

# Embedding Power BI Reports and Dashboards

**Setup Time:** 60 minutes

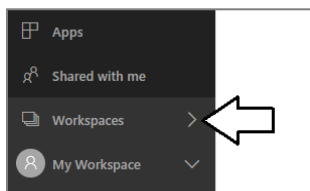
**Lab Folder:** C:\Student\Modules\08\_PBIEmbedding\Lab

**Overview:** In this lab you will work through the steps to develop a custom web application that embeds Power BI reports and dashboards. You will begin by creating an app workspace and populating it with embeddable Power BI resources including a dataset, a report and a dashboard. Next, you will register a new application with Azure AD in the Azure portal and configure this Azure AD application with the proper permissions to program using the Power BI Service API. After that, you will create a new web application using Visual Studio 2017 and ASP.NET MVC. 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.

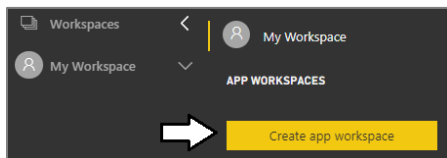
## Exercise 1: Get Started using the Power BI Onboarding Experience

In this exercise, you will create a new app workspace and then you will work to populate this workspace with embeddable content which will include a dataset, a report and a dashboard.

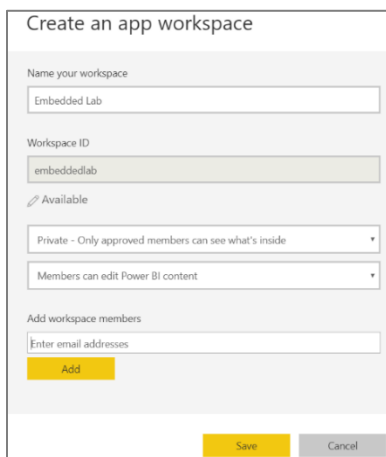
1. Make sure you are logged into the Power BI service with your primary Office 365 user account.
2. Create a new app workspace named **Embedded Lab**.
  - a) Click the **Workspace** flyout menu in the left navigation.



- b) Click the **Create app workspace** button to display the **Create an app workspace** dialog.

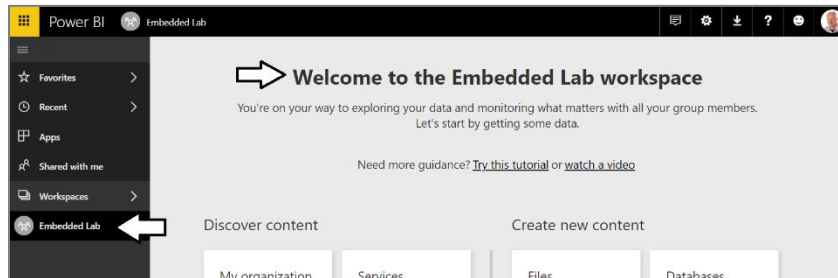


- c) In the **Create an app workspace** pane, enter a name of **Embedded Lab**.
- d) Accept all the other default settings and click **Save**.

A screenshot of the 'Create an app workspace' dialog box. The 'Name your workspace' field contains 'Embedded Lab'. The 'Workspace ID' field contains 'embeddedlab'. The 'Available' section has 'Private - Only approved members can see what's inside' selected. The 'Members can edit Power BI content' dropdown is set to 'Members can edit Power BI content'. The 'Add workspace members' section has an 'Add' button. At the bottom, there are 'Save' and 'Cancel' buttons.

When you create a new app workspace, the account you are logged in as is automatically added as a workspace admin.

- e) When you click **Save**, the Power BI service should create the new app workspace and then switch your current Power BI session to be running within the context of this new workspace named **Embedded Lab**.

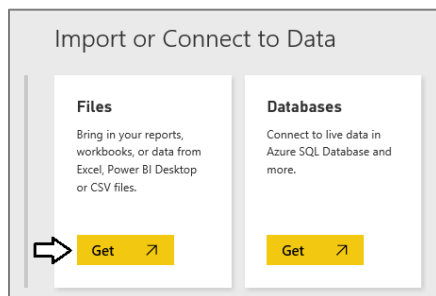


By creating a new workspace, you have effectively created new container for a new set of datasets, reports and dashboards.

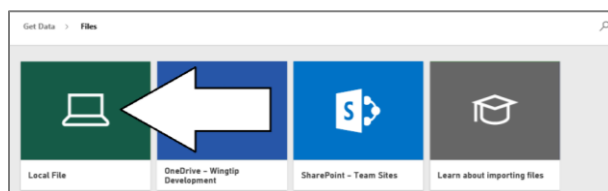
3. Import the **Wingtip Sales Analysis.pbix** project into the **Embedded** app workspace.
- a) Using Windows Explorer, verify there is a PBIX file named **Wingtip Sales Analysis.pbix** at the following path.

**C:\Student\Projects\Wingtip Sales Analysis.pbix**

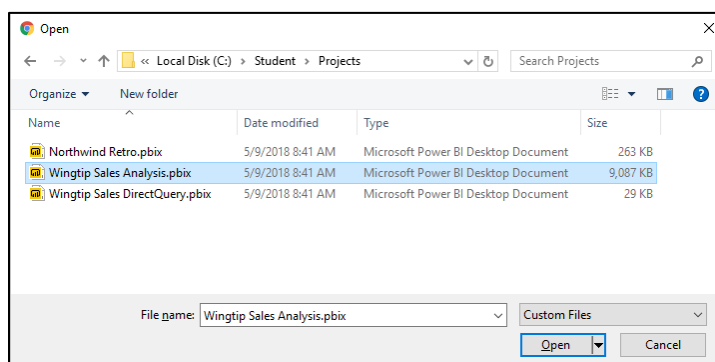
- b) If you do not have a local copy of the **Student** folder, you can download this PBIX file from [here](#).
- c) In the Power BI Service, click the **Get** button in the **Files** section of the Welcome page.



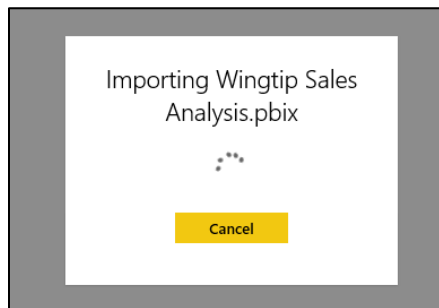
- d) On the **Get Data > Files** page, click the **Local File** button to display the Windows **Open** file dialog.



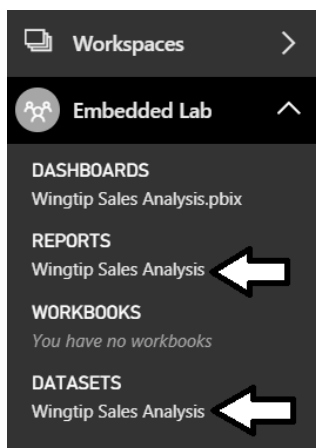
- e) In the Windows **Open** file dialog, select the project file at **c:\Student\PBIX\Wingtip Sales Analysis.pbix** and click **Open**.



- f) Wait while the Power BI service uploads the PBIX files and imports its assets into the **Embedded Lab** app workspace

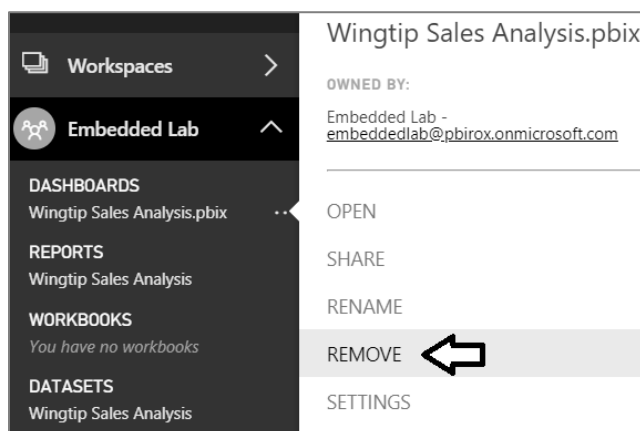


- g) Once the import process completes, you'll see a new dataset, a new report and a new dashboard in the left navigation menu.



4. Remove the dashboard that was created during the import process.

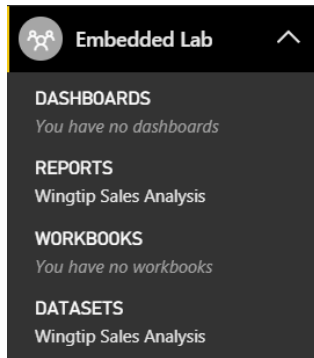
- a) Dropdown the flyout menu for the **Wingtip Sales Analysis.pbix** dashboard and click the **REMOVE** menu command.



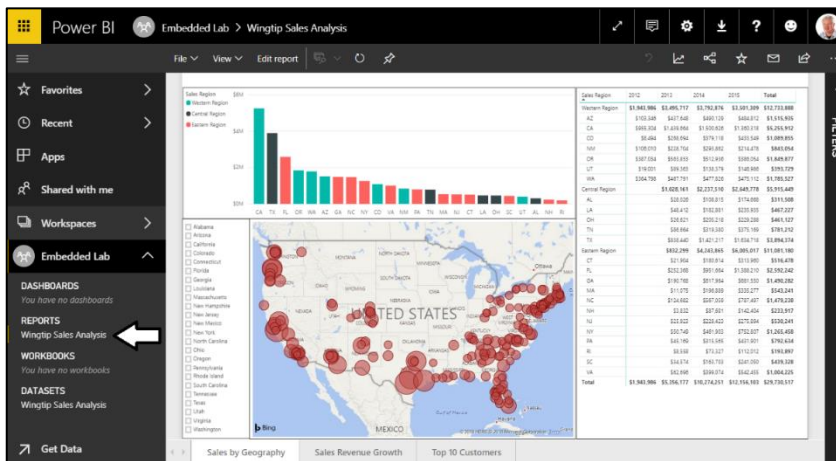
- b) Confirm that you want to delete the dashboard by clicking the **Delete** button the **Delete dashboard** dialog.



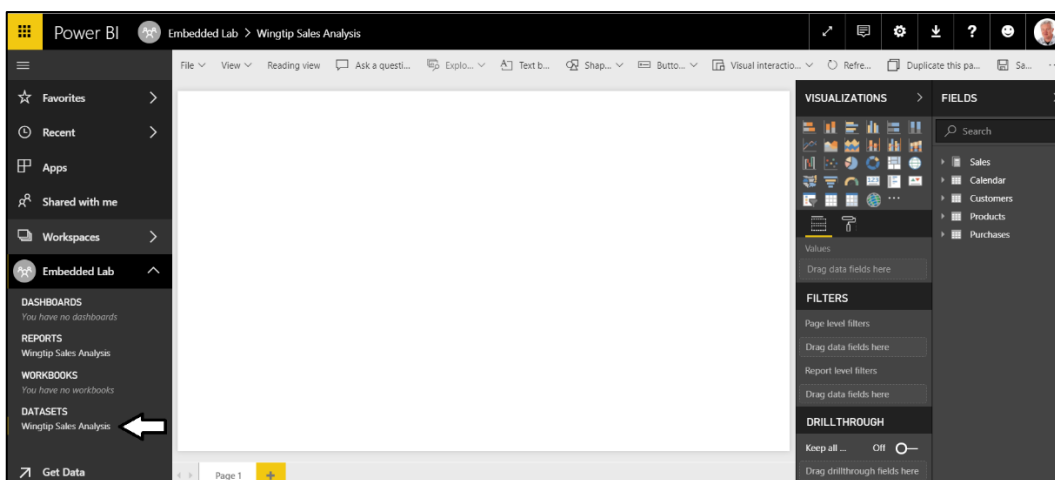
- c) You should be able to confirm that the dashboard has been removed.



5. Click on the report named **Wingtip Sales Analysis** in the **Reports** section. Examine the pages in the report.



6. Click on the dataset named **Wingtip Sales Analysis** in the **Datasets** section. The Power BI service responds by displaying a new report that allows you to begin adding visuals.



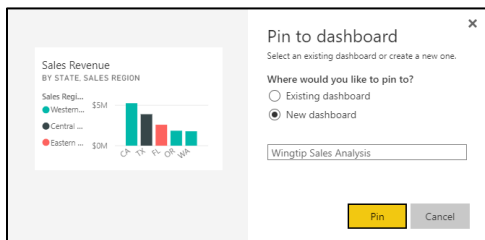
When you navigate to a dataset in the Power BI service, it provides a different experience compared to when in Power BI Desktop. That's because Power BI Desktop allows you to customize and extend a dataset while the browser-based experience of the Power BI Service only allows you to consume datasets but not to modify them. Given the fact that a dataset is a read-only object, the Power BI Service responds to user's request to navigate to a dataset by opening a new report and showing the **Fields** list for that dataset.

## 7. Create a new Dashboard named **Wingtip Sales Analysis**.

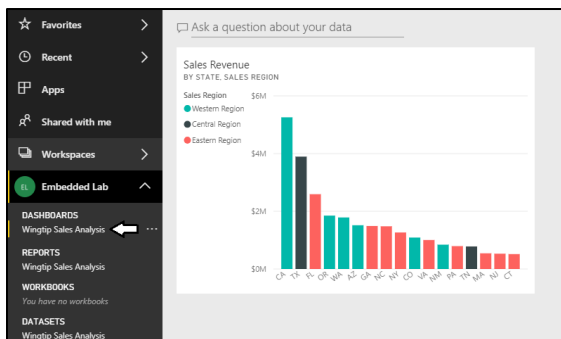
- Navigate to the **Sales by Geography** page of the **Wingtip Sales Analysis** report.
- Inspect the Stacked column chart which displays a sales revenue breakdown across sales regions and product categories.
- Locate and click the button with the thumbtack icon which is used to pin a report visualization to a dashboard.



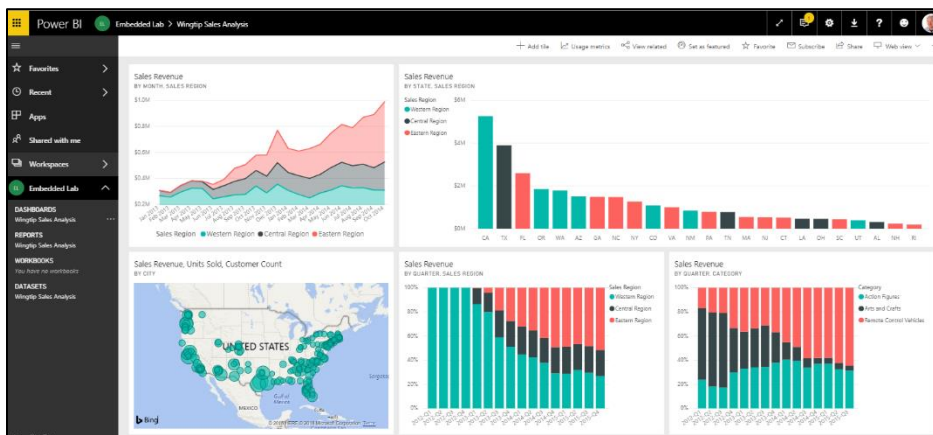
- When you click the thumbtack button, you will be prompted with the dialog which asks you where to pin the visualization.
- Select **New Dashboard**, give it a name of **Wingtip Sales Analysis** and click the **Pin** button.



- At this point, the **Wingtip Sales Analysis** dashboard should be created and a link to it should appear in the left navigation.



- Repeat the process several times of pinning a visual from the report to create new dashboard tiles. Choose whatever visuals you'd like from the report but make sure your dashboard contains at least 4 to 5 tiles as shown in the following screenshot.



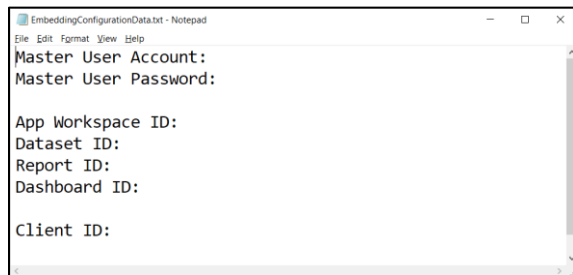
Now that you have finished preparing the app workspace with the content you will use for Power BI embedding, you must record a few key pieces of configuration data you will need later in this lab. First, you will record which Active Directory user account you will use as the master user account. Next, you will record the identifying GUID for the **Embedded Lab** app workspace and the identifying GUIDs for the dataset, report and dashboard you created inside this app workspace.

8. Record configuration data that you will need later in this lab.

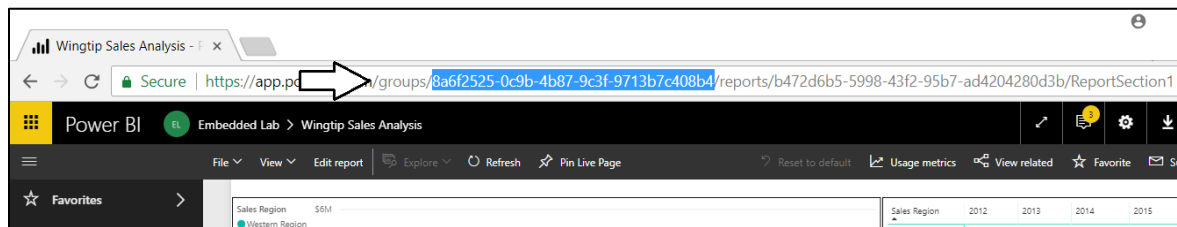
- a) Locate the configuration file named **EmbeddingConfigurationData.txt** which is located at the following path.

**C:\Student\Modules\04\_PowerBIEmbedding\Lab\EmbeddingConfigurationData.txt**

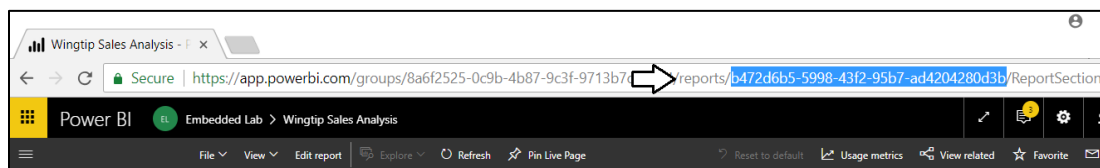
- b) Open **EmbeddingConfigurationData.txt** using Windows Notepad and inspect its contents. It contains the names of 7 essential pieces of configuration data you will need later in this lab.



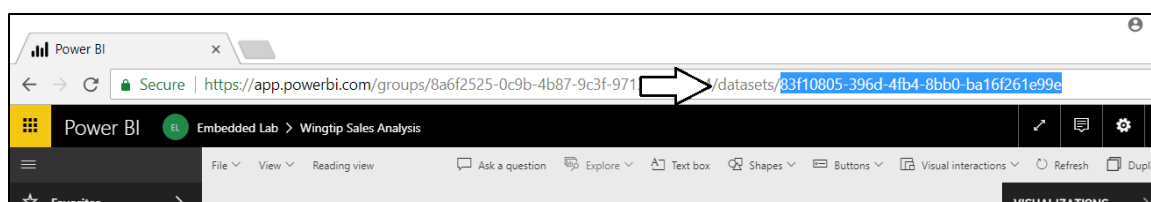
- c) Complete the top two lines by adding the name of your primary Office 365 account and the password for your account.  
d) Navigate to the **Wingtip Sales Analysis** report inside the **Embedded Lab** app workspace.  
e) Locate and copy the app workspace ID from the URL by copying the GUID that comes after **/groups/**.



- f) Copy the app workspace ID into **EmbeddingConfigurationData.txt**.  
g) Navigate back to the **Wingtip Sales Analysis** report inside the **Embedded Lab** app workspace.  
h) Locate and copy the report ID from the URL by copying the GUID that comes after **/reports/**.



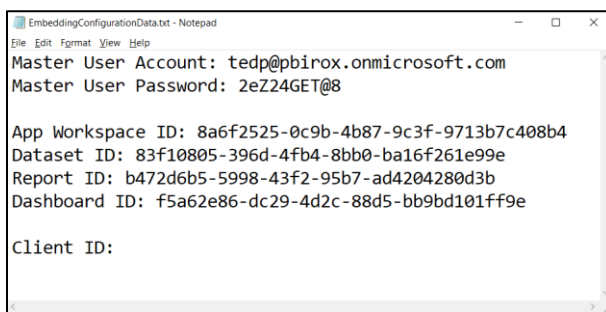
- i) Copy the report ID into **EmbeddingConfigurationData.txt**.  
j) Navigate to the **Wingtip Sales Analysis** dataset inside the **Embedded Lab** app workspace to create a new report.  
k) Locate and copy the dataset ID from the URL by copying the GUID that comes after **/datasets/**.



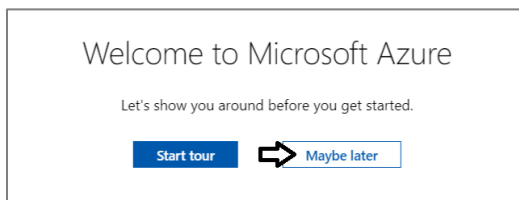
- l) Copy the dataset ID into **EmbeddingConfigurationData.txt**.
- m) Navigate to the **Wingtip Sales Analysis** dashboard inside the **Embedded Lab** app workspace.
- n) Locate and copy the dashboard ID from the URL by copy the GUID that comes after **/dashboards/**.



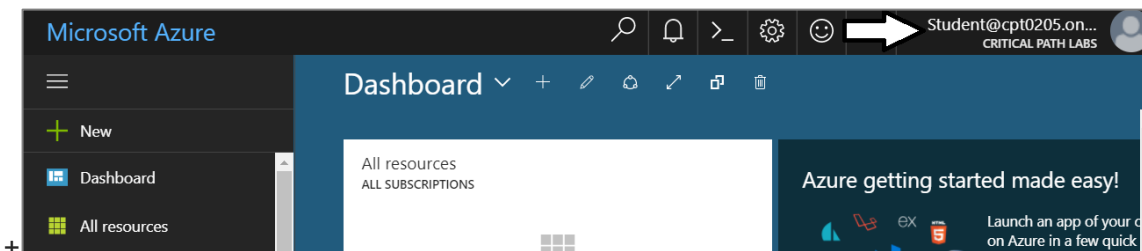
- o) Copy the dashboard ID into **EmbeddingConfigurationData.txt**.
- p) You should have now updated **EmbeddingConfigurationData.txt** with all the configuration data you need with the exception of the client ID that you will create in the next exercise.



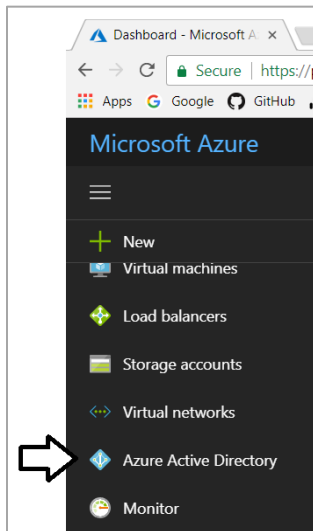
- q) Save your changes to **EmbeddingConfigurationData.txt**.
9. Log into the Azure Portal
- a) In the browser, navigate to the Azure portal at <https://portal.azure.com>.
  - b) When you are prompted to log in, provide the credentials to log in with your Office 365 user account name.
  - c) If you are prompted to start a tour of Microsoft Azure, click **Maybe later**.



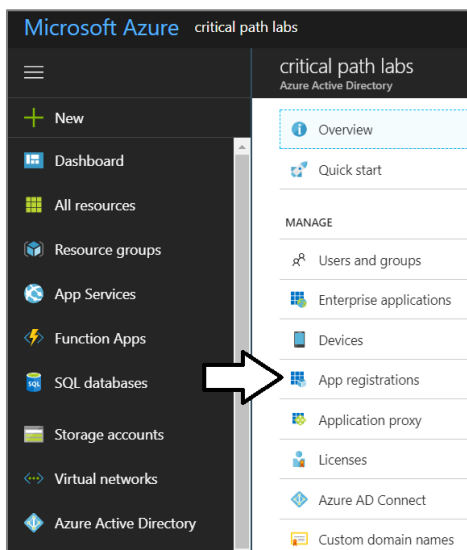
- d) Once you are log into the Azure portal, check the email address in the login menu in the upper right to make sure you are logged in the Azure portal with the Office 365 user account you have been using in this lab.



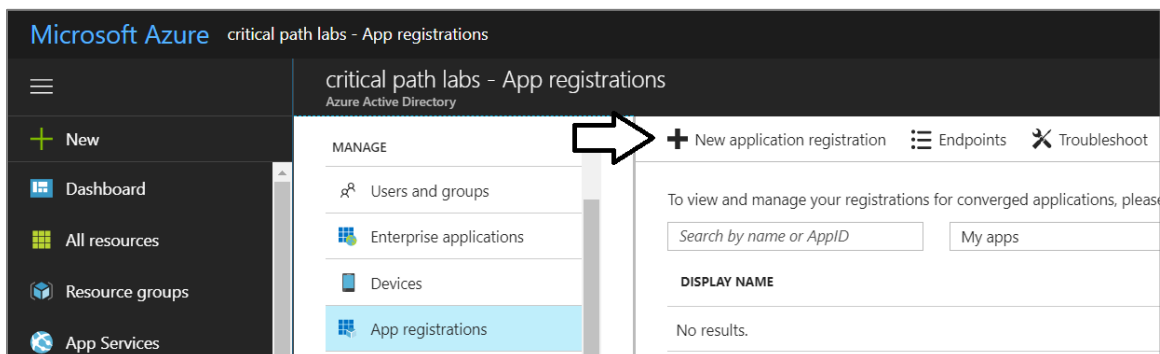
10. Register a new Azure application.
- a) In the left navigation, scroll down and click on the link for **Azure Active Directory**.



b) Click the link for **App registration**.



c) Click **New application registration**.

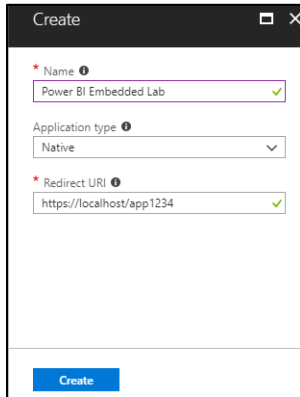


d) In the Create dialog...

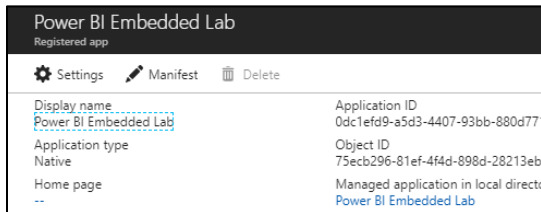
i) Add a **Name of Power BI Embedded Lab**.



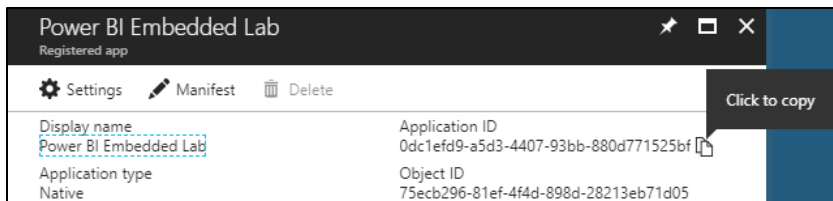
- ii) Set the **Application type** to **Native**.
- iii) Set the **Redirect URI** to <https://localhost/app1234>.
- iv) Click the **Create** button to create the new application.



- e) You should now see the overview page for the new Azure AD application.




- f) Copy the GUID for the Application ID.



Note that the application ID and the client ID are the same thing. This is a constant source of confusion to developers new to Azure AD authentication. The GUID that identifies an Azure AD application can be referred to as either the application ID or the client ID.

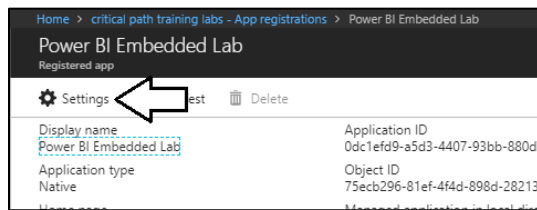
- g) Copy the Application ID as the client ID value in **EmbeddingConfigurationData.txt**.



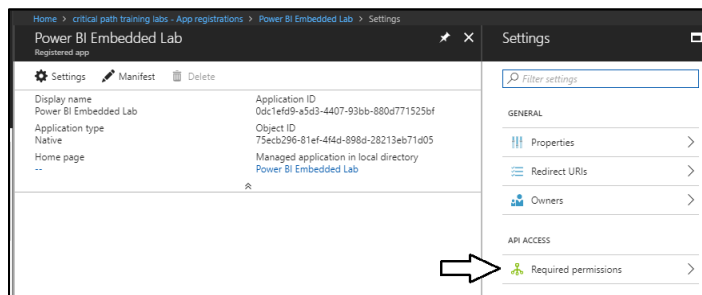
- h) Save changes to **EmbeddingConfigurationData.txt**.

## 11. Configure the new Azure application with permissions to call the Power BI Service API.

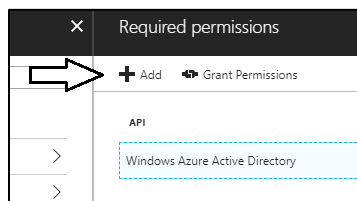
- a) Click on the **Settings** link to configure application settings,



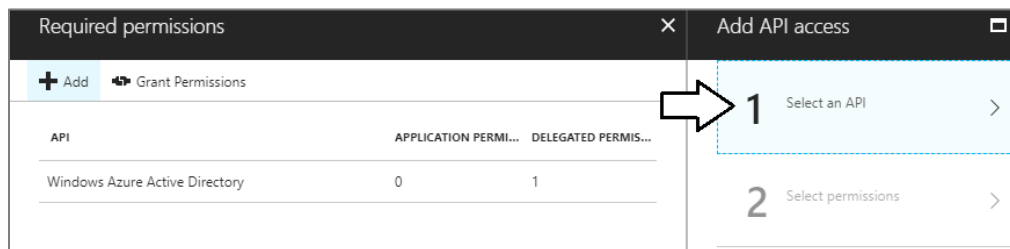
b) Click **Required permissions**.



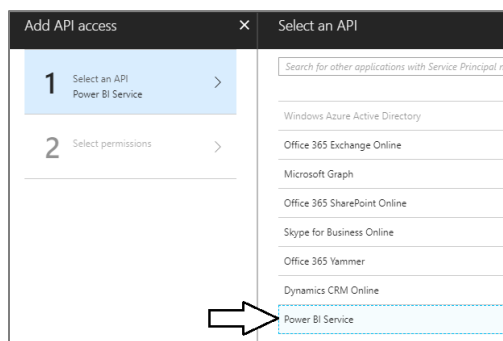
c) Click the **Add** button on the **Required permissions** blade.



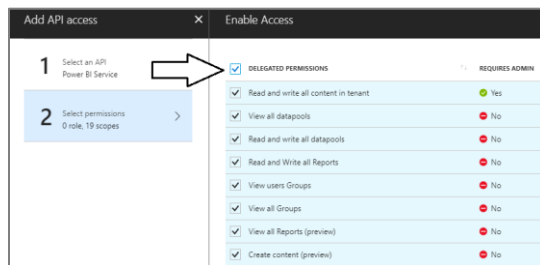
d) Click the **Select an API** option in the **Add API access** blade.



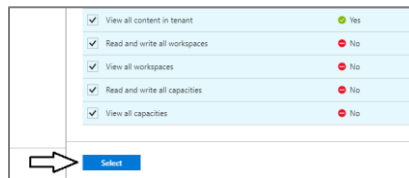
e) In the **Select an API** blade, click **Power BI Service**.



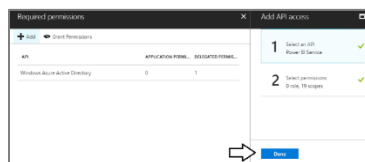
f) In the **Enable Access** blade, click the top checkbox for **DELEGATED PERMISSIONS** to select all the permissions.



- g) Once you have selected all the permissions, click the **Select** button at the bottom of the blade.



- h) Click the **Done** button at the bottom of the **Add API Access** blade.

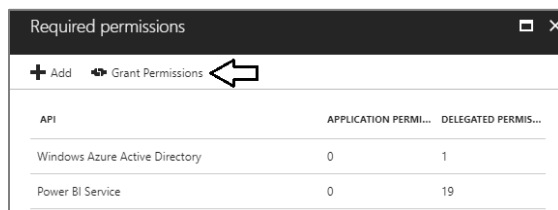


- i) At this point, you should be able to verify that the Power BI Service has been added to the **Required permissions** list.

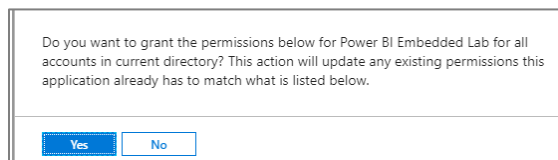
API	APPLICATION PERMI...	DELEGATED PERMIS...
Windows Azure Active Directory	0	1
Power BI Service	0	19

## 12. Grant permissions through the Azure portal to remove the need for an interactive login.

- a) In the **Required permissions** section, click the **Grant Permissions** button.



- b) When prompted by the dialog that asks you to grant permissions, click **Yes**.

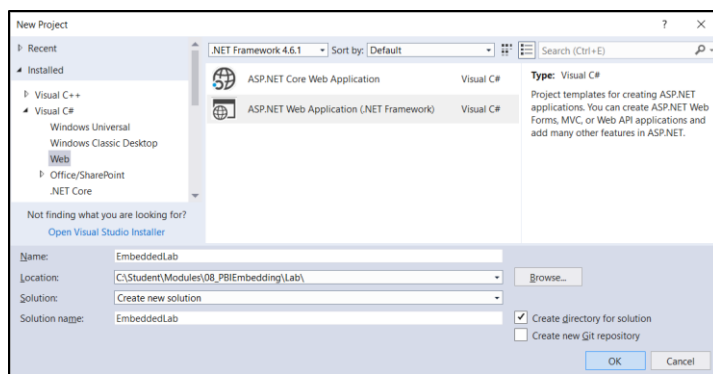


You have now completed the process of creating and configuring your Azure AD application for this lab.

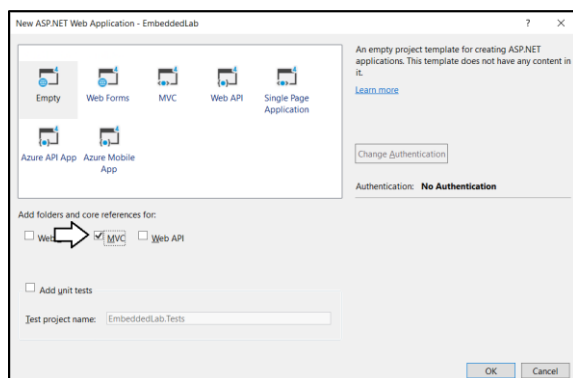
## Exercise 2: Create a new MVC Application using Visual Studio 2017

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

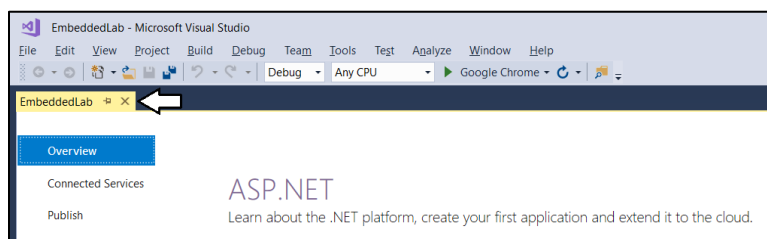
1. Launch **Visual Studio 2017**.
2. Create a new ASP.NET MVC project in Visual Studio 2017.
  - a) In Visual Studio select **File > New > Project**.
  - b) In the **New Project** dialog:
    - i) Select **Installed > Templates > Visual C# > Web**.
    - ii) Select the **ASP.NET Web Application** project template.
    - iii) Name the new project **EmbeddedLab**.
    - iv) Add the new project into the folder at **C:\Student\Modules\08\_PBIEmbedding\Lab**.
    - v) Click **OK** to display the **New ASP.Net Web Application** wizard.



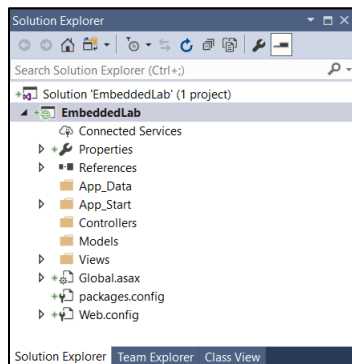
- c) In the **New ASP.Net Web Application** dialog, select the **Empty** template.
- d) In the section with the caption **Add folders and core references**, make sure the **MVC** checkbox is checked.
- e) Click the **OK** button to create the new project.



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

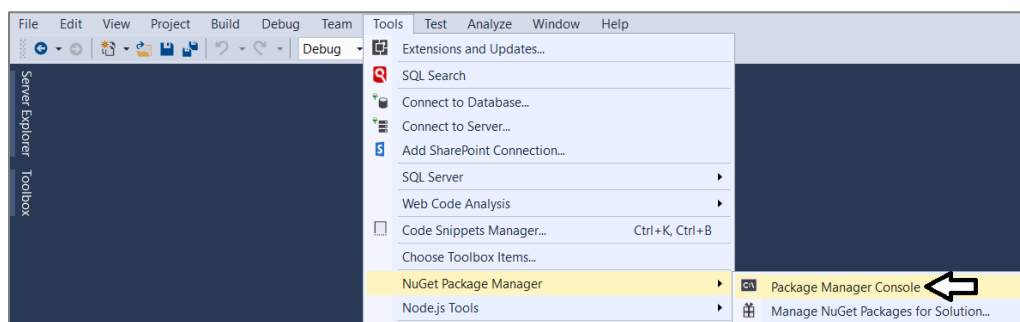


- g) Take a minute to familiarize yourself with the structure of the project in the **Solution Explorer**.

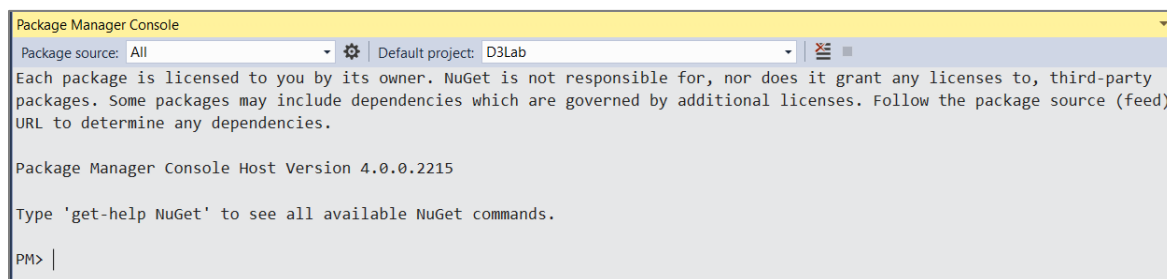


At this point, you have created a new ASP.NET MVC project based on the **Empty** project template. You will need to add an MVC controller and several MVC views before your application provides any type of user interface experience. Before adding a controller or writing any code, you will first update the project's NuGet packages that were automatically 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 Service API and the Power BI JavaScript API.

3. Configure the **Embedded Lab** 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 Manager 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 **bootstrap**.

```
Install-Package bootstrap
```

- d) Type in and execute the following command to install the NuGet package for **Azure Active Directory Authentication library**.

```
Install-Package Microsoft.IdentityModel.Clients.ActiveDirectory
```

- e) Type in and execute the following command to install the NuGet package for the **Power BI Service API**.

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

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

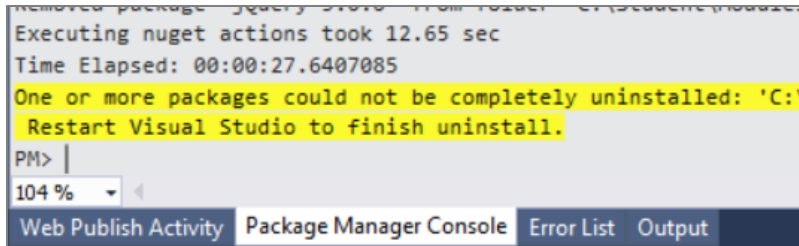
**Install-Package Microsoft.PowerBI.JavaScript**

Now that you have installed the required NuGet packages for Power BI embedding, you will now run the **Update-Package** cmdlet to make sure all the packages in your project are updated to the latest versions available in the NuGet repository.

- g) Type in and execute the following command to update all NuGet packages in the project to their most current version.

**Update-Package**

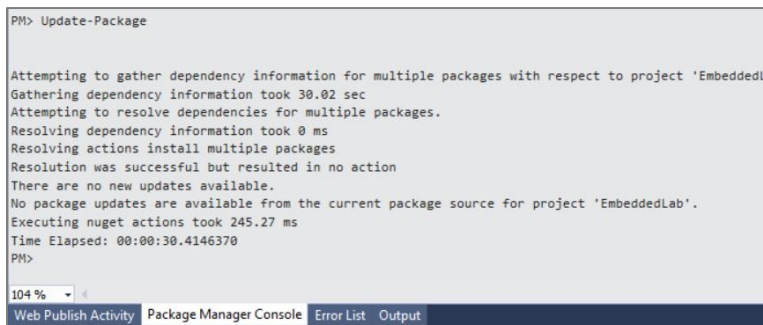
- h) The first time you run the **Update-Package** cmdlet, you will be prompted to restart Visual Studio to complete the update.



- i) Restart Visual Studio and open the **EmbeddedLab** project.  
j) Open the Package Manager Console window if it is not already open.  
k) Execute the **Update-Package** cmdlet one more time to ensure all NuGet packages are updated to their most current version.

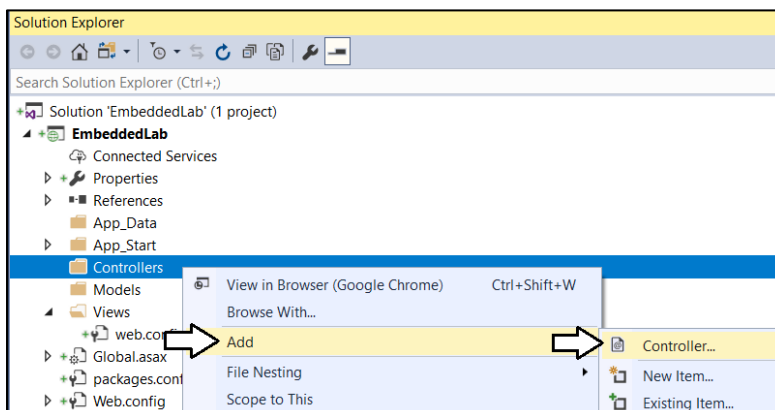
**Update-Package**

- l) You should now see an output message in the Package Manager Console indicating “There are no new updates available”.

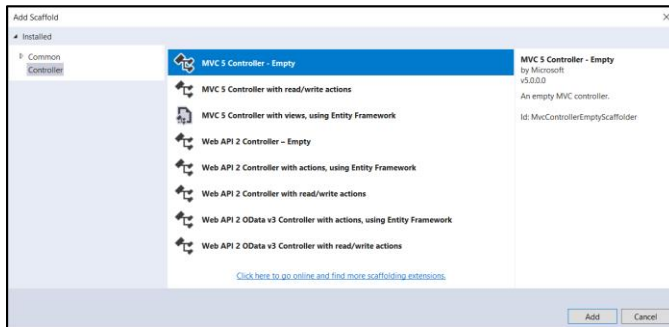


#### 4. Add the **HomeController** class.

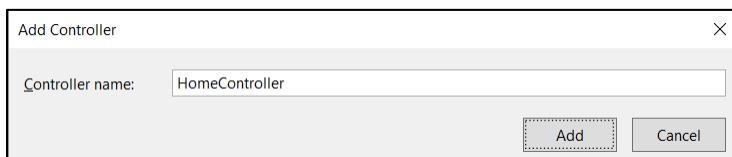
- a) In Solution Explorer, right-click on the **Controllers** folder.



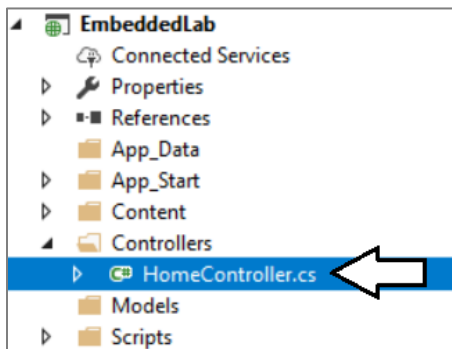
- b) In the **Add Scaffold** dialog, select the first option **MVC 5 Controller – Empty** and then click **Add**.



- 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**.

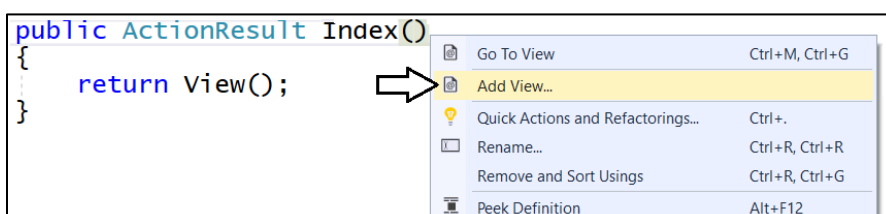


- e) Inside **HomeController.cs**, you will find the starting point for the **HomeController** class with a single method named **Index**.

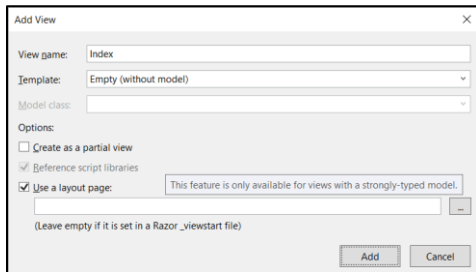
```
namespace EmbeddedLab.Controllers
{
    public class HomeController : Controller
    {
        // GET: Home
        public ActionResult Index()
        {
            return View();
        }
    }
}
```

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.

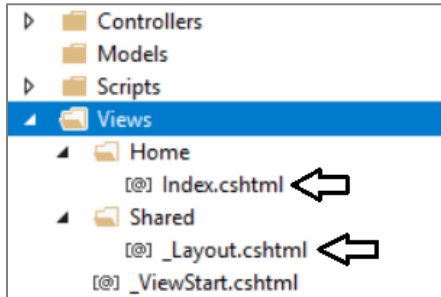


- b) In the **Add View** dialog, accept all the default setting as shown in the following screenshot and click **Add**.

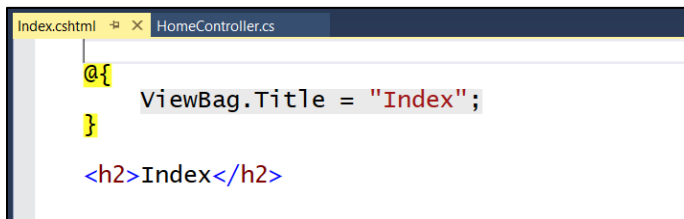


When you create a new view and leave the **Use a layout page** option selected, a new shared layout page named **\_Layouts.cshtml** is automatically added to the project in the **Views/Shared** folder.

- 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>Power BI Embedded Lab</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\08\_PBIEmbedded\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 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>Embedded Lab</title>
  <link href="~/Content/bootstrap.css" rel="stylesheet" />
  <link href="~/Content/Site.css" rel="stylesheet" />
  <script src="~/Scripts/jquery-3.3.1.js"></script>
</head>
```

- e) Make sure your script link to jQuery matches the version number of the jQuery library source file in the **Scripts** folder.  
f) 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.

- g) 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">
        @Html.ActionLink("Embedded Lab", "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>
```

- h) 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>
```

- i) 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>
```

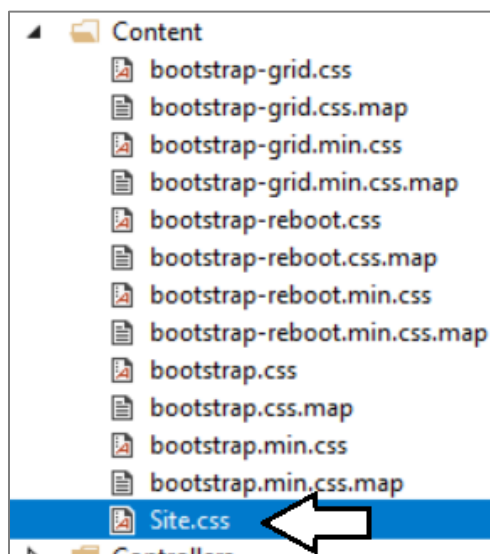
- j) Save your changes and close **\_Layouts.cshtml**.

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\08\_PBIEmbedded\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**.

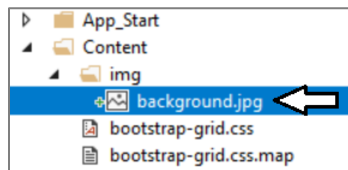
8. Add a new image named **background.jpg** to the project to provide a page background.

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

C:\Student\Modules\08\_PBIEmbedded\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.

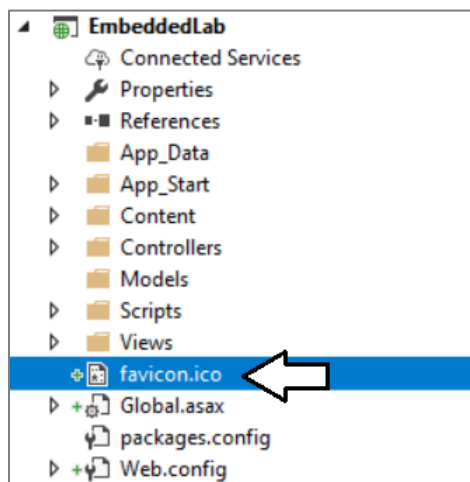


9. Add a **favicon.ico** file to the root folder of the **EmbeddedLab** project.

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

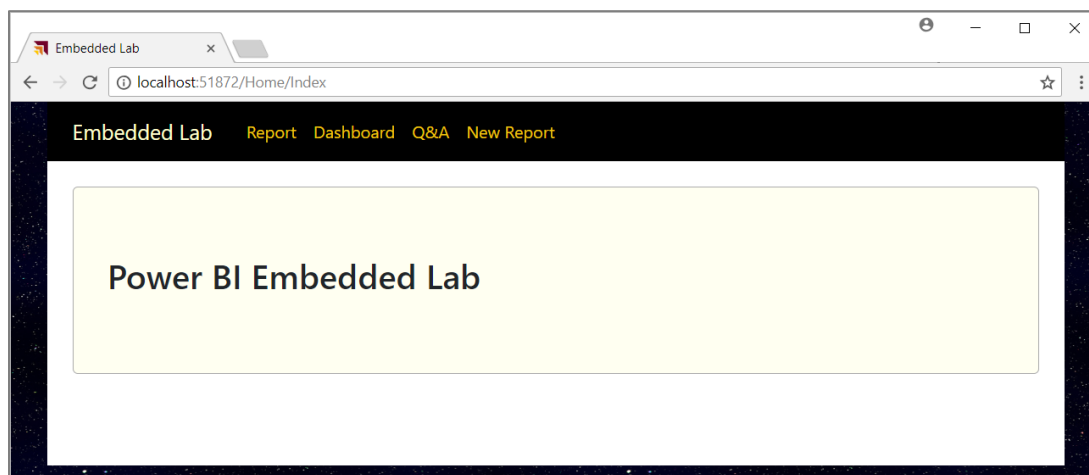
**C:\Student\Modules\08\_PBIEmbedded\Lab\StarterFiles\favicon.ico**

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



10. Test out the **EmbeddedLab** 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.



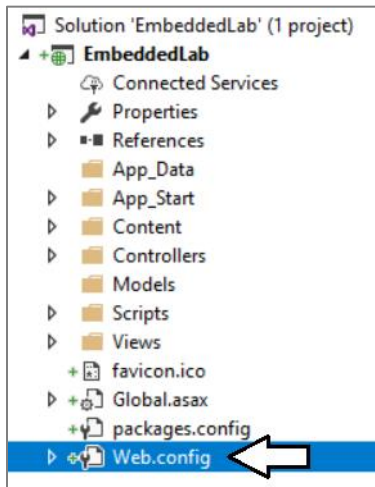
- c) Close the browser, return to Visual Studio and stop the debugger.

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: Embed Your First Power BI Report

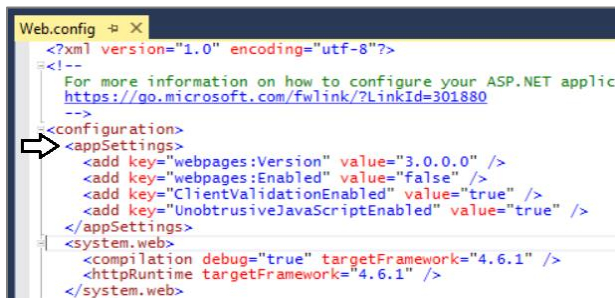
In this exercise, you will create an action method and a razor view named **Report** which will embed a Power BI report.

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 **EmbeddedLab** project.

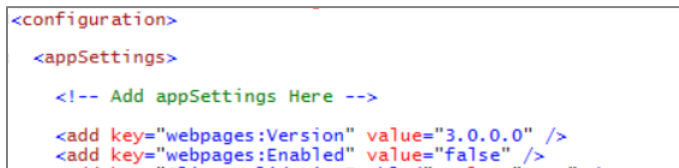


Make sure you open the **web.config** file located at the root of the project and not the **web.config** file inside the **Views** folder.

- b) Locate the **<appSettings>** element at the top of **web.config**.



- c) Add a few blank lines just after the **<appSettings>** element opening tag.



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

```
<add key="aad-account-name" value="" />
<add key="aad-account-password" value=" " />

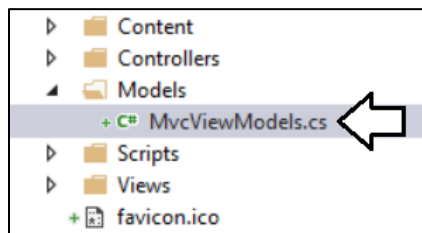
<add key="app-workspace-id" value="" />
<add key="dataset-id" value="" />
<add key="report-id" value="" />
<add key="dashboard-id" value=" " />

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

- e) Copy configuration values from **EmbeddingConfigurationData.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 appSettings Here -->
    <add key="aad-account-name" value="tedp@pbiox.onmicrosoft.com" />
    <add key="aad-account-password" value="2eZ24GET@8" />
    <add key="client-id" value="0dc1efd9-a5d3-4407-93bb-880d771525bf" />
    <add key="app-workspace-id" value="8a6f2525-0c9b-4b87-9c3f-9713b7c408b4" />
    <add key="dataset-id" value="83f10805-396d-4fb4-8bb0-ba16f261e99e" />
    <add key="report-id" value="b472d6b5-5998-43f2-95b7-ad4204280d3b" />
    <add key="dashboard-id" value="f5a62e86-dc29-4d2c-88d5-bb9bd101ff9e" />
    <add key="webpages:Version" value="3.0.0.0" />
  </appSettings>
</configuration>
```

- 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 EmbeddedLab.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;
    }

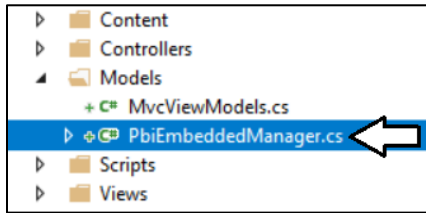
    // data required for embedding a dashboard
    public class DashboardEmbeddingData {
        public string dashboardId;
        public string dashboardName;
        public string embedUrl;
        public string accessToken;
    }

    // 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 **PbiEmbeddedManager** class.

- Add a new class named **PbiEmbeddedManager** inside the **Models** folder.
- The **Models** folder should now contain a C# source file named **PbiEmbeddedManager.cs**.



- Delete any code inside **PbiEmbeddedManager.cs** and replace it with the following starter code.

```
using System;
using System.Configuration;
using System.Threading.Tasks;
using Microsoft.Rest;
using Microsoft.PowerBI.Api.V2;
using Microsoft.PowerBI.Api.V2.Models;
using Microsoft.IdentityModel.Clients.ActiveDirectory;

namespace EmbeddedLab.Models {

    public class PbiEmbeddedManager {

    }

}
```

- Modify the **PbiEmbeddedManager** class by adding the following set of static fields.

```
class PbiEmbeddedManager {

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

    private static string userName = ConfigurationManager.AppSettings["aad-account-name"];
    private static string userPassword = ConfigurationManager.AppSettings["aad-account-password"];

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

    private static string clientId = ConfigurationManager.AppSettings["client-id"];

}
```

In addition to fields for the seven configuration values, there are other fields named **aadAuthorizationEndpoint**, **resourceUriPowerBi** and **urlPowerBiRestApiRoot** which are used when authenticating with Azure AD and calling to the Power BI Service API.

- At the bottom of **PbiEmbeddedManager** class, add a new method named **GetAccessToken** using the following code.

```
private static string GetAccessToken() {

    AuthenticationContext authenticationContext = new AuthenticationContext(aadAuthorizationEndpoint);

    AuthenticationResult userAuthnResult =
        authenticationContext.AcquireTokenAsync(
            resourceUriPowerBi,
            clientId,
            new UserPasswordCredential(userName, userPassword)).Result;

    return userAuthnResult.AccessToken;

}
```

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

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

You have implemented the essential behavior in the **PbiEmbeddedManager** 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. Now you are at a point where you can add methods to the **PbiEmbeddedManager** class which call into the Power BI Service API to retrieve embedding data.

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

- a) At the bottom of the **PbiEmbeddedManager** class, add a method named **GetReportEmbeddingData** with 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.

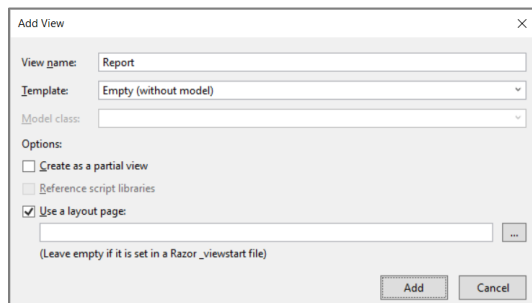
- a) Inside the **Controllers** folder, open the C# source file named **HomeController.cs**.  
b) Update the set of **using** statements at the top of **HomeController.cs** using the following code.

```
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Threading.Tasks;  
using System.Web;  
using System.Web.Mvc;  
using EmbeddedLab.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 PbiEmbeddedManager.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 EmbeddedLab.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;

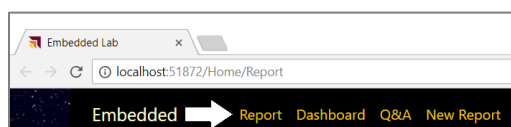
    var config = {
        type: 'report',
        id: embedReportId,
        embedUrl: embedUrl,
        accessToken: accessToken,
        permissions: models.Permissions.All,
        tokenType: models.TokenType.Embed,
        viewMode: models.ViewMode.View,
        settings: {
            filterPaneEnabled: 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>
```

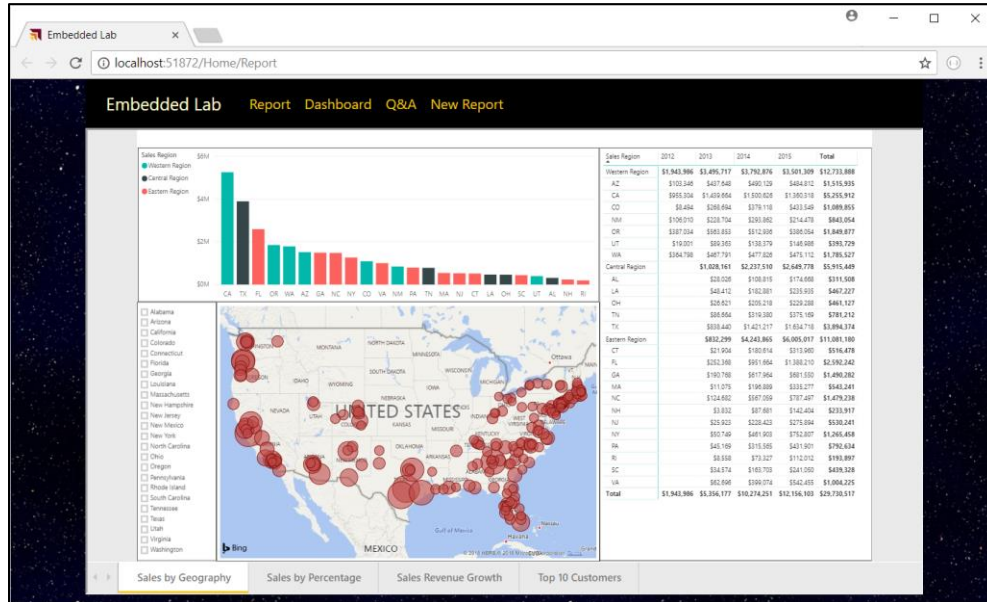
- h) Save your changes and close **Report.cshtml**.
6. Test out the application by running it in the Visual Studio debugger.
- a) Press the **{F5}** key in Visual Studio to begin a new debugging session.
  - b) Click the **Report** link in the top navigation menu.





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 report has been embedded on the web page.



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 and return to Visual Studio and stop the current debugging session.

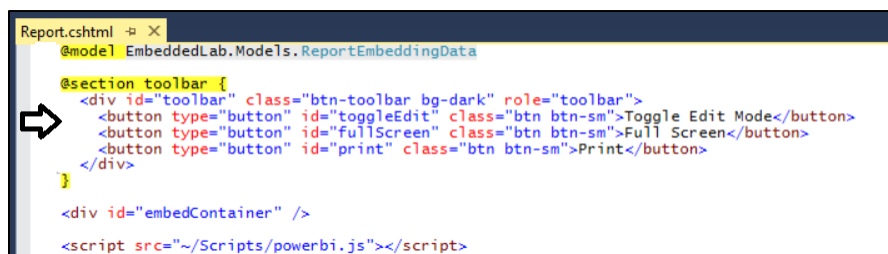
## Exercise 4: Add a Toolbar to Interact with an Embedded Report

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 close **</script>** tag.

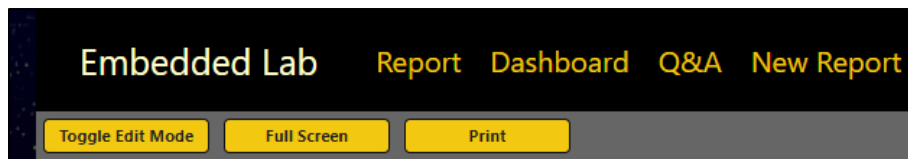
```
var viewMode = "view";

$("#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
    });
});

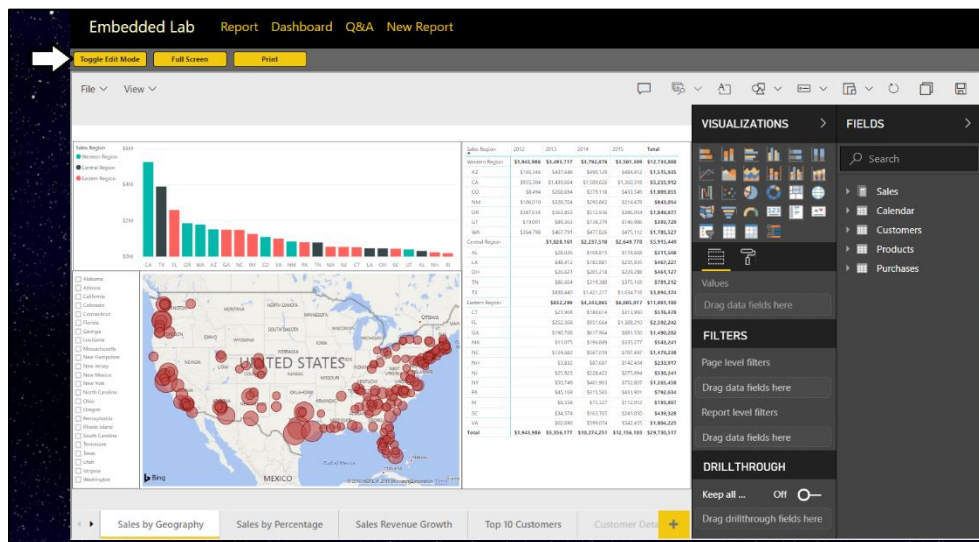
$("#fullScreen").click(function () {
    report.fullscreen();
});

$("#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.

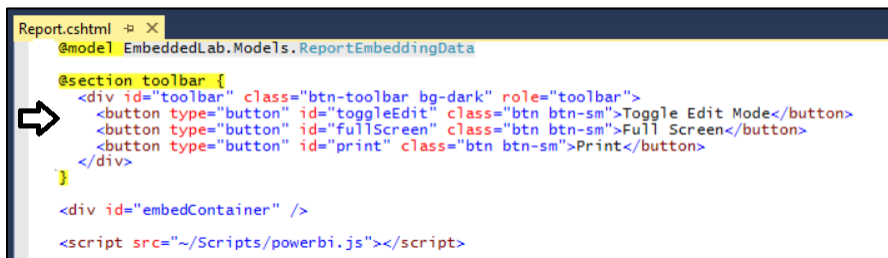
## Exercise 5: Add an Interactive Toolbar for an Embedded Report

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.

3. 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>  
}
```

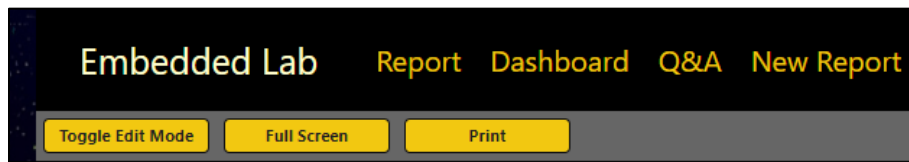
- c) The top of **Report.cshtml** should match the following screenshot.



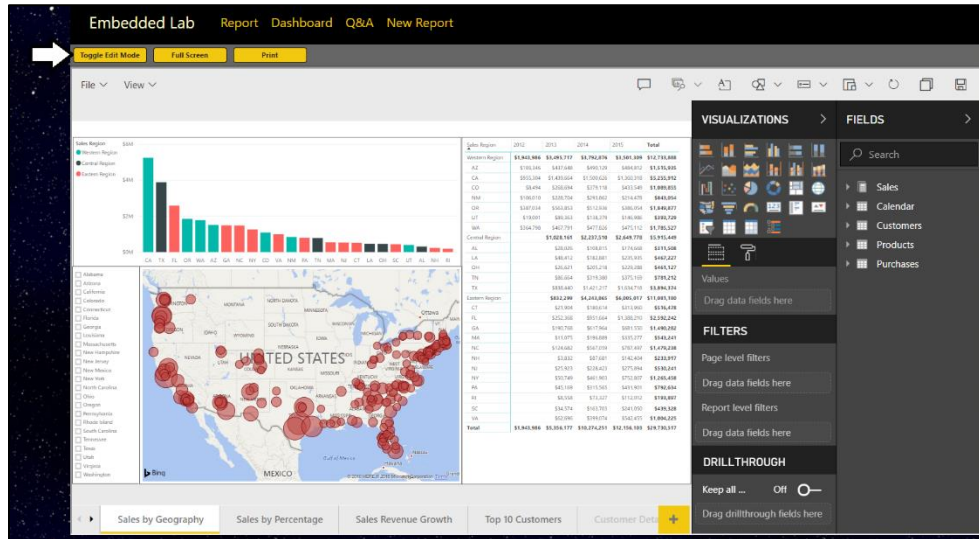
- d) Inside **Report.cshtml**, move down inside **<script>** block and add a few new lines after the line which calls **powerbi.embed**.
  - e) Copy and paste the following JavaScript code at the bottom of the **<script>** block just before the close **</script>** tag.

```
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**.
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 **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 6: Embed 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.

5. Add a new method to the **PbiEmbeddingManger** class named **GetDashboardEmbeddingData**.
  - a) Open **PbiEmbeddedManager.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 **PbiEmbeddedManager.cs**.

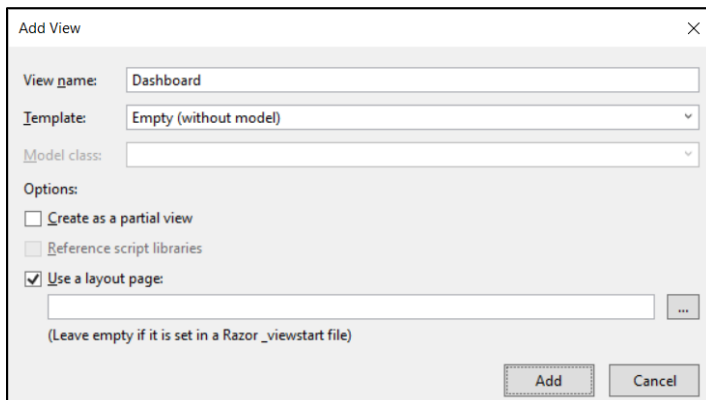
6. Add a new action method to the **HomeController** class named **Dashboard**.

- 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 PbiEmbeddedManager.GetDashboardEmbeddingData();  
    return View(embeddingData);  
}
```

7. 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.

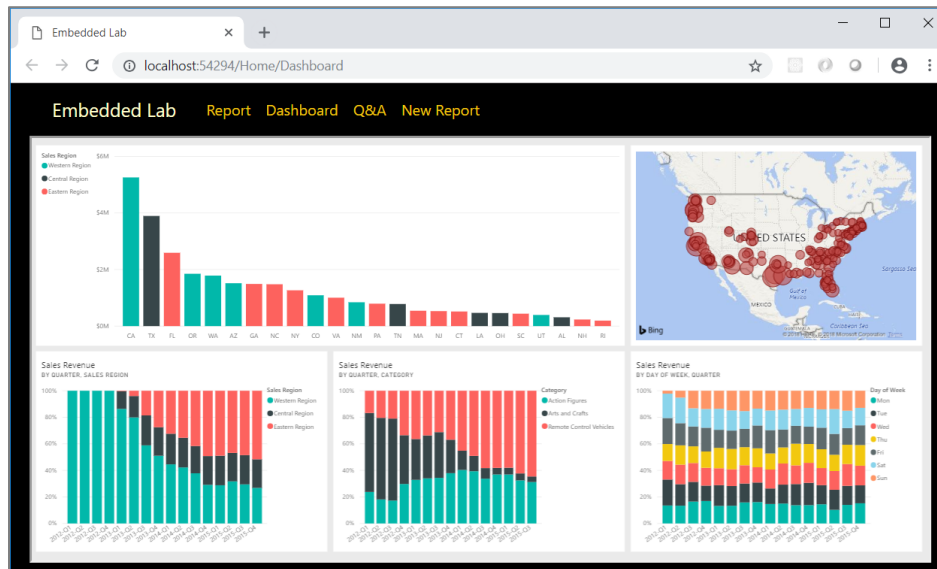


- c) You should see that a new razor view file named **Dashboard.cshtml** has been created in the **Views/Home** folder.
- d) Delete any existing code inside **Dashboard.cshtml** and replace it with the following HTML code.

```
@model EmbeddedLab.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**.
8. 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 7: Embed 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.

9. Add a new method to the **PbiEmbeddedManager** class named **GetQnaEmbeddingData**.
  - a) Open **PbiEmbeddedManager.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;  
    string datasetId = dataset.Id;  
  
    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 **PbiEmbeddedManager.cs**.

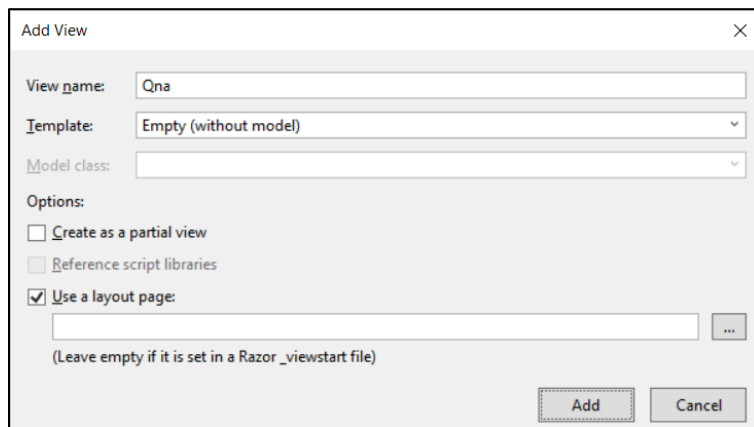
10. Add a new action method to the **HomeController** class named **Qna**.

- 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 PbiEmbeddedManager.GetQnaEmbeddingData();  
    return View(embeddingData);  
}
```

11. 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.



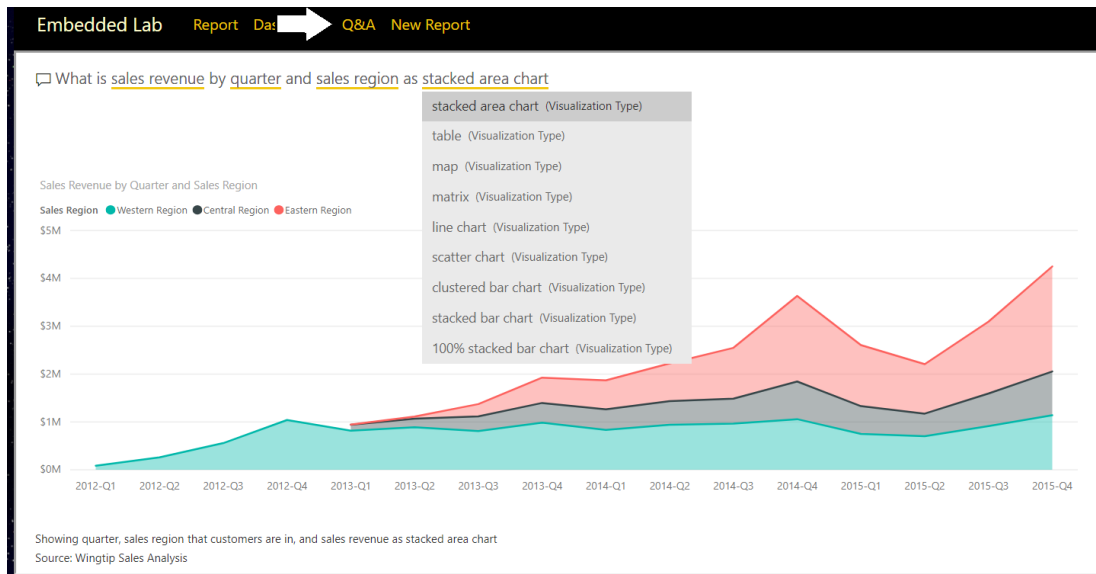
- c) You should see that a new razor view file named **Qna.cshtml** has been created in the **Views/Home** folder.
- d) Delete any existing code inside **Qna.cshtml** and replace it with the following HTML code.

```
@model EmbeddedLab.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**.
12. 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 and return to Visual Studio and stop the current debugging session.

## Exercise 8: Embed 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.

13. Add a new method to the **PbiEmbeddingManager** class named **GetNewReportEmbeddingData**.
  - a) Open **PbiEmbeddedManager.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.

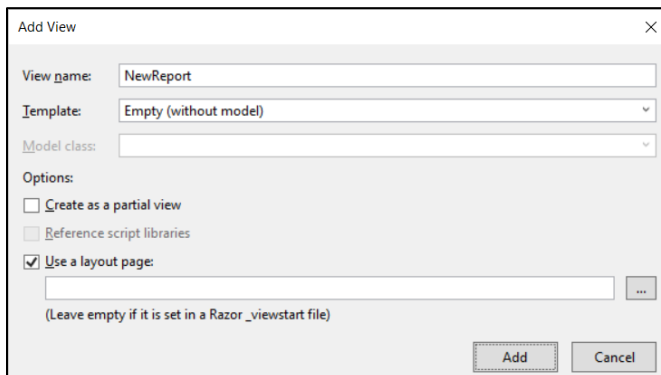
14. Add a new action method to the **HomeController** class named **NewReport**.

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

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

15. Create a razor view for the **NewReport** action method.

- Right-click on the **NewReport** action method and select the **Add View...** command from the context menu.
- In the **Add View** dialog, accept all the default settings and click the **Add** button.



- You should see that a new razor view file named **NewReport.cshtml** has been created in the **Views/Home** folder.
- Delete any existing code inside **NewReport.cshtml** and replace it with the following HTML code.

```
@model EmbeddedLab.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 after 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 create the **Reports** action method in the **Home** controller class to load an existing report that has just been created.

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

- a) In **PbiEmbeddedManager.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
    };
}
```

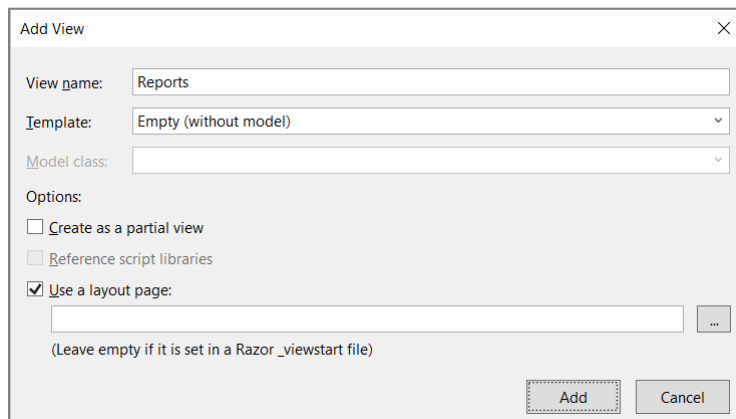
17. Add a new action method to the **HomeController** class named **Reports**.

- 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 PbiEmbeddedManager.GetEmbeddingDataForReport(reportId);
    return View(embeddingData);
}
```

18. Create a razor view for the **Reports** action method.

- a) Right-click on the **Reports** 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.



The 'Add View' dialog box is shown with the following fields and options:

- View name:** Reports
- Template:** Empty (without model)
- Model class:** (empty dropdown)
- Options:**
  - ☐ Create as a partial view
  - ☐ Reference script libraries
  - ☒ Use a layout page:
    - Text input field (empty)
    - Buttons: ...
- Footer:** (Leave empty if it is set in a Razor \_viewstart file)
- Buttons:** Add, Cancel

A page break has been inserted here to prevent the following code section from wrapping across pages.

- 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 EmbeddedLab.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
        });
    });

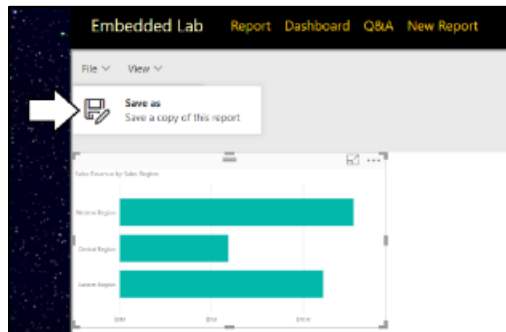
    $("#fullScreen").click(function () {
        report.fullscreen();
    });

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

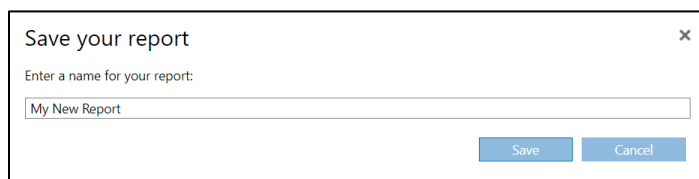
</script>
```

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

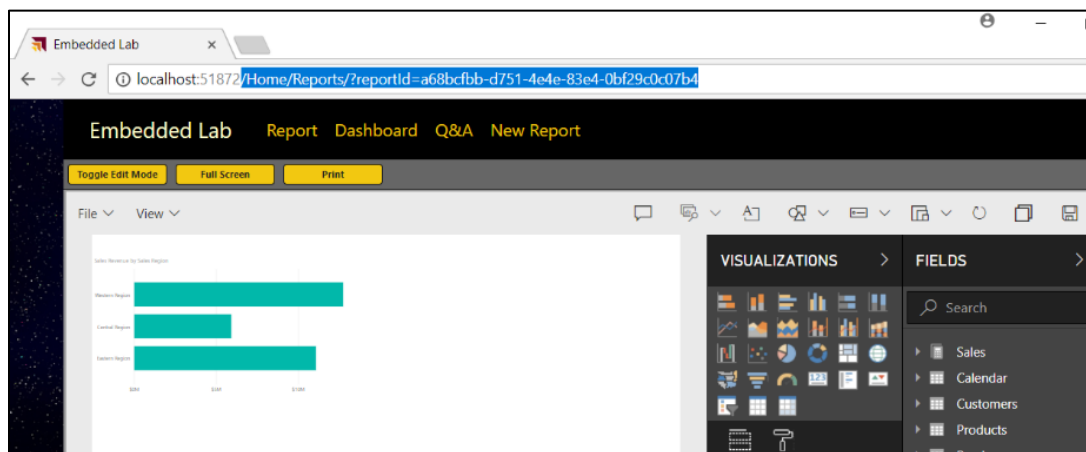
- Press the **{F5}** key in Visual Studio to begin a new debugging session.
- Click the **New Report** link in the top navigation menu and you should see an new empty in design mode.
- Add a simple visual to the new report.
- Save the new report by dropping down the **File** menu and selecting the **Save as** command.



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



- 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.



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