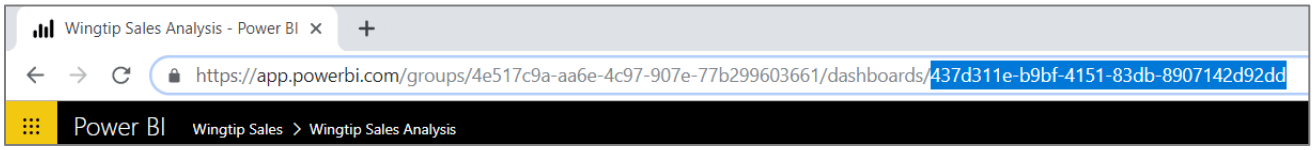
   f) Copy the dataset ID into the text file Notepad.
   g) Navigate back to the **Wingtip Sales Analysis** report inside the **Wingtip Sales** app workspace.
   h) Locate and copy the report ID from the URL by copying the GUID that comes after **/reports/**.



   i) Copy the report ID into the text file Notepad.

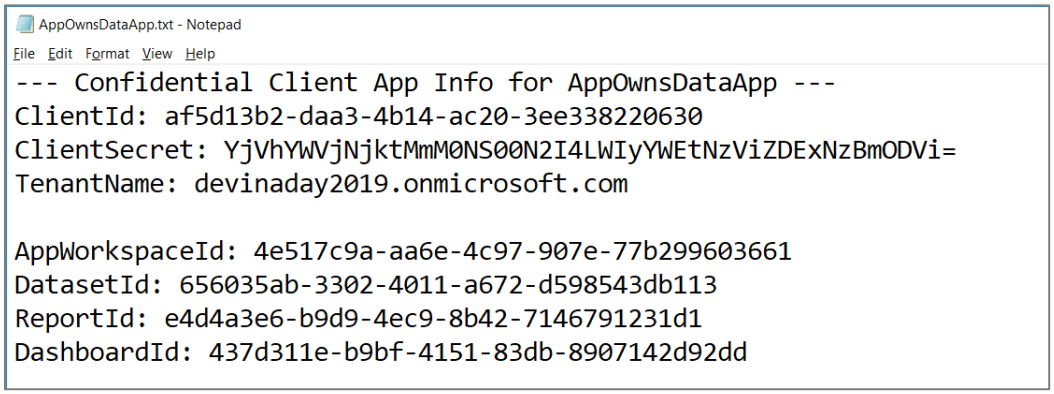
j) Navigate to the **Wingtip Sales Analysis** dashboard.

k) Locate and copy the dashboard ID from the URL by copying the GUID that comes after **/dashboards/**.



l) Copy the dashboard ID into the text file Notepad.

m) You should have now updated the text file Notepad with all the configuration data you need.



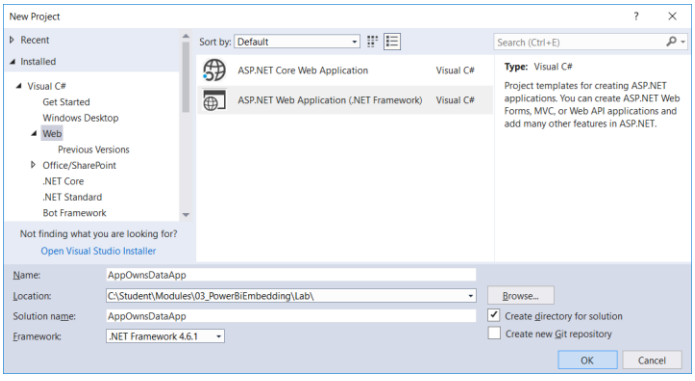
n) Save your changes to **AppOwnsDataApp.txt**.

o) Leave **AppOwnsDataApp.txt** open as you will need it when developing a new application in the next exercise.

Exercise 2: Create a new MVC Application using Visual Studio 2017

In this exercise you will create a new Web Application project using Visual Studio 2017 and the ASP.NET MVC framework.

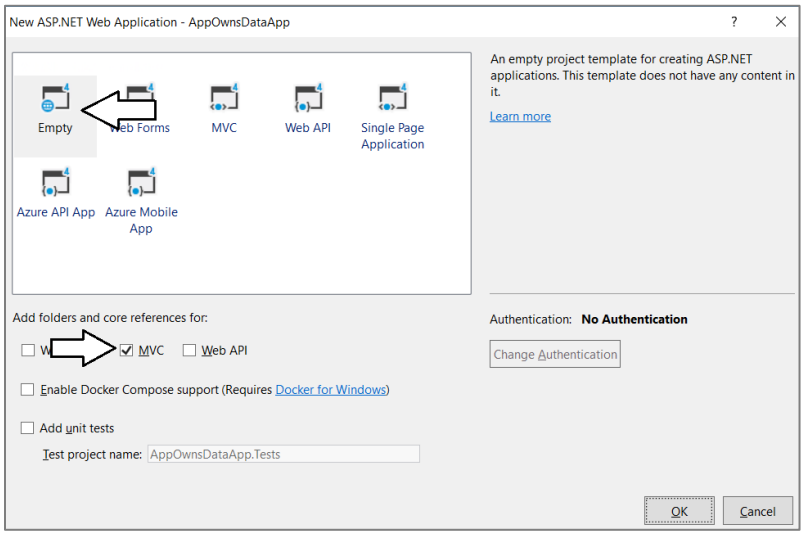
1. Launch **Visual Studio 2017**.
2. Create a new ASP.NET MVC project in Visual Studio 2017.
   a) In Visual Studio select **File > New > Project**.
   b) In the **New Project** dialog:
      i) Select **Installed > Templates > Visual C# > Web**.
      ii) Select the **ASP.NET Web Application** project template.
      iii) Name the new project **AppOwnsDataApp**.
      iv) Add the new project into the folder at **C:\Student\Modules\03_PowerBiEmbedding\Lab**.
      v) Click **OK** to display the **New ASP.Net Web Application** wizard.



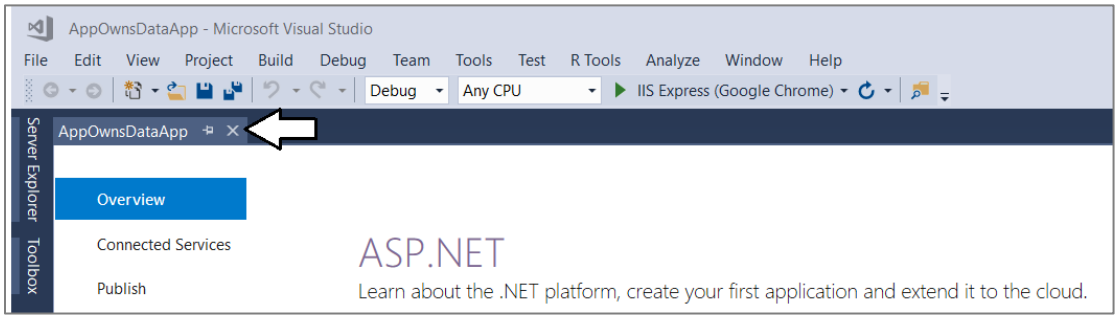
c) In the **New ASP.Net Web Application** dialog, select the **Empty** template.

d) In the section with the caption **Add folders and core references**, make sure the **MVC** checkbox is checked.

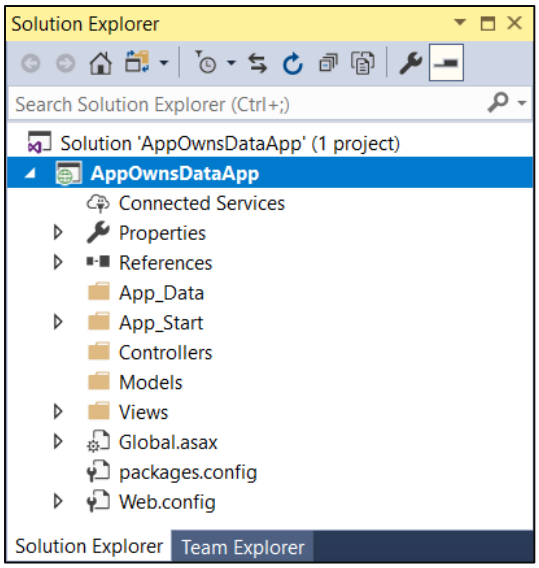
e) Click the **OK** button to create the new project.



f) When Visual Studio finishes creating the project, it displays an information page. Close this page by clicking the **x** in the tab.



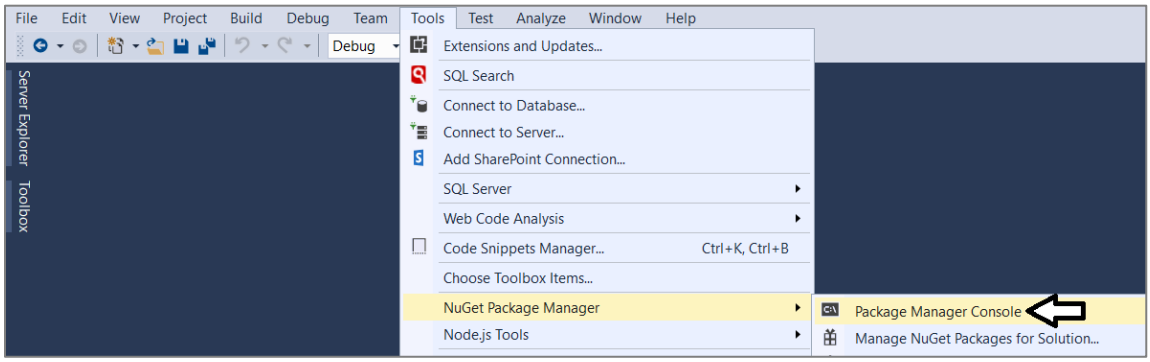
g) Take a minute to familiarize yourself with the structure of the **AppOwnsDataApp** project in the **Solution Explorer**.



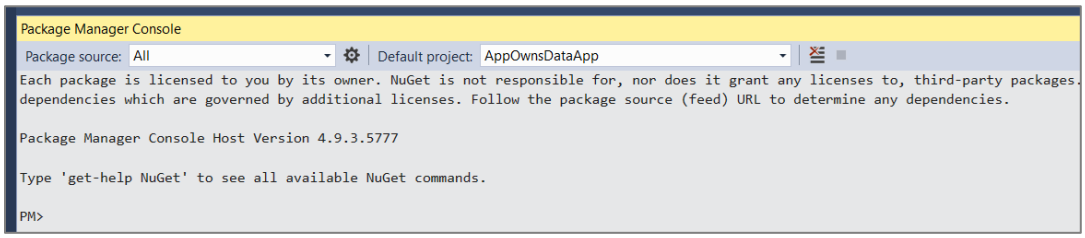
At this point, you have created a new ASP.NET MVC project based on the **Empty** project template. You will need to add an MVC controller and several MVC views before your application provides any type of user interface experience. Before adding a controller or writing any code, you will first update the project's NuGet packages included with your new project. You will also prepare for Power BI embedding by adding the NuGet package for the Azure Active Directory Authentication library (ADAL) and the NuGet packages for the Power BI .NET SDK and the Power BI JavaScript API.

3. Configure the **AppOwnsDataApp** project with the required set of NuGet packages

   a) From the Visual Studio menu, select the command **Tools > NuGet Package Manager > Package Manager Console**.



   b) You should now see the **Package Manage Console** with a `PM>` command prompt as shown in the following screenshot



   c) Type in and execute the following command to install the NuGet package for **jQuery**.

```
Install-Package jquery
```

   d) Type in and execute the following command to install the NuGet package for **bootstrap**.

```
Install-Package bootstrap
```

   e) Type in and execute the following command to install the NuGet package for the **Microsoft Authentication library (MSAL)**.

```
Install-Package Microsoft.Identity.Client
```

   f) Type in and execute the following command to install the NuGet package for the **Power BI .NET SDK**.

```
Install-Package Microsoft.PowerBI.Api
```

   g) Type in and execute the following command to install the NuGet package for the **Power BI JavaScript API**.

```
Install-Package Microsoft.PowerBI.JavaScript
```

   h) Type in and execute the following command to update the NuGet package for the **Newtonsoft.Json**.

```
Update-Package Newtonsoft.Json
```

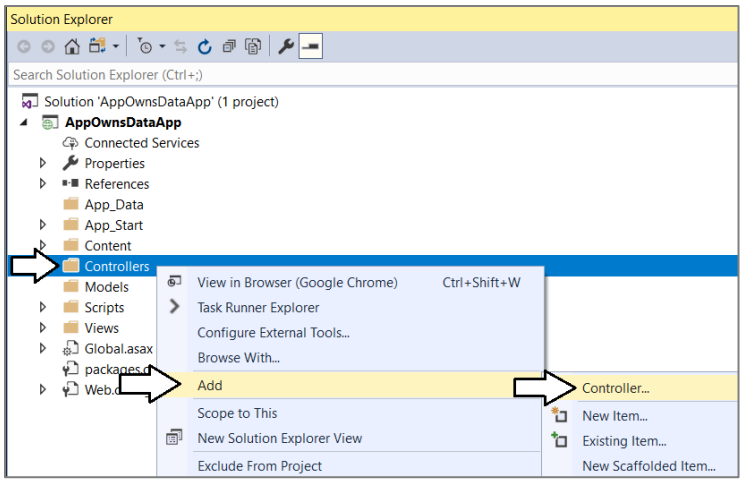
   i) Type in and execute the following command to update the NuGet package for the **Microsoft.Rest.ClientRuntime**.

```
Update-Package Microsoft.Rest.ClientRuntime
```

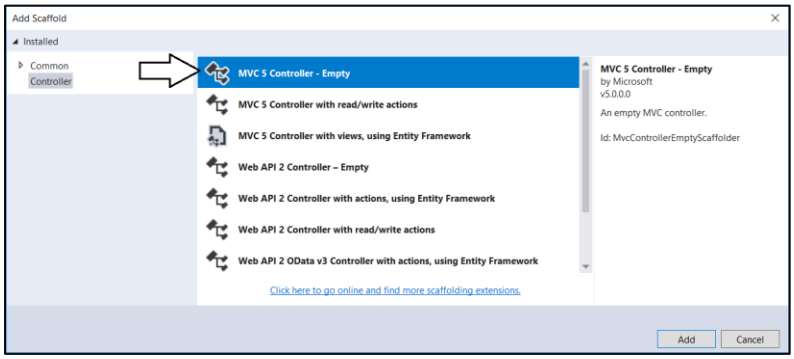
Now you have installed all the required NuGet packages for Power BI embedding.

4. Add the **HomeController** class.

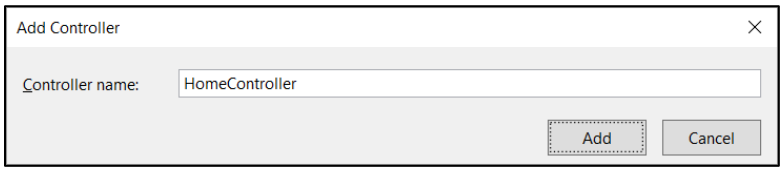
   a) In Solution Explorer of the **AppOwnsDataApp** project, right-click on the **Controllers** folder.



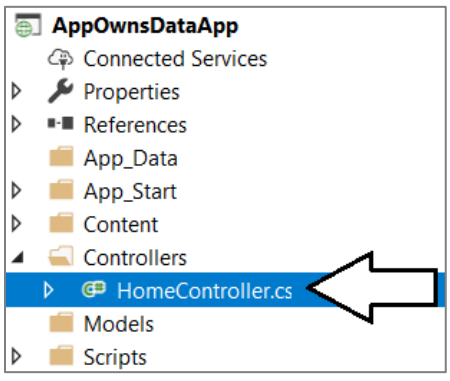
   b) In the **Add Scaffold** dialog, select the first option **MVC 5 Controller – Empty** and then click **Add**.



   c) In the **Add Controller** dialog, enter a **Controller name** of **HomeController** and then click **Add**.



   d) You should now see a new source file in the **Controllers** folder named **HomeController.cs**.

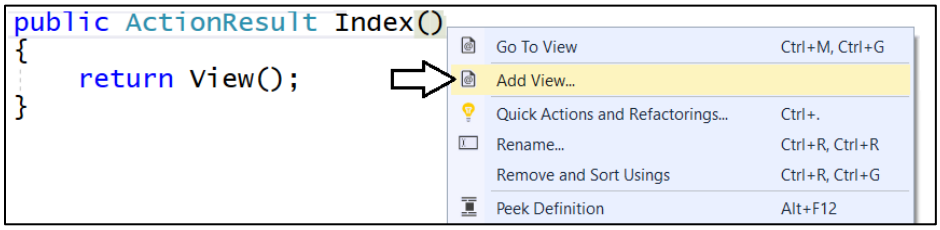


e) Inside **HomeController.cs**, you will find the starting point for the **HomeController** class with a single method named **Index**.

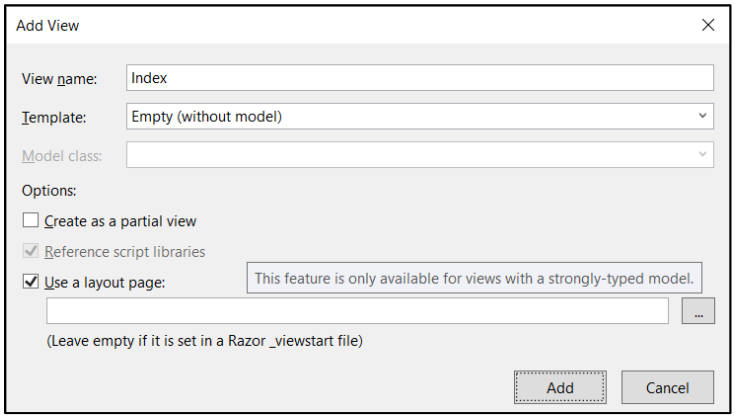
```
namespace EmbeddedLab.Controllers
{
    public class HomeController : Controller
    {
        // GET: Home
        public ActionResult Index()
        {
            return View();
        }
    }
}
```

5. Add a view for the **Index** action method of the **Home** controller class.

a) Inside **HomeController.cs**, right-click the **Index** method and select the **Add View…** command.



b) In the **Add View** dialog, accept all the default settings as shown in the following screenshot and click **Add**.

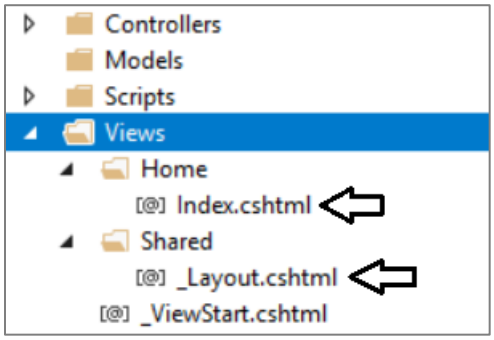


When you create a new view and leave the **Use a layout page** option selected, a new shared layout page named **_Layouts.cshtml** is automatically added to the project in the **Views/Shared** folder.

c) In Solution Explorer, you should be able to verify that your project contains two new files.

i) Inside the **Views/Home** folder there is a new razor view file named **Index.cshtml**.

ii) Inside the **Views/Shared** folder there is a new shared layout page named **_Layouts.cshtml**.



d) Examine the code that has been added to **Index.cshtml**.

e) Delete all the code inside **Index.cshtml** and replace it with the following HTML code.

```
<div id="homePageContainer" class="container" >

  <div class="jumbotron">
    <h2>Welcome to the App-Owns-Data App</h2>
  </div>

</div>
```

f) Save your changes and close **Index.cshtml**.

Over the next few steps, you will add the HTML code for a shared layouts page into **_Layout.cshtml** in a sequence of several different copy-and-paste operations. If you'd rather copy and paste the all the code for **_Layout.cshtml** at once, you can find the completed HTML code inside a file named **Layout.cshtml.txt** located in the **C:\Student\Modules\03_PowerBiEmbedding\Lab\Snippets** folder.

6. Modify the shared layouts page named **_Layouts.cshtml**.

a) In Solution Explorer, expand the **Views** folder and then expand the **Shared** folder.

b) Double-click on **_Layouts.cshtml** to open it in an editor window.

c) Delete the entire contents of **_Layouts.cshtml** and replace it with the following HTML starter page.

```
<!DOCTYPE html>
<html>

<head>
</head>

<body>
</body>

</html>
```

d) Copy and paste the following HTML code to provide the **head** section

```
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>App-Owns-Data App</title>
  <link href="~/Content/bootstrap.css" rel="stylesheet" />
  <link href="~/Content/Site.css" rel="stylesheet" />
  <script src="~/Scripts/jquery-3.4.1.js"></script>
  <script src="~/Scripts/bootstrap.js"></script>
</head>
```

Make sure your script link to jQuery matches the version number of the jQuery library source file in the **Scripts** folder. The code listing above shows version **jquery-3.4.1.js** but your project might have a more recent version with a different version number.

e) Copy and paste the following HTML code to provide the **body** section of the page.

```
<body>

  <!-- Add Banner with TopNav and Toolbar Here -->

  <!-- Add Main Body Content Here -->

  <!-- Add JavaScript Code to Resize Page Elements Here -->

</body>
```

Now you will copy and paste HTML markup code into each of the three sections in the HTML **body** element.

f) Copy and paste the following code into the body just below the ***Add Banner with TopNav and Toolbar Here*** comment.

```
<!-- Add Banner with TopNav and Toolbar Here -->

<div id="banner" class="container">
  <nav id="topnav" class="navbar navbar-expand-sm navbar-dark bg-dark">
    <ul class="navbar-nav">
      <li class="nav-item active d-none d-md-block">
        @Html.ActionLink("App-Owns-Data App", "Index", "Home",
                         routeValues: null, htmlAttributes: new { @class = "nav-link navbar-brand" })
      </li>
      <li class="nav-item">
        @Html.ActionLink("Report", "Report", "Home",
                        routeValues: null, htmlAttributes: new { @class = "nav-link" })
      </li>
      <li class="nav-item">
        @Html.ActionLink("Dashboard", "Dashboard", "Home",
                        routeValues: null, htmlAttributes: new { @class = "nav-link" })
      </li>
      <li class="nav-item">
        @Html.ActionLink("Q&A", "Qna", "Home",
                        routeValues: null, htmlAttributes: new { @class = "nav-link" })
      </li>
      <li class="nav-item">
        @Html.ActionLink("New Report", "NewReport", "Home",
                        routeValues: null, htmlAttributes: new { @class = "nav-link" })
      </li>
    </ul>
  </nav>
  @RenderSection("toolbar", required: false)
</div>
```

g) Copy and paste the following code into the body just below the **Add Main Body Content Here** comment

```
<!-- Add Main Body Content Here -->

<div id="content-box" class="container body-content">
  @RenderBody()
</div>
```

h) Copy and paste the following code into the body just below the **Add Main Body Content Here** comment

```
<!-- Add JavaScript Code to Resize Page Elements -->

<script>
  $(function () {
    var heightBuffer = 12;
    var newHeight = $(window).height() - ($("#banner").height() + heightBuffer);
    $("#content-box").height(newHeight);
    $("#embedContainer").height(newHeight);
    $(window).resize(function () {
      var newHeight = $(window).height() - ($("#banner").height() + heightBuffer);
      $("#content-box").height(newHeight);
      $("#embedContainer").height(newHeight);
    });
  });
</script>
```

i) Save your changes and close _**Layouts.cshtml**.

7. Modify the **Sites.css** file with a set of custom CSS styles.

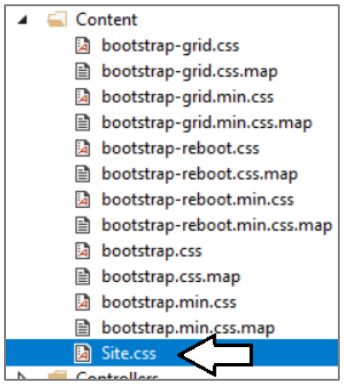
a) Using Windows Explorer, locate the snippet file named **Site.css.txt** in the **Students** at the following location.

```
C:\Student\Modules\03_PowerBiEmbedding\Lab\Snippets\Site.css.txt
```

b) Double click on **Site.css.txt** to open it in Notepad.

c) Select all the CSS code inside **Site.css.txt,** copy it to the Windows clipboard and return to Visual Studio.

d) In Solution Explorer, expand the **Content** folder and then double-click on **Sites.css** open it in an editor window.



e) Delete all the existing content from **Sites.css** and paste in the content from the Windows clipboard.

f) Save your changes and close **Sites.css**.

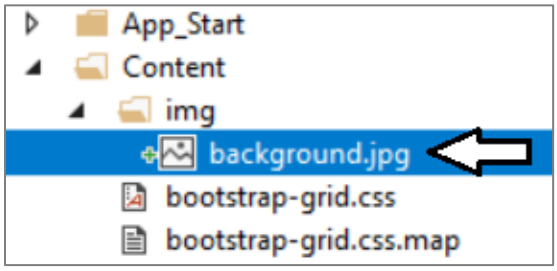
8. Add a new image named **background.jpg** to the project to provide a page background.

a) Using Windows Explorer, locate the file named **background.jpg** in the **Students** folder at the following location.

```
C:\Student\Modules\03_PowerBiEmbedding\Lab\StarterFiles\background.jpg
```

b) In Solution Explorer, create a new folder named **img** inside the **Contents** folder.

c) Copy the file **background.jpg** into the **img** folder.

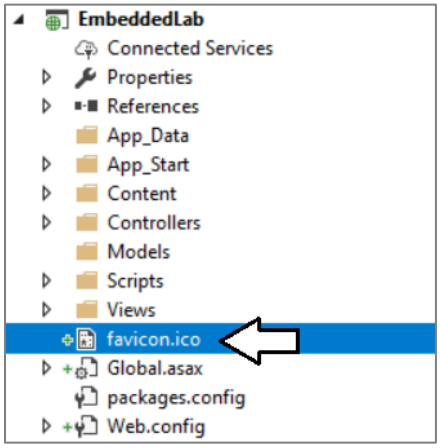


9. Add a **favicon.ico** file to the root folder of the **AppOwnsDataApp** project.

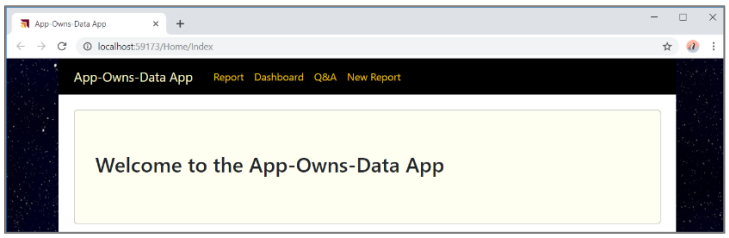
a) Using Windows Explorer, locate the file named **favicon.icon** in the **Students** folder at the following location.

```
C:\Student\Modules\03_PowerBiEmbedding\Lab\StarterFiles\favicon.ico
```

b) Copy the file named **favicon.ico** to the root folder of your project.



10. Test out the **AppOwnsDataApp** project using the Visual Studio debugger
    a) Press the **{F5}** key to start up the project in the Visual Studio debugger.
    b) When the project starts, the home page should load in the browser and match the following screenshot.



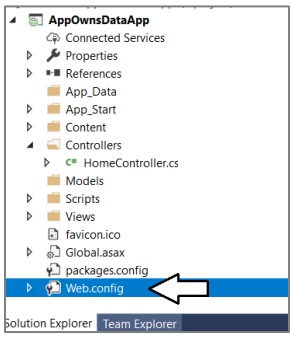
    c) Close the browser, return to Visual Studio and stop the debugger.

Note that the navigation links on the top navigation menu are not working yet. Over the next few exercises, you will add MVC action methods and razor views to implement Power BI embedding behavior behind each of these navigation links.

Exercise 3: Embedding Your First Power BI Report

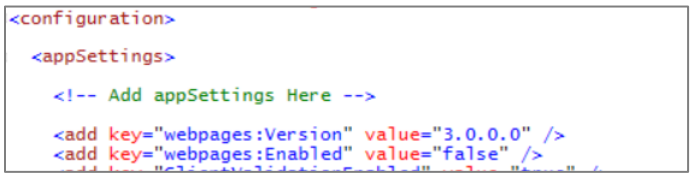
In this exercise, you will create an action method and a razor view named **Report** which will embed a Power BI report.

1. Modify the project's **web.config** file to add **appSetting** values for the required configuration data.
    a) Open the **web.config** file located at the root of the **AppOwnsDataApp** project.



Make sure you open the **web.config** file located at the root of the project and not the **web.config** file inside the **Views** folder.

    b) Locate the **<appSettings>** element at the top of **web.config**.
    c) Add a few blank lines just after the **<appSettings>** element opening tag.



```
<configuration>
  <appSettings>

    <!-- Add appSettings Here -->

    <add key="webpages:Version" value="3.0.0.0" />
    <add key="webpages:Enabled" value="false" />
```

    d) Copy and paste the following XML code into the **web.config** file underneath the **<appSettings>** opening tag.

```
<add key="client-id" value="" />
<add key="client-secret" value="" />
<add key="tenant-name" value="" />

<add key="app-workspace-id" value="" />
<add key="dataset-id" value="" />
<add key="report-id" value="" />
<add key="dashboard-id" value="" />
```

e) Copy the seven configuration values from **AppOwnsDataApp.txt** into the new **appSetting** values in **web.config.**

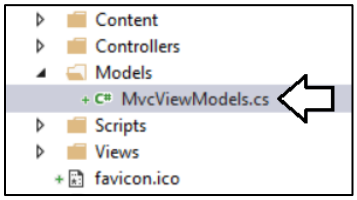
f) You should be able to supply values for each of the seven **appSetting** values as shown in the following screenshot.

```
<configuration>
  <appSettings>
    <add key="webpages:Version" value="3.0.0.0" />
    <add key="webpages:Enabled" value="false" />
    <add key="ClientValidationEnabled" value="true" />
    <add key="UnobtrusiveJavaScriptEnabled" value="true" />

    <add key="client-id" value="aa2886d8-15a2-4ed5-92f1-c0244e004175" />
    <add key="client-secret" value="ZGMxN2QxMjYtMzczMC00MDY2LWJmNTctYzM5Y2ZmOWFhODFl=" />
    <add key="tenant-name" value="pbi0520.onmicrosoft.com" />

    <add key="app-workspace-id" value="7186f418-2a89-4efc-a8fc-b184ad353c70" />
    <add key="dataset-id" value="cbed1d20-4452-419f-b17c-76e7d4aed203" />
    <add key="report-id" value="7455ec66-954c-41b7-a5d6-786df8ed4f8a" />
    <add key="dashboard-id" value="55c4ff7f-bb84-4303-874a-d3631a5a1fc7" />

  </appSettings>
```

g) Save your changes and close **web.config**.

2. Create classes to provide MVC view models for Power BI embedding data.

a) Add a new C# source file named **MvcViewModels.cs** inside the **Models** folder.

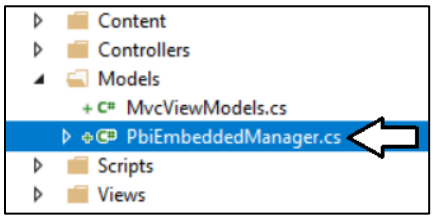


b) If there is any code inside **MvcViewModels.cs**, delete it and replace it with the following code.

```
namespace AppOwnsDataApp.Models {

  // data required for embedding a report
  public class ReportEmbeddingData {
    public string reportId;
    public string reportName;
    public string embedUrl;
    public string accessToken;
  }

  // data required for embedding a new report
  public class NewReportEmbeddingData {
    public string workspaceId;
    public string datasetId;
    public string embedUrl;
    public string accessToken;
  }

  // data required for embedding a dashboard
  public class DashboardEmbeddingData {
    public string dashboardId;
    public string dashboardName;
    public string embedUrl;
    public string accessToken;
  }

  // data required for embedding a dashboard
  public class QnaEmbeddingData {
    public string datasetId;
    public string embedUrl;
    public string accessToken;
  }

}
```

c) Save your changes and close **MvcViewModels.cs**.

3. Create the **PbiEmbeddingManager** class.
   a) Add a new class named **PbiEmbeddingManager** inside the **Models** folder.
   b) The **Models** folder should now contain a C# source file named **PbiEmbeddingManager.cs**.



   c) Delete any code inside **PbiEmbeddingManager.cs** and replace it with the following starter code.

```
using System;
using System.Configuration;
using System.Threading.Tasks;
using Microsoft.Identity.Client;
using Microsoft.PowerBI.Api.V2;
using Microsoft.PowerBI.Api.V2.Models;
using Microsoft.Rest;

namespace AppOwnsDataApp.Models {

  public class PbiEmbeddingManager {
  }

}
```

   d) Modify the **PbiEmbeddingManager** class by adding the following set of static fields.

```
class PbiEmbeddingManager {

  private static readonly string clientId = ConfigurationManager.AppSettings["client-id"];
  private static readonly string clientSecret = ConfigurationManager.AppSettings["client-secret"];
  private static readonly string tenantName = ConfigurationManager.AppSettings["tenant-name"];

  private static readonly string workspaceId = ConfigurationManager.AppSettings["app-workspace-id"];
  private static readonly string datasetId = ConfigurationManager.AppSettings["dataset-id"];
  private static readonly string reportId = ConfigurationManager.AppSettings["report-id"];
  private static readonly string dashboardId = ConfigurationManager.AppSettings["dashboard-id"];

  // endpoint for tenant-specific authority
  private static readonly string tenantAuthority = "https://login.microsoftonline.com/" + tenantName;

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

}
```

In addition to the seven fields with configuration data, there are two other fields named **tenantAuthority** and **urlPowerBiRestApiRoot** which are used to authenticate with Azure AD and to determine the endpoint for the Power BI Service API.

   e) At the bottom of **PbiEmbeddingManager** class, add a method named **GetAppOnlyAccessToken** using the following code.

```
static string GetAppOnlyAccessToken() {

  var appConfidential = ConfidentialClientApplicationBuilder.Create(clientId)
                        .WithClientSecret(clientSecret)
                        .WithAuthority(tenantAuthority)
                        .Build();

  string[] scopesDefault = new string[] { "https://analysis.windows.net/powerbi/api/.default" };
  var authResult = appConfidential.AcquireTokenForClient(scopesDefault).ExecuteAsync().Result;
  return authResult.AccessToken;
}
```

f) Underneath the **GetAppOnlyAccessToken** method, add a new method named **GetPowerBiClient** using the following code.

```
private static PowerBIClient GetPowerBiClient() {
  var tokenCredentials = new TokenCredentials(GetAppOnlyAccessToken(), "Bearer");
  return new PowerBIClient(new Uri(urlPowerBiRestApiRoot), tokenCredentials);
}
```

You have implemented the essential behavior in the **PbiEmbeddingManager** class to authenticate with Azure AD and to create new **PowerBIClient** objects which represents the top-level entry point into the Power BI Service API when using the Power BI .NET SDK. Now you are at a point where you can add methods to the **PbiEmbeddingManager** class to retrieve the data required for embedding.

4. Add the **GetReportEmbeddingData** method to the **PbiEmbeddingManager** class.

a) At the bottom of the **PbiEmbeddingManager** class, add the **GetReportEmbeddingData** method using the following code.

```
public static async Task<ReportEmbeddingData> GetReportEmbeddingData() {

  PowerBIClient pbiClient = GetPowerBiClient();

  var report = await pbiClient.Reports.GetReportInGroupAsync(workspaceId, reportId);
  var embedUrl = report.EmbedUrl;
  var reportName = report.Name;

  GenerateTokenRequest generateTokenRequestParameters = new GenerateTokenRequest(accessLevel: "edit");
  string embedToken =
        (await pbiClient.Reports.GenerateTokenInGroupAsync(workspaceId,
                                                           report.Id,
                                                           generateTokenRequestParameters)).Token;

  return new ReportEmbeddingData {
    reportId = reportId,
    reportName = reportName,
    embedUrl = embedUrl,
    accessToken = embedToken
  };

}
```

b) Save your changes to **PbiEmbeddedManger.cs**.

Now that you have added the **GetReportEmbeddingData** method, you will create a new action method that calls this method.

5. Add the **Report** action method to the **HomeController** class.

a) Inside the **Controllers** folder, open the C# source file named **HomeController.cs**.

b) Update the set of **using** statements at the top of **HomeController.cs** using the following code.

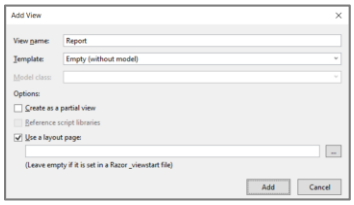
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Web;
using System.Web.Mvc;
using AppOwnsDataApp.Models;
```

c) Underneath the **Index** method, add a new asynchronous action method named **Report** using the following code.

```
public class HomeController : Controller {

  public ActionResult Index() {
    return View();
  }

  public async Task<ActionResult> Report() {
    ReportEmbeddingData embeddingData = await PbiEmbeddingManager.GetReportEmbeddingData();
    return View(embeddingData);
  }

}
```

d) Right-click on the **Report** method and select the **Add View…** command from the context menu.

e) In the **Add View** dialog, accept all the default settings and click the **Add** button.



f) You should be able to verify that a new razor view file named **Report.cshtml** has been created in the **Views/Home** folder.

g) Delete all existing content from **Report.cshtml** and replace it with the following code.

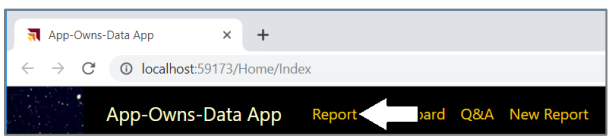
```
@model AppOwnsDataApp.Models.ReportEmbeddingData

<div id="embedContainer" />

<script src="~/Scripts/powerbi.js"></script>

<script>

  // data required for embedding Power BI report
  var embedReportId = "@Model.reportId";
  var embedUrl = "@Model.embedUrl";
  var accessToken = "@Model.accessToken";

  // Get models object to access enums for embed configuration
  var models = window['powerbi-client'].models;

  var config = {
    type: 'report',
    id: embedReportId,
    embedUrl: embedUrl,
    accessToken: accessToken,
    permissions: models.Permissions.All,
    tokenType: models.TokenType.Embed,
    viewMode: models.ViewMode.View,
    settings: {
      filterPaneEnabled: false,
      navContentPaneEnabled: true,
    }
  };

  // Get a reference to HTML element that will be embed container
  var reportContainer = document.getElementById('embedContainer');

  // Embed the report and display it within the div container.
  var report = powerbi.embed(reportContainer, config);

</script>
```

h) Save your changes and close **Report.cshtml**.

6. Test out the application by running it in the Visual Studio debugger.

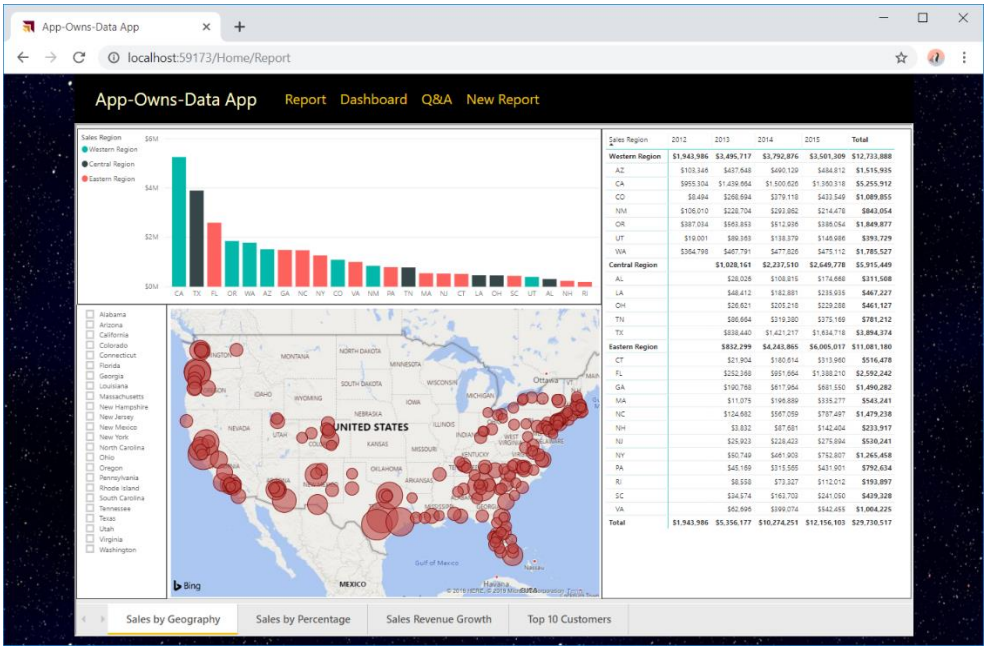
a) Press the **{F5}** key in Visual Studio to begin a new debugging session.

b) Click the **Report** link in the top navigation menu.



If the editor window with a razor view such as **Report.cshtml** is the active window when you press the **{F5}**, the Visual Studio debugger will automatically take you to this view at the start of your debugging session.

c) You should now see the **Wingtip Sales Analysis** report has been embedded on the page for the **Report** view.



Try resizing the browser window. You will see that your application responds by dynamically changing the size of the HTML element with the ID of **embedContainer** and the embedded report responds automatically by changing its size to fit the new dimensions.

d) Close the browser window, return to Visual Studio and stop the current debugging session.

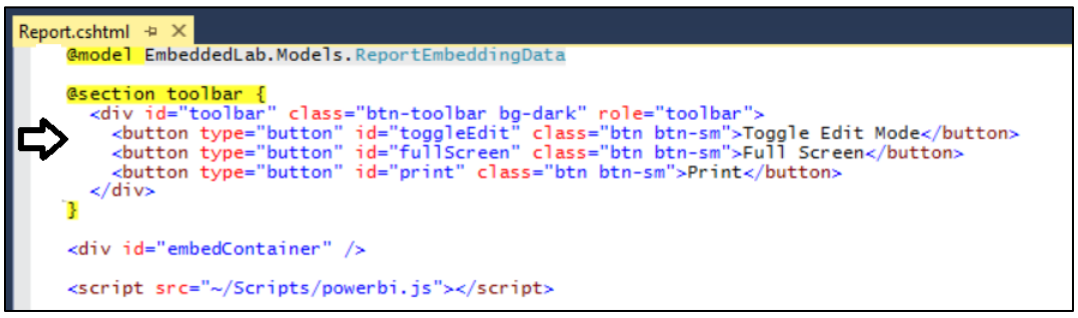
Exercise 4: Adding a Toolbar for Embedded Reports

In this exercise, you continue to work on **Report.cshtml** by adding a new toolbar with three command buttons. You will also add JavaScript code behind these command buttons to invoke actions on the embedded report.

1. Add the HTML layout code for a toolbar into **Report.cshtml**.
   a) Open **Report.cshtml** if it is not already open.
   b) Copy and paste the following HTML code into **Report.cshtml** just below the **@model** directive at the top.

```
@section toolbar {
  <div id="toolbar" class="btn-toolbar bg-dark" role="toolbar" >
    <button type="button" id="toggleEdit" class="btn btn-sm" >Toggle Edit Mode</button>
    <button type="button" id="fullScreen" class="btn btn-sm" >Full Screen</button>
    <button type="button" id="print" class="btn btn-sm" >Print</button>
  </div>
}
```

   c) The top of **Report.cshtml** should match the following screenshot.



d) Inside **Report.cshtml**, move down inside **<script>** block and add a few new lines after the line which calls **powerbi.embed**.

e) Copy and paste the following JavaScript code at the bottom of the **<script>** block just before the closing **</script>** tag.

```
var viewMode = "view";

$("#toggleEdit").click(function () {
  // toggle between view and edit mode
  viewMode = (viewMode == "view") ? "edit" : "view";
  report.switchMode(viewMode);
  // show filter pane when entering edit mode
  var showFilterPane = (viewMode == "edit");
  report.updateSettings({
    "filterPaneEnabled": showFilterPane
  });
});

$("#fullScreen").click(function () {
  report.fullscreen();
});

$("#print").click(function () {
  report.print();
});
```

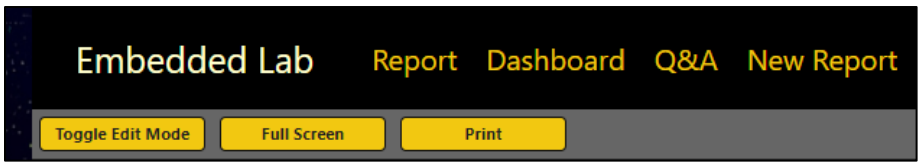
f) Save your changes to **Report.cshtml**.

2. Test out the application by running it in the Visual Studio debugger.

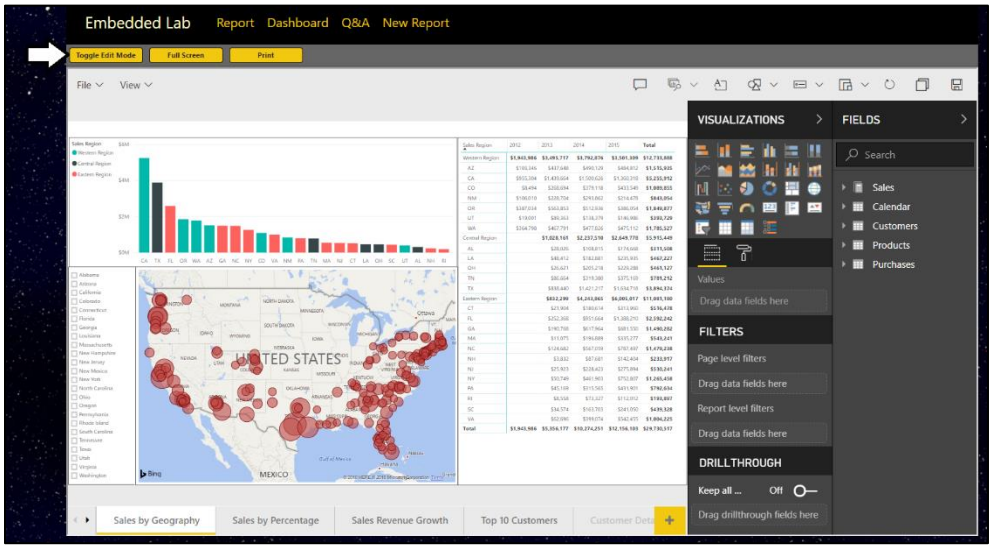
a) Press the **{F5}** key in Visual Studio to begin a new debugging session.

b) Click the **Report** link in the top navigation menu.

c) You should now see three toolbar buttons with the captions **Toggle Edit Mode**, **Full Screen** and **Print**.



d) Click the **Toggle Edit Mode** button several times. The report should toggle back and forth between edit and reader mode.



e) Experiment by clicking the **Full Screen** button.

f) Experiment by clicking the **Print button**.

g) Close the browser window and return to Visual Studio and stop the current debugging session.

Exercise 5: Embedding a Dashboard

In this exercise you will embed a dashboard. As you will see, it's not very different from the steps you have already implemented to embed a report.

1. Add a new method to the **PbiEmbeddingManger** class named **GetDashboardEmbeddingData**.
   a) Open **PbiEmbeddingManager.cs** in an editor window if it's not already open.
   b) Navigate to the bottom of the class definition just beneath the **GetReportEmbeddingData** method
   c) Paste in the definition for a new method named **GetDashboardEmbeddingData** using the following code.

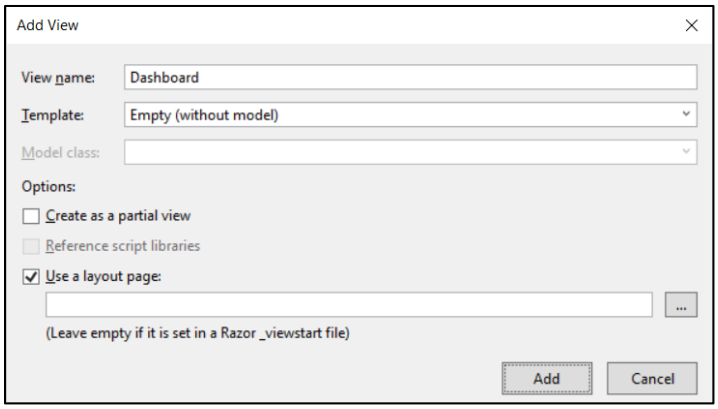
```
public static async Task<DashboardEmbeddingData> GetDashboardEmbeddingData() {

  PowerBIClient pbiClient = GetPowerBiClient();

  var dashboard = await pbiClient.Dashboards.GetDashboardInGroupAsync(workspaceId, dashboardId);
  var embedUrl = dashboard.EmbedUrl;
  var dashboardDisplayName = dashboard.DisplayName;

  GenerateTokenRequest generateTokenRequestParameters = new GenerateTokenRequest(accessLevel: "view");

  string embedToken =
     (await pbiClient.Dashboards.GenerateTokenInGroupAsync(workspaceId,
                                                           dashboardId,
                                                           generateTokenRequestParameters)).Token;

  return new DashboardEmbeddingData {
    dashboardId = dashboardId,
    dashboardName = dashboardDisplayName,
    embedUrl = embedUrl,
    accessToken = embedToken
  };

}
```

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

2. Add a new action method to the **HomeController** class named **Dashboard**.
   a) Open **HomeController.cs** in an editor window if it's not already open.
   b) Add a new action method named **Dashboard** just beneath the **Report** method using the following code.

```
public async Task<ActionResult> Dashboard() {
  DashboardEmbeddingData embeddingData = await PbiEmbeddingManager.GetDashboardEmbeddingData();
  return View(embeddingData);
}
```

3. Create a razor view for the **Dashboard** action method.
   a) Right-click on the **Dashboard** action method and select the **Add View…** command from the context menu.
   b) In the **Add View** dialog, accept all the default settings and click the **Add** button.



c) You should see that a new razor view file named **Dashboard.cshtml** has been created in the **Views/Home** folder.

d) Delete any existing code inside **Dashboard.cshtml** and replace it with the following HTML code.

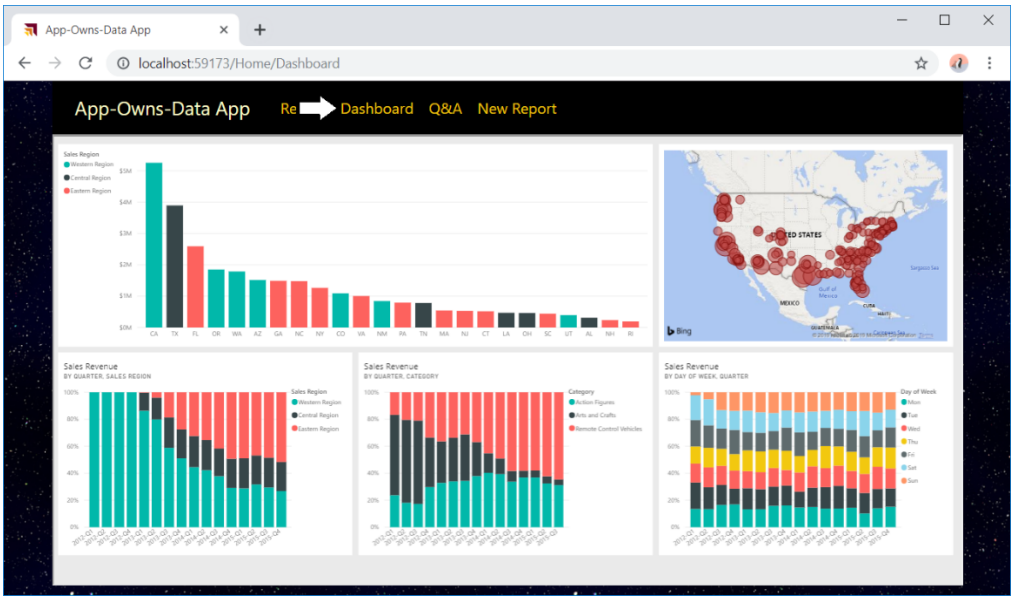
```
@model AppOwnsDataApp.Models.DashboardEmbeddingData

<div id="embedContainer" />

<script src="~/Scripts/powerbi.js"></script>
<script>

  // data required for embedding Power BI report
  var embedDashboardId = "@Model.dashboardId";
  var embedUrl = "@Model.embedUrl";
  var accessToken = "@Model.accessToken";

  // Get models object to access enums for embed configuration
  var models = window['powerbi-client'].models;

  var config = {
    type: 'dashboard',
    id: embedDashboardId,
    embedUrl: embedUrl,
    accessToken: accessToken,
    tokenType: models.TokenType.Embed,
    pageView: "fitToWidth"
  };

  // Get a reference to the embedded report HTML element
  var embedContainer = document.getElementById('embedContainer');

  // Embed the report and display it within the div container.
  var dashboard = powerbi.embed(embedContainer, config);

</script>
```

e) Save your changes to **Dashboard.cshtml**.

4. Test out the application by running it in the Visual Studio debugger.

a) Press the **{F5}** key in Visual Studio to begin a new debugging session.

b) Click the **Dashboard** link in the top navigation menu and you should see the dashboard embedded in the web page.



c) Try changing the size of the browser window and see how the application responds by adjusting the size of the dashboard.

d) Close the browser window and return to Visual Studio and stop the current debugging session.

Exercise 6: Embedding the Power BI Q&A Experience

In this exercise you will embed the Power BI Q&A experience. To accomplish this, you will be required to provide the dataset ID associated with the dataset on which you want to execute natural language queries.

1. Add a new method to the **PbiEmbeddingManger** class named **GetQnaEmbeddingData**.
   a) Open **PbiEmbeddingManager.cs** in an editor if it's not already open.
   b) Navigate to the bottom of the class definition just beneath the **GetDashboardEmbeddingData** method
   c) Add a new method named **GetQnaEmbeddingData** by copying and pasting the following code.

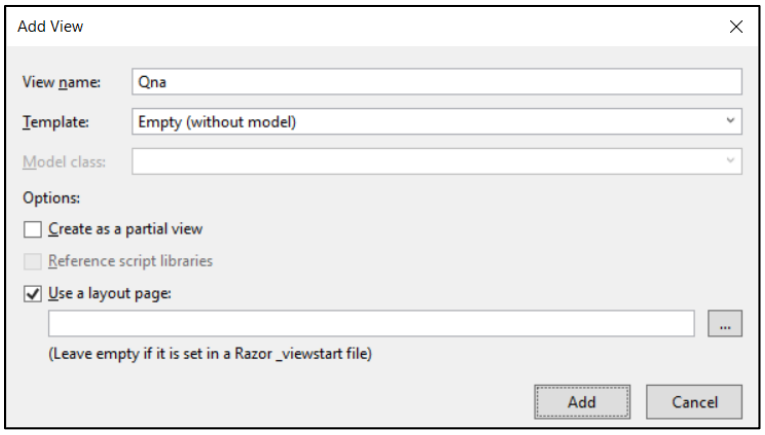
```
public async static Task<QnaEmbeddingData> GetQnaEmbeddingData() {

  PowerBIClient pbiClient = GetPowerBiClient();

  var dataset = await pbiClient.Datasets.GetDatasetByIdInGroupAsync(workspaceId, datasetId);

  string embedUrl = "https://app.powerbi.com/qnaEmbed?groupId=" + workspaceId;

  GenerateTokenRequest generateTokenRequestParameters = new GenerateTokenRequest(accessLevel: "view");
  string embedToken =
         (await pbiClient.Datasets.GenerateTokenInGroupAsync(workspaceId,
                                                             dataset.Id,
                                                             generateTokenRequestParameters)).Token;

  return new QnaEmbeddingData {
    datasetId = datasetId,
    embedUrl = embedUrl,
    accessToken = embedToken
  };

}
```

   d) Save your changes to Open **PbiEmbeddingManager.cs**.

2. Add a new action method to the **HomeController** class named **Qna**.
   a) Open **HomeController.cs** in an editor window if it's not already open.
   b) Add a new action method named **Qna** just beneath the **Dashboard** method using the following code.

```
public async Task<ActionResult> Qna() {
  QnaEmbeddingData embeddingData = await PbiEmbeddingManager .GetQnaEmbeddingData();
  return View(embeddingData);
}
```

3. Create a razor view for the **Qna** action method.
   a) Right-click on the **Qna** action method and select the **Add View…** command from the context menu.
   b) In the **Add View** dialog, accept all the default settings and click the **Add** button.



   c) You should see that a new razor view file named **Qna.cshtml** has been created in the **Views/Home** folder.

d) Delete any existing code inside **Qna.cshtml** and replace it with the following HTML code.

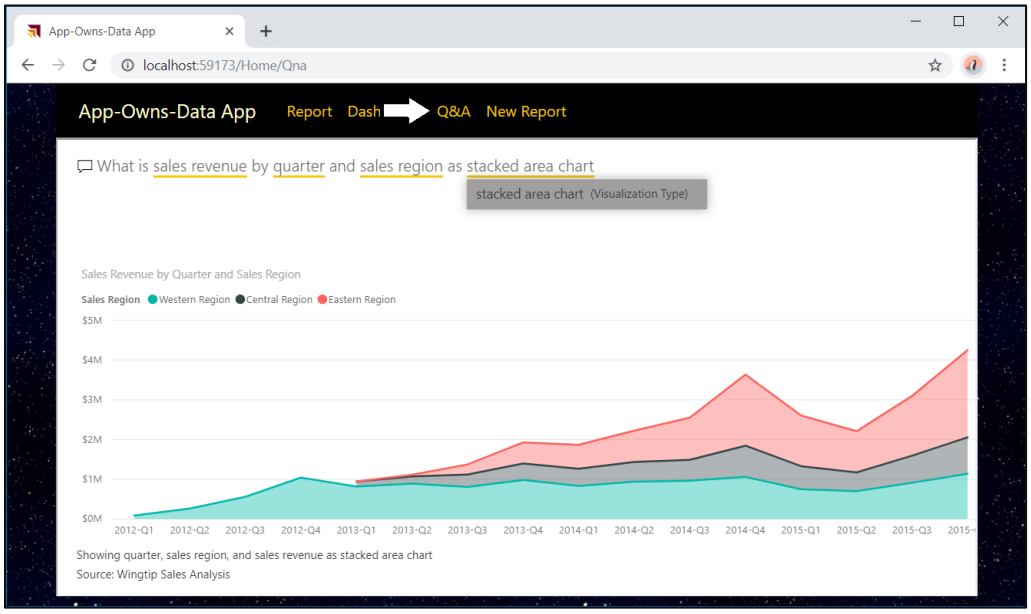
```
@model AppOwnsDataApp.Models.QnaEmbeddingData

<div id="embedContainer" />

<script src="~/Scripts/powerbi.js"></script>
<script>

  // Get data required for embedding
  var datasetId = "@Model.datasetId";
  var embedUrl = "@Model.embedUrl";
  var accessToken = "@Model.accessToken";

  // Get models object to access enums for embed configuration
  var models = window['powerbi-client'].models;

  var config = {
    type: 'qna',
    tokenType: models.TokenType.Embed,
    accessToken: accessToken,
    embedUrl: embedUrl,
    datasetIds: [datasetId],
    viewMode: models.QnaMode.Interactive,
    question: "What is sales revenue by quarter and sales region as stacked area chart"
  };

  // Get a reference to the embedded report HTML element
  var embedContainer = document.getElementById('embedContainer');

  // Embed the report and display it within the div container.
  var embeddedObject = powerbi.embed(embedContainer, config);

</script>
```

e) Save your changes to **Qna.cshtml**.

4. Test out the application by running it in the Visual Studio debugger.

a) Press the **{F5}** key in Visual Studio to begin a new debugging session.

b) Click the **Q&A** link in the top navigation menu and you should see the Q&A experience embedded in the web page.



c) Experiment by typing questions in English and seeing how the Q&A experience responds with data and visualizations.

d) Close the browser window, return to Visual Studio and stop the current debugging session.

Exercise 7: Embedding a New Report

In this exercise you will implement the behavior to embed a new report based on a specific dataset. This exercise will be a bit more complicated than the previous exercises because you must implement a client-side event handler to handle the report "Save As" event in which you will redirect the browser to a new action method named **Reports** passing the new report ID in a query string parameter.

1. Add a new method to the **PbiEmbeddingManger** class named **GetNewReportEmbeddingData**.
   a) Open **PbiEmbeddingManager.cs** in an editor window if it is not already open.
   b) Navigate to the bottom of the class definition just beneath the **GetQnaEmbeddingData** method.
   c) Add a new method named **GetNewReportEmbeddingData** by copying and pasting the following code.

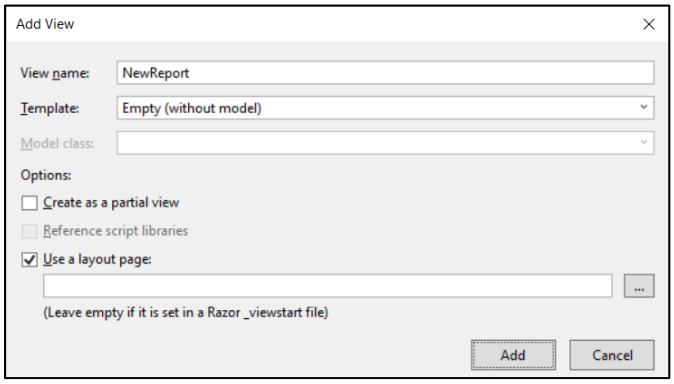
```
public static async Task<NewReportEmbeddingData> GetNewReportEmbeddingData() {

  string embedUrl = "https://app.powerbi.com/reportEmbed?groupId=" + workspaceId;

  PowerBIClient pbiClient = GetPowerBiClient();

  GenerateTokenRequest generateTokenRequestParameters =
                       new GenerateTokenRequest(accessLevel: "create", datasetId: datasetId);
  string embedToken =
    (await pbiClient.Reports.GenerateTokenForCreateInGroupAsync(workspaceId,
                                                                generateTokenRequestParameters)).Token;

  return new NewReportEmbeddingData {
    workspaceId = workspaceId,
    datasetId = datasetId,
    embedUrl = embedUrl,
    accessToken = embedToken
  };

}
```

Notice that you are required to pass a dataset ID when generating an embed token which will be used to embed a new report.

2. Add a new action method to the **HomeController** class named **NewReport**.
   a) Open **HomeController.cs** in an editor window if it's not already open.
   b) Add a new action method named **NewReport** just beneath the **Qna** method using the following code.

```
public async Task<ActionResult> NewReport() {
  NewReportEmbeddingData embeddingData = await PbiEmbeddingManager .GetNewReportEmbeddingData();
  return View(embeddingData);
}
```

3. Create a razor view for the **NewReport** action method.
   a) Right-click on the **NewReport** action method and select the **Add View…** command from the context menu.
   b) In the **Add View** dialog, accept all the default settings and click the **Add** button.



   c) You should see that a new razor view file named **NewReport.cshtml** has been created in the **Views/Home** folder.

d) Delete any existing code inside **NewReport.cshtml** and replace it with the following HTML code.

```
@model AppOwnsDataApp.Models.NewReportEmbeddingData

<div id="embedContainer" />

<script src="~/Scripts/powerbi.js"></script>
<script>

  // Get data required for embedding
  var embedWorkspaceId = "@Model.workspaceId";
  var embedDatasetId = "@Model.datasetId";
  var embedUrl = "@Model.embedUrl";
  var accessToken = "@Model.accessToken";

  // Get models object to access enums for embed configuration
  var models = window['powerbi-client'].models;

  var config = {
    datasetId: embedDatasetId,
    embedUrl: embedUrl,
    accessToken: accessToken,
    tokenType: models.TokenType.Embed,
  };

  // Get a reference to the embedded report HTML element
  var embedContainer = document.getElementById('embedContainer');

  // Embed the report and display it within the div container.
  var report = powerbi.createReport(embedContainer, config);

  // add event handler to load existing report afer saving new report
  report.on("saved", function (event) {
    console.log("saved");
    console.log(event.detail);
    window.location.href = "/Home/Reports/?reportId=" + event.detail.reportObjectId;
  });

</script>
```

e) Save your changes to **NewReport.cshtml**.

You should observe how the code in this script block registers a callback function by calling the **report.on("Saved")** method. You should also observe that this event handle is written to redirect the browser to the **Reports** action of the **Home** controller along with a query string parameter named **reportId** which will be used to pass the identifying GUID of the newly created report. Over the next few steps you will add the **Reports** action method to the **Home** controller class to load an existing report that has just been created.

4. Add a new method to the **PbiEmbeddingManger** class named **GetEmbeddingDataForReport**.

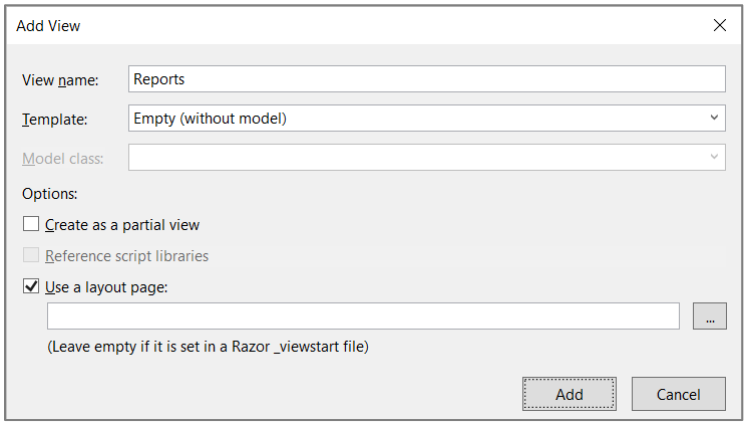
a) In **PbiEmbeddingManager.cs**, add the **GetEmbeddingDataForReport** method by copying and pasting the following code.

```
public static async Task<ReportEmbeddingData> GetEmbeddingDataForReport(string currentReportId) {
  PowerBIClient pbiClient = GetPowerBiClient();
  var report = await pbiClient.Reports.GetReportInGroupAsync(workspaceId, currentReportId);
  var embedUrl = report.EmbedUrl;
  var reportName = report.Name;

  GenerateTokenRequest generateTokenRequestParameters = new GenerateTokenRequest(accessLevel: "edit");
  string embedToken =
        (await pbiClient.Reports.GenerateTokenInGroupAsync(workspaceId,
                                                           currentReportId,
                                                           generateTokenRequestParameters)).Token;

  return new ReportEmbeddingData {
    reportId = currentReportId,
    reportName = reportName,
    embedUrl = embedUrl,
    accessToken = embedToken
  };

}
```

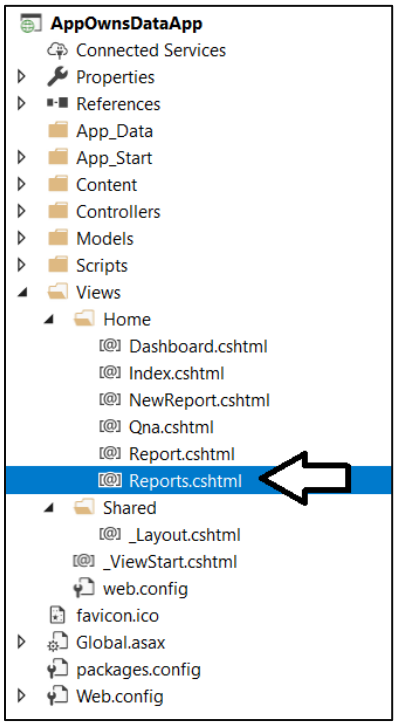
5. Add a new action method to the **HomeController** class named **Reports**.
   a) Open **HomeController.cs** in an editor window if it's not already open.
   b) Add a new action method named **Reports** just beneath the **NewReports** method using the following code.

```
public async Task<ActionResult> Reports(string reportId) {

  ReportEmbeddingData embeddingData =
      await PbiEmbeddingManager.GetEmbeddingDataForReport(reportId);

  return View(embeddingData);
}
```

6. Create a razor view for the **Reports** action method.
   a) Right-click on the R**eports** action method and select the **Add View…** command from the context menu.
   b) In the **Add View** dialog, accept all the default settings and click the **Add** button.



   c) You should see that a new razor view file named **Reports.cshtml** has been created in the **Views/Home** folder.



d) Delete any existing code inside **Reports.cshtml** and replace it with the following HTML code.

```
@model AppOwnsDataApp.Models.ReportEmbeddingData

@section toolbar {
  <div id="toolbar" class="btn-toolbar bg-dark" role="toolbar">
    <button type="button" id="toggleEdit" class="btn btn-sm">Toggle Edit Mode</button>
    <button type="button" id="fullScreen" class="btn btn-sm">Full Screen</button>
    <button type="button" id="print" class="btn btn-sm">Print</button>
  </div>
}

<div id="embedContainer" />

<script src="~/Scripts/powerbi.js"></script>

<script>

  // Data required for embedding Power BI report
  var embedReportId = "@Model.reportId";
  var embedUrl = "@Model.embedUrl";
  var accessToken = "@Model.accessToken";

  // Get models object to access enums for embed configuration
  var models = window['powerbi-client'].models;

  var config = {
    type: 'report',
    id: embedReportId,
    embedUrl: embedUrl,
    accessToken: accessToken,
    tokenType: models.TokenType.Embed,
    permissions: models.Permissions.All,
    viewMode: models.ViewMode.Edit,
    settings: {
      filterPaneEnabled: false,
      navContentPaneEnabled: true,
    }
  };

  // Get a reference to HTML element that will be embed container
  var reportContainer = document.getElementById('embedContainer');

  // Embed the report and display it within the div container.
  var report = powerbi.embed(reportContainer, config);

  var viewMode = "edit";

  $("#toggleEdit").click(function () {
    // toggle between view and edit mode
    viewMode = (viewMode == "view") ? "edit" : "view";
    report.switchMode(viewMode);
    // show filter pane when entering edit mode
    var showFilterPane = (viewMode == "edit");
    report.updateSettings({
      "filterPaneEnabled": showFilterPane
    });
  });

  $("#fullScreen").click(function () {
    report.fullscreen();
  });

  $("#print").click(function () {
    report.print();
  });

</script>
```

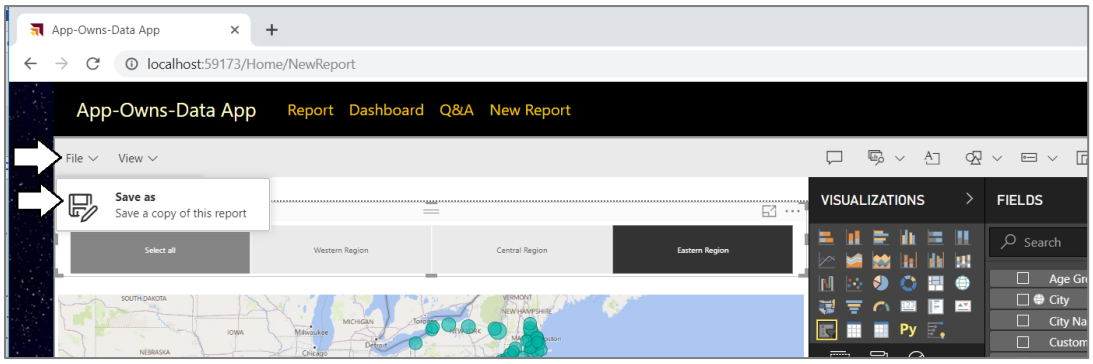
7. Test out the application by running it in the Visual Studio debugger.

a) Press the **{F5}** key in Visual Studio to begin a new debugging session.

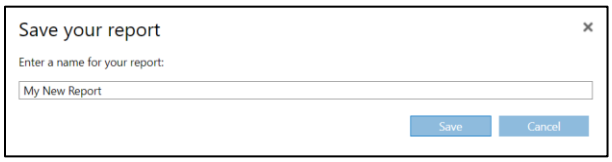
b) Click the **New Report** link in the top navigation menu and you should see an new empty in design mode.

c) Add one or two visuals to the new report.

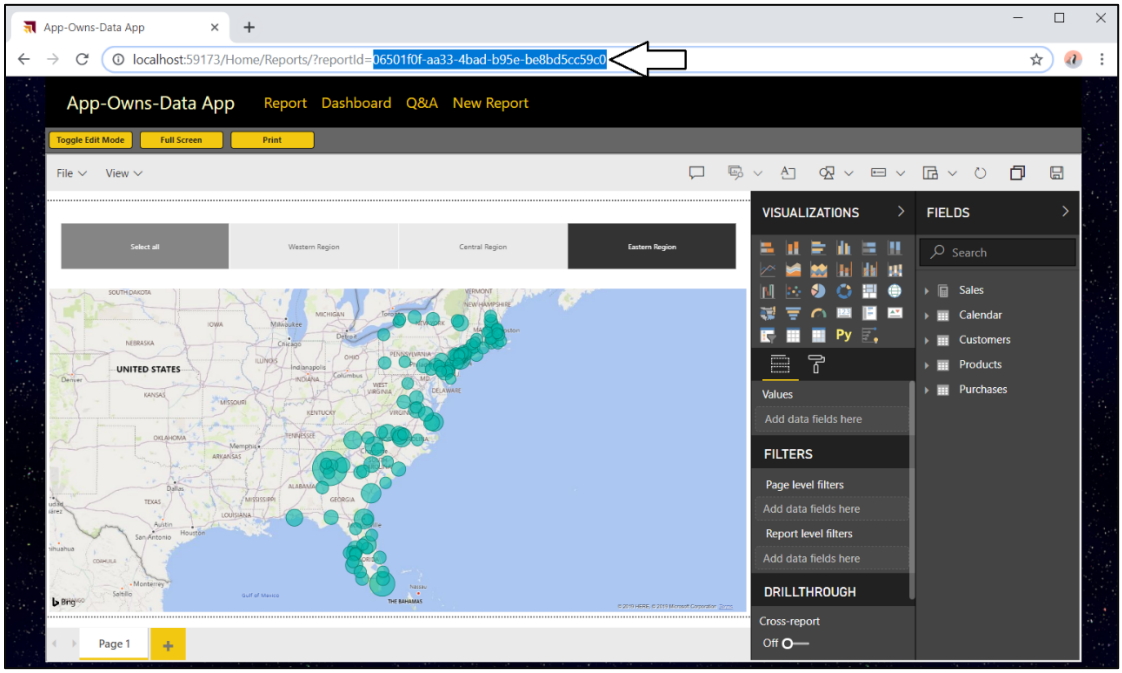
d) Save the new report by dropping down the **File** menu and selecting the **Save as** command.



e) In the **Save your report** dialog, give the new report a name such as **My New Report** and click the **Save** button.



f) After the report has been saved, the browser should redirect to the **Home/Reports** action method and your application should be able to load in the newly created report using the GUID for its report ID.



g) When you are done with your testing, close the browser, return to Visual Studio and stop the current debugging session.

Congratulations. You have made it to the end of this lab.