

# Developing SharePoint-hosted Add-ins

**Lab Time:** 40 minutes

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

**Lab Overview:** In this lab you will create a new Developer site and also create two new SharePoint-hosted Add-in projects to get some experience developing and testing SharePoint-hosted add-ins and custom app parts.

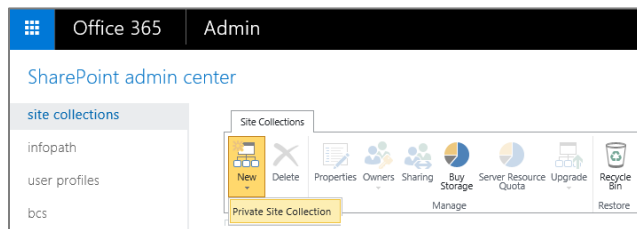
## Exercise 1: Creating a New Developer Site for Testing

In this exercise you will prepare your Office 365 development environment by ensuring you have access to a SharePoint Developer site that's been created within the tenancy associated with your Office 365 developer account.

1. If you have already created a Developer site in SharePoint online, navigate to this Developer site in the browser to ensure that you can log in with your Office 365 developer account credentials and see the home page. Once you can access the Developer site with the browser, you can move ahead to exercise 2.
2. If you have not already created a Developer site in SharePoint online, you must do so before continuing to the next exercise.
  - a) Using the browser, navigate to the SharePoint admin center (i.e. tenant admin site collection) for your SharePoint Online tenancy at the following URL.

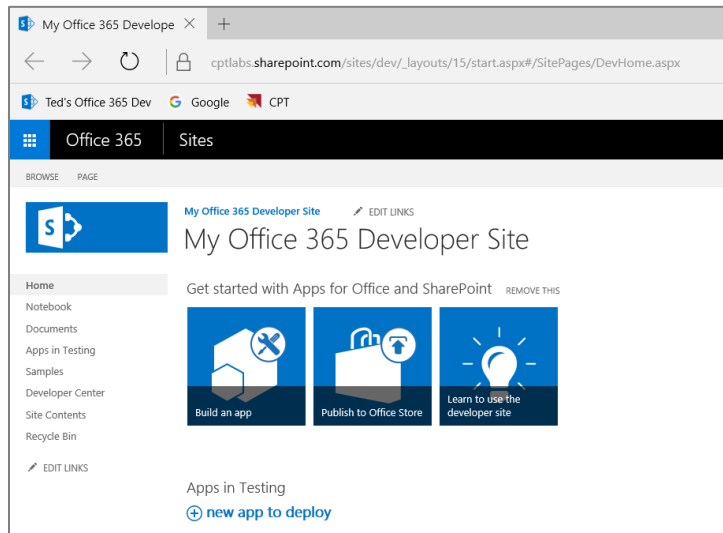
**[https://\[YOUR\\_TENANCY\].admin.sharepoint.com](https://[YOUR_TENANCY].admin.sharepoint.com)**

- b) With **site collections** selected on the left-side of the page, drop down the **New** menu and click **Private Site Collection**.



- c) In the **new site collection** form, fill in the required information and select the site template titled **Developer Site**. Click the **OK** button to create the new Developer site.

- d) Wait for the site collection to be created. This may take a minute or two. Once the Developer site has been created, navigate to this site in the browser to ensure you can get to the home page.

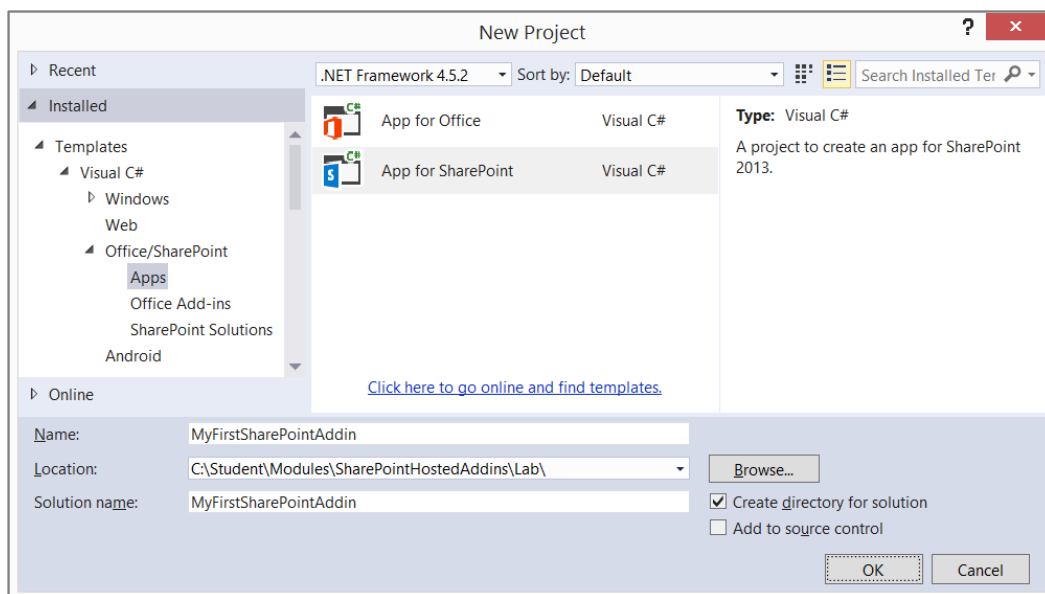


Now that you have a Developer site, you can begin to use it as you develop, test and debug SharePoint add-ins.

## Exercise 2: Creating and Debugging a SharePoint-hosted Add-in

In this exercise you will create and test a very simple SharePoint-hosted add-in project.

1. Create a new project in Visual Studio 2015:
  - a) Launch **Visual Studio 2015** as administrator.
  - b) In Visual Studio select **File → New → Project**.
  - c) In the **New Project** dialog, find the **App for SharePoint** template (Templates → Visual C# → Office / SharePoint → Apps)
  - d) **Name:** MyFirstSharePointHostedAddin
  - e) **Location:** C:\Student\Modules\SharePointHosted\Lab
  - f) **Solution name:** MyFirstSharePointHostedAddin
  - g) Click **OK** to create the new project.



- h) In the **New App for SharePoint** wizard, use the following values to complete the wizard and click **Finish**.
- i) **What site do you want to use for debugging?** *[Enter the URL for your Developer site]*
  - ii) **How do you want to host your app for SharePoint?** SharePoint-hosted

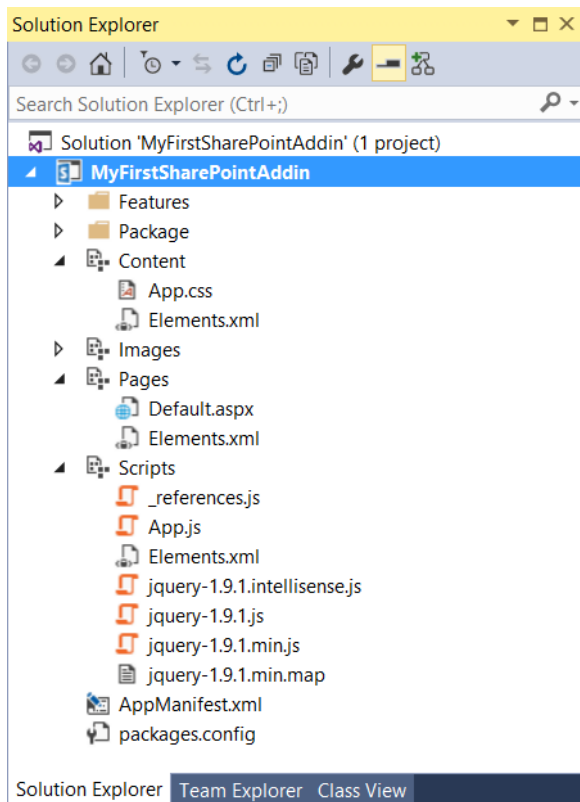
The screenshot shows the 'New app for SharePoint' wizard window. The title bar says 'New app for SharePoint'. The main heading is 'Specify the app for SharePoint settings'. Below this, there is a question: 'What SharePoint site do you want to use for debugging your app?'. A text box contains the URL 'https://cptlabs.sharepoint.com/'. Below this, there is a link: 'Don't have a developer site? Sign up for an Office 365 Developer site to develop, test and deploy apps for Office and SharePoint'. Then, there is another question: 'How do you want to host your app for SharePoint?'. There are two radio buttons: 'Provider-hosted' and 'SharePoint-hosted'. The 'SharePoint-hosted' radio button is selected. Below this, there is a link: 'Learn more about this choice'. At the bottom, there are four buttons: '< Previous', 'Next >', 'Finish', and 'Cancel'. The 'Next >' button is highlighted with a blue border.

If you are prompted to log in, enter your credentials to log in using your Office 365 developer account.

- i) On the **Specify the target SharePoint version** page, accept the default selection of **SharePoint Online** and click **Finish**.

The screenshot shows the 'New app for SharePoint' wizard window. The title bar says 'New app for SharePoint'. The main heading is 'Specify the target SharePoint version'. Below this, there is a text block: 'Choose the earliest version of SharePoint that you want to target. We'll add the right file references to your project so that you can access the APIs that are available in that version.' Then, there is a question: 'What's the earliest version of SharePoint that you want your app to target?'. There are two radio buttons: 'SharePoint 2013' and 'SharePoint Online'. The 'SharePoint Online' radio button is selected. At the bottom, there are four buttons: '< Previous', 'Next >', 'Finish', and 'Cancel'. The 'Finish' button is highlighted with a blue border.

- j) Wait for Visual Studio to complete the process of creating the new SharePoint add-in project.
2. Take a moment and inspect the file and folder structure of the new project.



3. Here are a few things you should observe about the new SharePoint-hosted add-in project.
  - a) Like a traditional SharePoint solution-based project you have a **Features** and **Packages** node.
  - b) There are four folders named **Content**, **Images**, **Pages** and **Scripts** which are a special type of project item known as SharePoint Modules which provision their files to the respective folders in the add-in web during add-in installation.
  - c) **Content/App.css** is an empty Cascading Style Sheet file which can be used to style the appearance of the add-in.
  - d) **Images/AppIcon.png** is the default image used for the add-in.
  - e) **Pages/Default.aspx** is the default start page for the add-in.
  - f) **Scripts/\_references.js** is an Intellisense support file for JavaScript libraries. This file isn't provisioned to SharePoint.
  - g) **Scripts/App.js** is the main file that containing JavaScript code which controls the behavior and core logic for your add-in.
  - h) The **Scripts** folder contains source files for the jQuery library.
  - i) **AppManifest.xml** is the add-in manifest. It tells SharePoint the basic information it needs about the add-in such as:
    - i) Name, Product ID, App Version Number and minimum version for the SharePoint host environment.
    - ii) Security configuration and permissions.
    - iii) App Title to display on app launcher tile on Site Contents page of the host web.
    - iv) The URL of the app's start page.
4. Customize the start page for the SharePoint add-in:
  - a) Using the **Solution Explorer** tool window, right-click the **Pages/Default.aspx** file and select **Open**.
  - b) Inspect (*but do not modify*) the content placeholder with the ID of **PlaceHolderAdditionalPageHead**. You can see there are references to the jQuery library and to the **App.js** file as well as a reference to the **App.css** file as well.
  - c) Locate the **PlaceHolderPageTitleInTitleArea** placeholder and update its content to match the following code listing.
 

```
<asp:Content ContentPlaceHolderID="PlaceHolderPageTitleInTitleArea" runat="server">
    My First SharePoint Add-in
</asp:Content>
```
  - d) Locate the **PlaceHolderMain** placeholder and remove the content inside. Add content into the **PlaceHolderMain** placeholder to match the following code listing.

```
<asp:Content ContentPlaceHolderID="PlaceHolderMain" runat="server">
```

```

<div id="toolbar">
  <input type="button" id="cmdPressMe" value="Press Me" />
</div>

<div id="content_box" />

</asp:Content>

```

- e) Save and close **default.aspx**.
5. Add some CSS code to style the HTML elements that were added to the add-in start page.
  - a) Using the **Solution Explorer** tool window, right-click the **Content/app.css** file and select **Open**.
  - b) Update the contents of the app.css file to match the following code listing.

```

#toolbar {
  border: black solid 1px;
  border-radius: 8px;
  padding: 8px;
  background-color: #EEE;
}

#content_box {
  margin-top: 8px;
  font-size: 18px;
  color: blue;
}

```

- c) Save and close **app.css**.
6. Write a little JavaScript code to give your add-in some behavior:
  - a) Using the **Solution Explorer** tool window, right-click the **Scripts / app.js** file and select **Open**.
  - b) Update the contents of the app.js file to match the following code listing.

```

'use strict';

$(onPageLoad);

function onPageLoad() {
  $("#cmdPressMe").click(onPressMe);
}

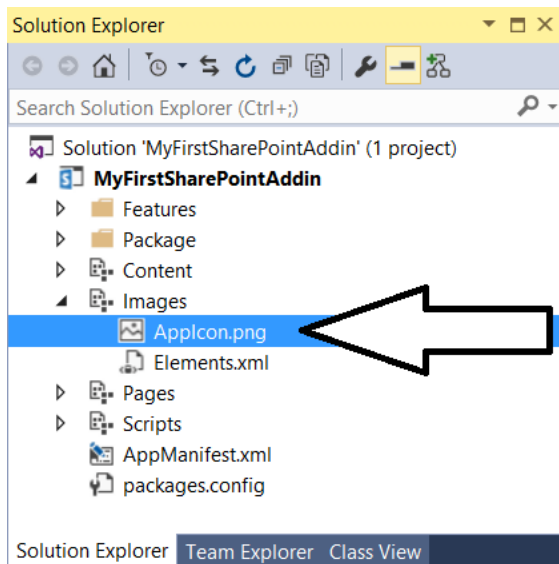
function onPressMe() {
  $("#content_box").text("Hello SharePoint Add-ins");
}

```

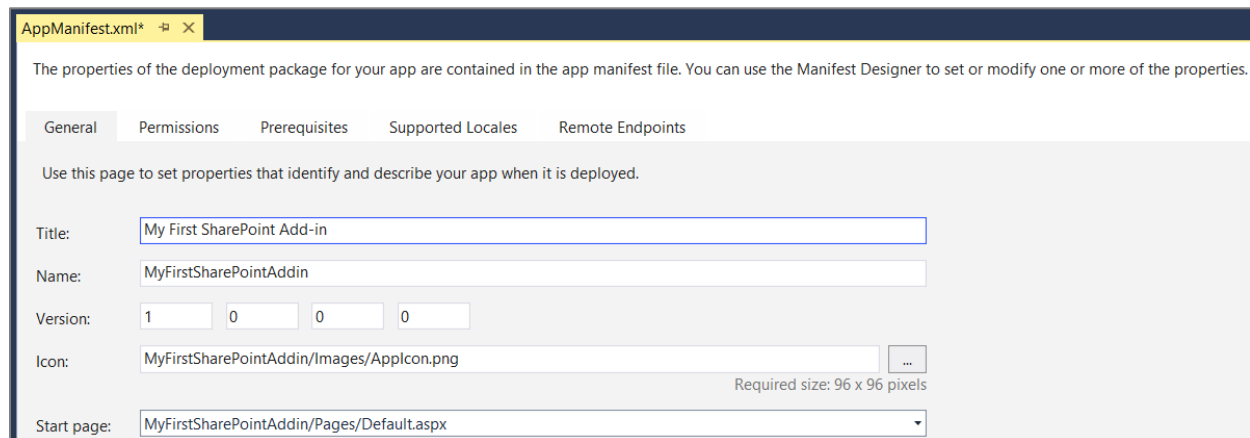
- c) Save and close **app.js**.
7. Replace the add-ins icon with a custom icon.
  - a) Open Windows explorer and locate the following file in the **Student** folder.

```
C:\Student\ExtraStudentFiles\Images\AppIcon.png
```

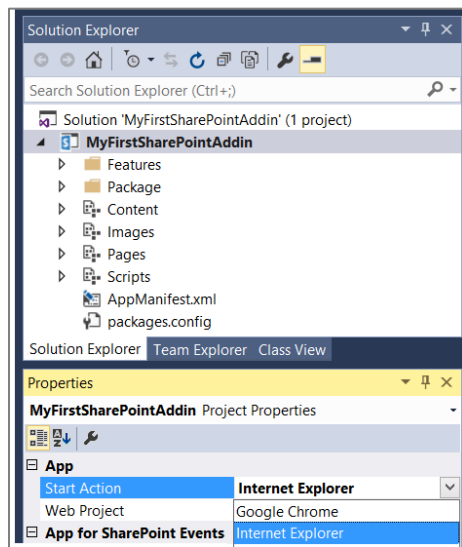
- b) Use the **AppIcon.png** file from the **Student** folder to replace the **AppIcon.png** file in the **Images** folder of the add-in project.



8. Modify the add-in manifest to provide a better title for the add-in:
  - a) Using the **Solution Explorer** tool window, right-click the **AppManifest.xml** file and select **Open**.
  - b) Update the add-in **Title** property to **My First SharePoint Add-in** so that it's more human-readable

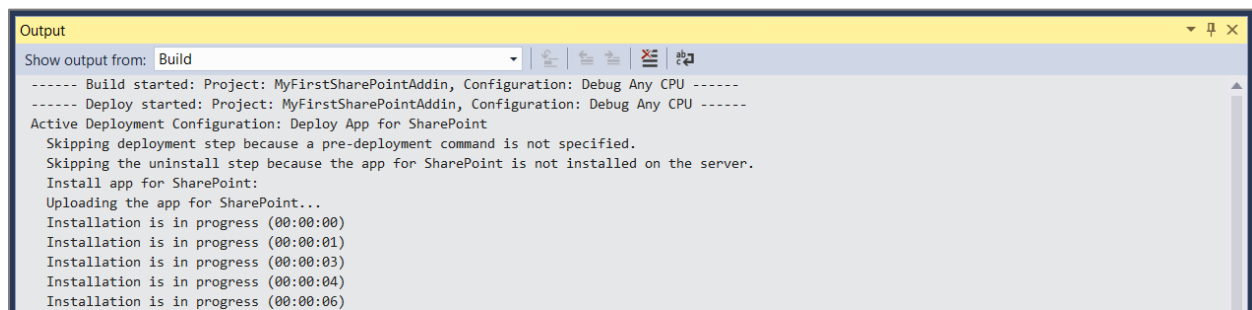


- c) Save and close **AppManifest.xml**.
9. Configure which browser to use when debugging the add-in project with the Visual Studio debugger.
  - a) In Solution Explorer, select the top-level node for the **MyFirstAddinProject**.
  - b) Locate the project-level **Start Action** property in the Properties window.
  - c) Set the **Start Action** property to the browser of your choosing such as **Internet Explorer** or **Chrome**. Note that you can start one debugging session using Internet Explorer and then start the next debugging session using Chrome.

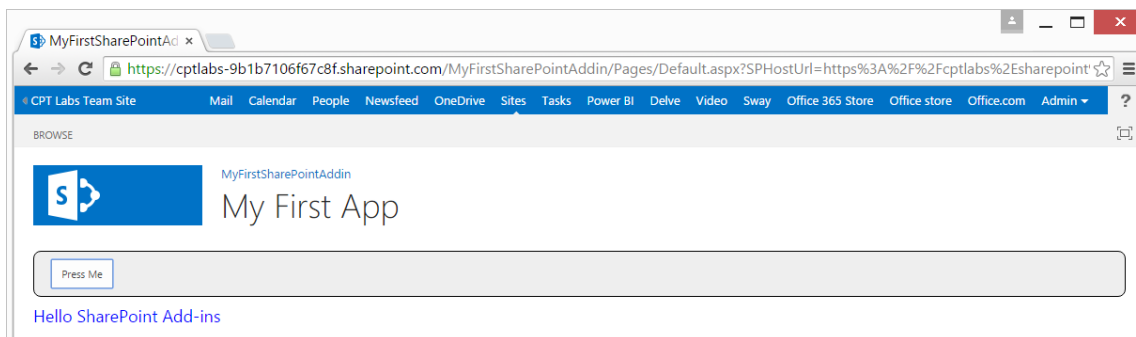


10. Build and test your application by pressing {F5} or **Debug** → **Start Debugging**.

- a) Pressing {F5} builds and add-in package and begin the add-in installation process. The installation process for an add-in can take a minute or two to complete the first time you press {F5}. You can monitor the debugger's progress in the **Output** window.

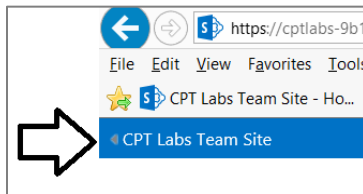


- b) Once the solution has been deployed, Visual Studio launches a browser session and you should be redirected to the add-in start page which is **default.aspx**.
- c) When the page loads, click the **Push me!** button to see your text get written to the page:

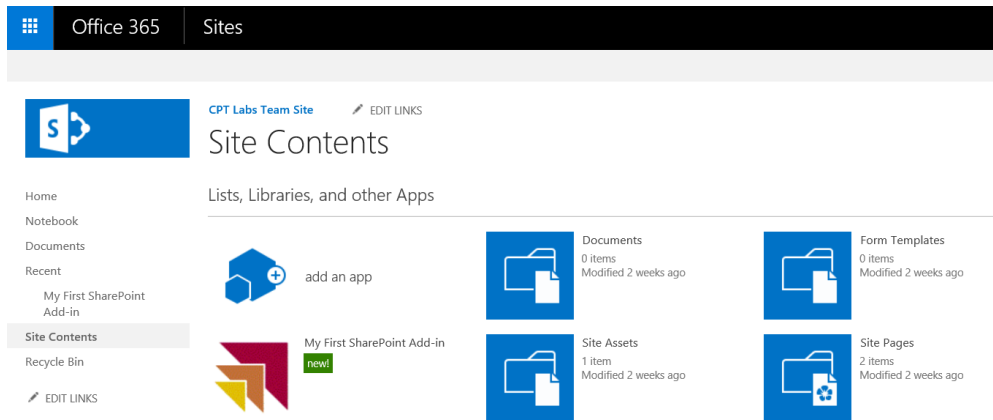


11. Return to the host web and see where SharePoint Online displays the custom logo for this add-in.

- a) Click the link on the far left of the link bar to navigate to the host web.



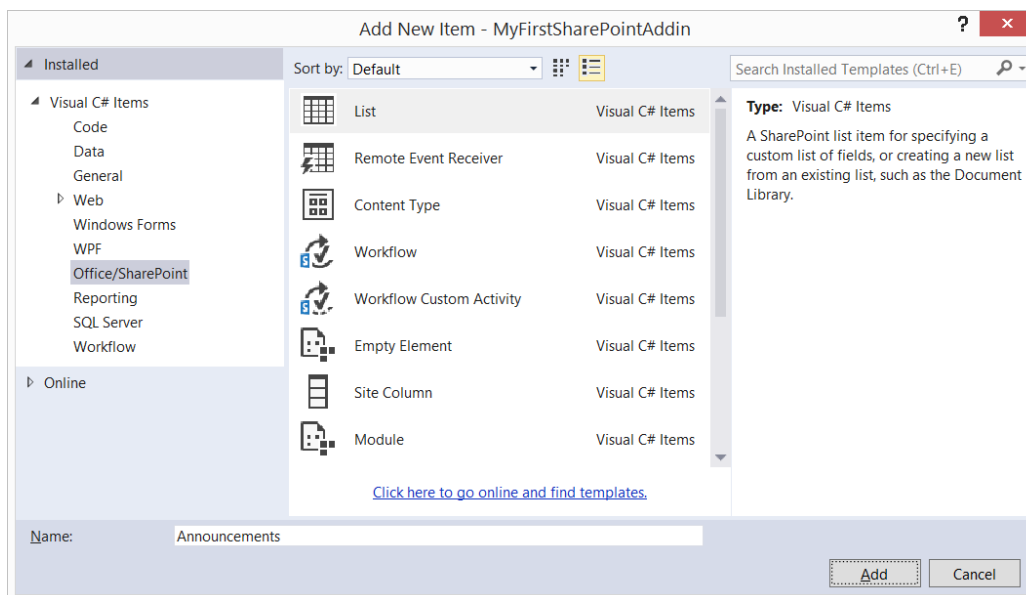
- b) On the site's **Quick Launch** navigation (on the left side of the screen) click on **Site Contents**.
- c) On the **Site Contents** page notice the icon for the app we just deployed My First SharePoint Hosted App.



12. Close the browser to stop the debugger and return to Visual Studio.

## Configure the SharePoint-hosted add-in To Create a List in the Add-in Web

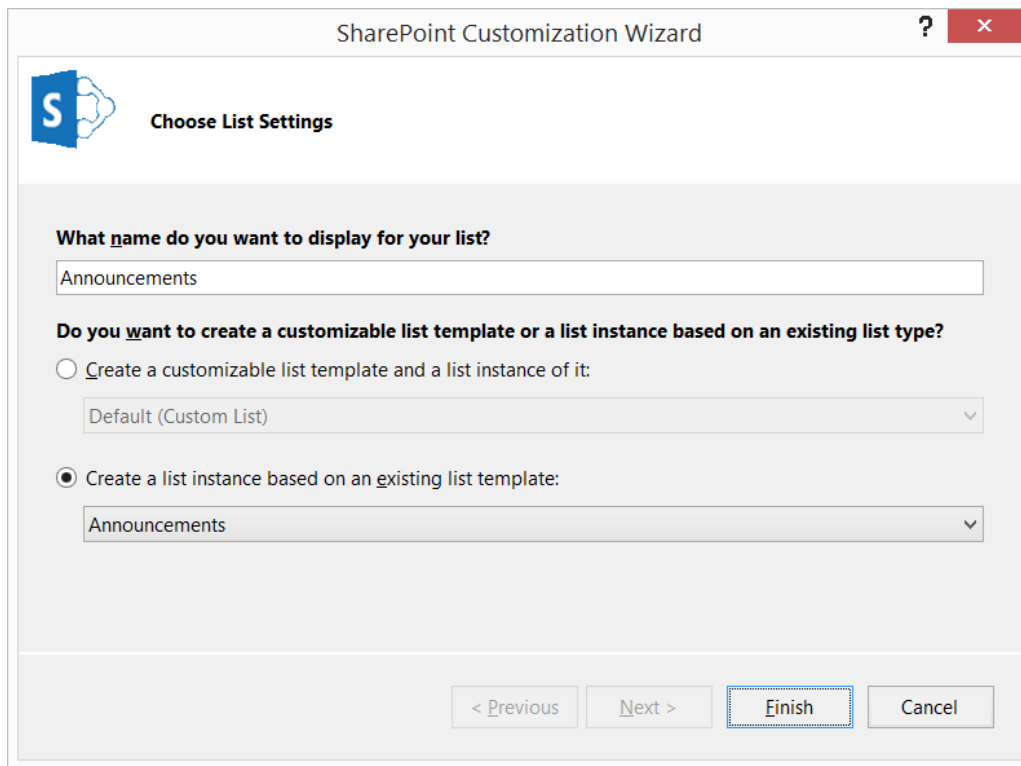
13. Add a new project item to create a SharePoint list in the add-in web during installation:
  - a) Using the **Solution Explorer**, right-click the **My First SharePoint Hosted App** project and select **Add → New Item**.
  - b) In the **Add New Item** dialog, select the **List** template found in the **Visual C# Items / Office / SharePoint** category. Enter a name of **Announcements** and click **Add**.



- c) In the **SharePoint Customization Wizard** dialog, use the following values to complete the form and click **Finish**:
  - i) **What name do you want to display for your list?** Announcements



- ii) **Do you want to create a customizable list template or a list instance based on an existing list type:** Create a list instance based on an existing list template: **Announcements**



The image shows the 'SharePoint Customization Wizard' window with the 'Choose List Settings' tab selected. The wizard has a title bar with a question mark and a close button. The main content area has a header with the SharePoint logo and the title 'Choose List Settings'. Below this, there are two sections. The first section is titled 'What name do you want to display for your list?' and has a text box containing 'Announcements'. The second section is titled 'Do you want to create a customizable list template or a list instance based on an existing list type?'. It has two radio buttons. The first radio button is labeled 'Create a customizable list template and a list instance of it:' and is currently unselected. Below it is a dropdown menu showing 'Default (Custom List)'. The second radio button is labeled 'Create a list instance based on an existing list template:' and is currently selected. Below it is a dropdown menu showing 'Announcements'. At the bottom of the wizard, there are four buttons: '< Previous', 'Next >', 'Finish', and 'Cancel'. The 'Finish' button is highlighted with a blue border.

14. Save all changes: **File → Save All**.

15. Open the file default.aspx and add the following HTML link to after the `<div id="content_box"></div>` ta.

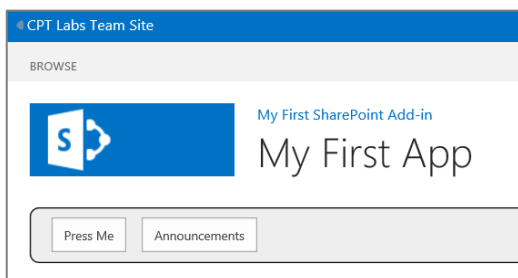
```
<div id="toolbar">
  <input type="button" id="cmdPressMe" value="Press Me" />
  <input
    type="button"
    value="Announcements"
    onclick="JavaScript: location.href = '../Lists/Announcements'" />
</div>
```

## Build and Test the Project

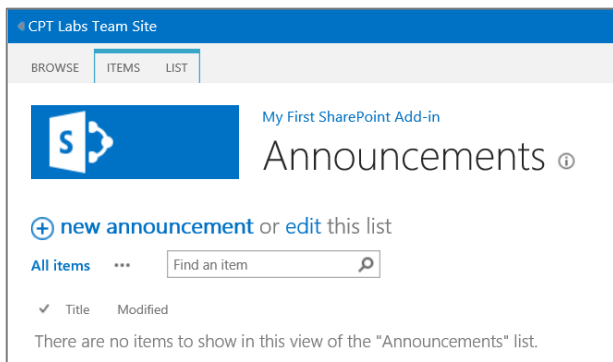
16. Build and test your application by pressing **[F5]** or **Debug → Start Debugging**.

17. Once the solution has been deployed, Internet Explorer will launch and navigate to the add-in start page.

18. Notice there is nothing in the user interface of the add-in to link to the Announcements list except for the link button that you added.

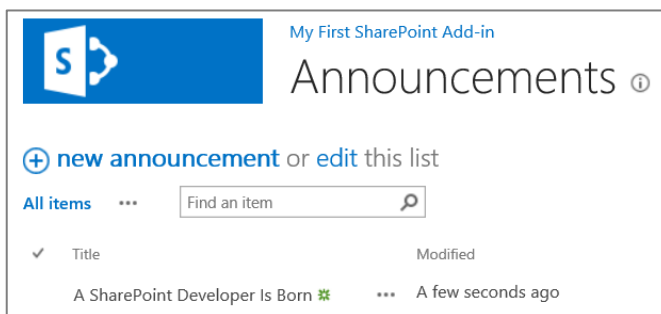


19. Click the **Announcements** link button to navigate to the **Announcements** list.



20. Click the **new announcement** link button to display a form for creating a new announcement. Create new announcement item using test date like that shown in the following screenshot. Click the **Save** button to create the new item.

21. Once you save the item, you should be able to see it listed in the default view for the **Announcements** list.



22. Close the browser to stop the debugger and return to Visual Studio.

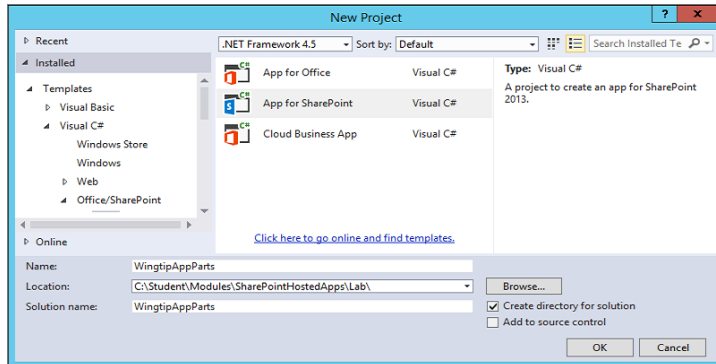
In this exercise you created a simple SharePoint-hosted add-in and made some basic customizations to it. Later modules and labs will build upon this foundation (e.g. working with the CSOM and REST API's, customizing the user interface, and creating robust client-side code with additional permissions).

### Exercise 3: Creating the Hello World App Part

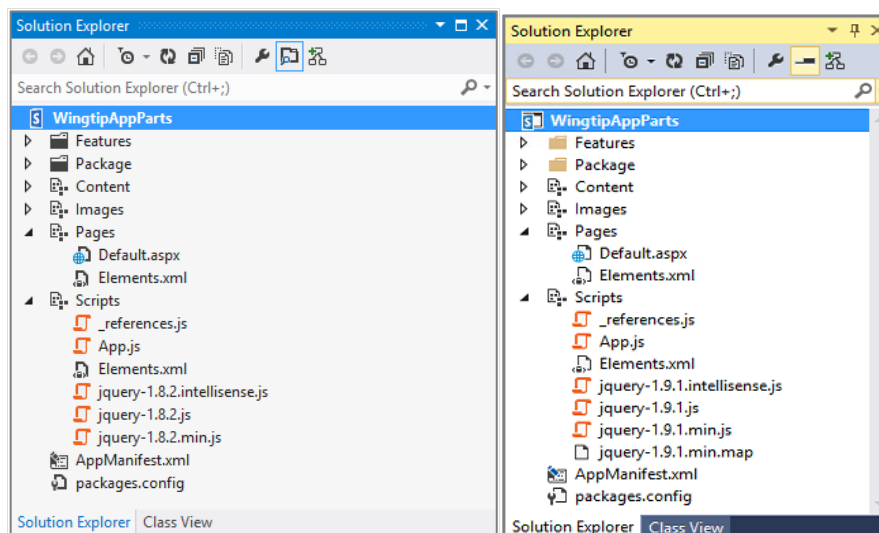
In this exercise you will create a new SharePoint-hosted add-in with a simple app part.

1. Create a new SharePoint App project in Visual Studio 2015:
  - a) In Visual Studio select **File → New → Project**.

- b) In the **New Project** dialog:
  - i) Find the **App for SharePoint** template under the **Templates → Visual C# → Office / SharePoint → Apps** section.
  - ii) **Name:** WingtipAppParts
  - iii) **Location:** C:\Student\Modules\SharePointHostedApps\Lab
  - iv) Click **OK**.



- c) In the **New app for SharePoint** dialog, choose the option to create a SharePoint-hosted app and make sure the URL is the one for your SharePoint Developer site.
- d) Once the new project has been created, examine its structure and the source files inside.



2. Open the **AppManifest.xml** file by double clicking on it. Change the **Title** to Wingtip App Parts. Save and close the **AppManifest.xml** file.
3. Open the **App.js** file in the **Scripts** folder. Delete all the contents from **App.js** leaving it as an empty file for now. Save your changes to **App.js** and close the file.
4. Open **Default.aspx** in Code View. Replace the content inside the two placeholders named **PlaceHolderPageTitleInTitleArea** and **PlaceHolderMain** with the following HTML code.

```
<asp:Content ContentPlaceHolderID="PlaceHolderPageTitleInTitleArea" runat="server">
    Wingtip App Parts
</asp:Content>

<asp:Content ContentPlaceHolderID="PlaceHolderMain" runat="server">

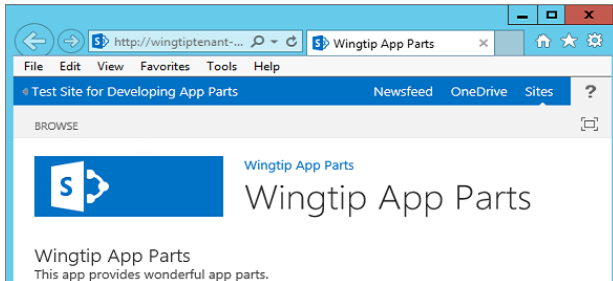
    <h2>Wingtip App Parts</h2>
    <div>This app provides wonderful app parts.</div>

</asp:Content>
```

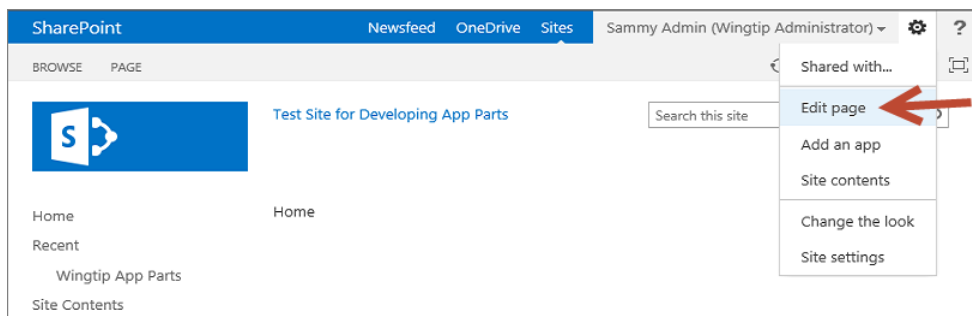
5. Save your changes to **Default.aspx** and then close this file.

Remember that the purpose of this lab exercise is to create app parts. However, the app still requires a start page even if the start page doesn't really provide any real functionality. However, the start page is helpful for testing because it provides a link back to the host web where you will be testing and debugging your app parts. When you launch a debugging session, you should become familiar with the process of redirecting from the app start page back to the host web so you can create an instance of your app parts for testing and debugging purposes.

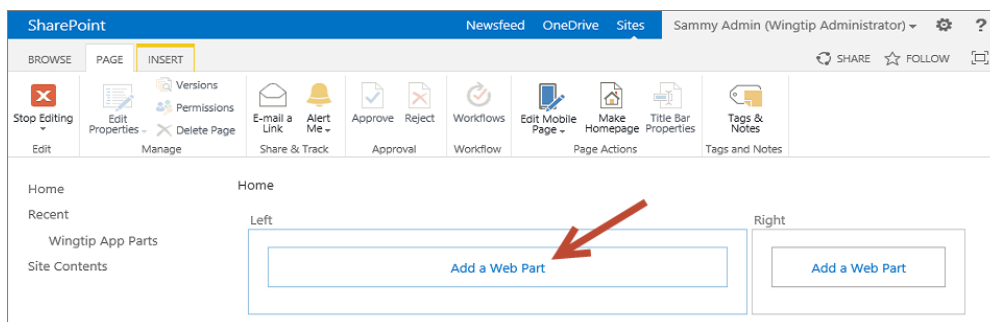
6. Test your work by pressing the **{F5}** key to launch a debugging session. When the app starts, you should see the start page appear as the one shown in the following screenshot. Leave this start page open for the next step.



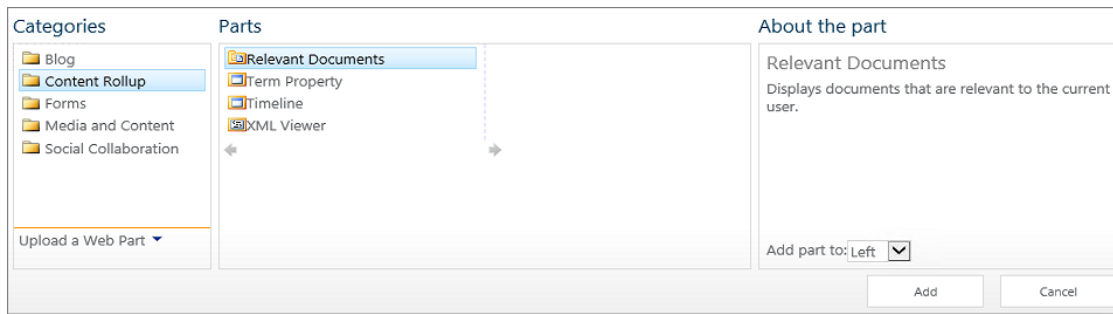
7. Navigate to the host web and see what web parts are available out of the box.
- Click the link in the top left corner of the start page to navigate back to the host web. This should redirect you to the home page of the Blank site at <http://apppart.wingtip.com>.
  - Drop down the **Site Actions** menu and select the **Edit page** command to move the page into Edit Mode.



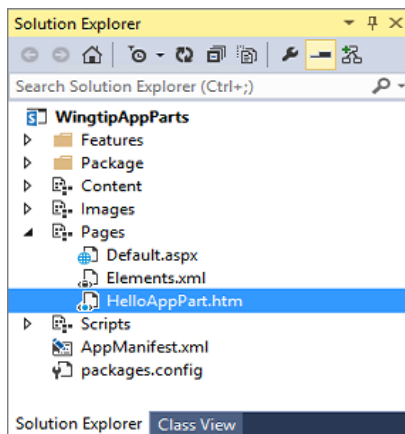
- c) Once the page is in Edit Mode, you should see two web part zones. Click the **Add a Web Part** link in the left web part zone. This action will display the web part catalog.



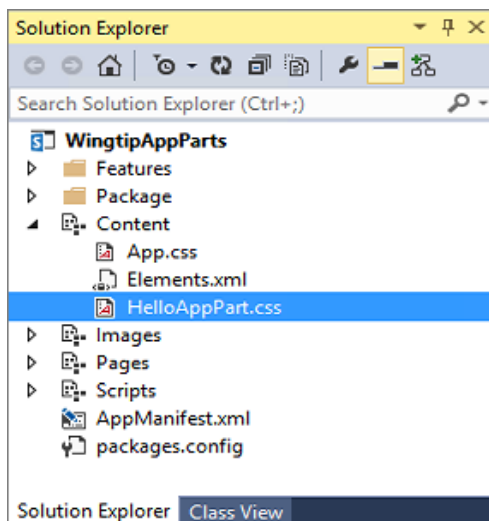
- d) While you do not need to add a web part to the page in this step, your objective is simply to see what web parts are available. In just a bit, you will see your custom app parts available in this web part catalog.



- e) Once you have looked through the available set of out-of-the-box web parts, close the browser to end the debugging session.
8. Return to Visual Studio.
9. Create an HTML page for an app part.
  - a) Add a new HTML page to the **Pages** folder named **HelloAppPart.htm**.
    - i) In the **WingtipAppParts** project right click on the **Pages** folder and select **Add → New Item...**
    - ii) In the **Add New Item** dialog box, Select **Visual C# → Web** from the categories on the left side then select **HTML Page** from the templates in the middle and give this page the name: **HelloAppPart.htm**



- b) Add a new CSS file to the **Content** folder named **HelloAppPart.css**.
  - i) In the **WingtipAppParts** project right click on the **Content** folder and select **Add → New Item...**
  - ii) In the **Add New Item** dialog box, Select **Visual C# → Web** from the categories on the left side then select **Style Sheet** from the templates in the middle and give this page the name: **HelloAppPart.css**



- c) Modify the contents of **HelloAppPart.css** to look like the following CSS listing.

```
body {  
    background-color: yellow;  
}  
  
h4 {  
    color: blue;  
    border-bottom: 1px solid blue;  
}
```

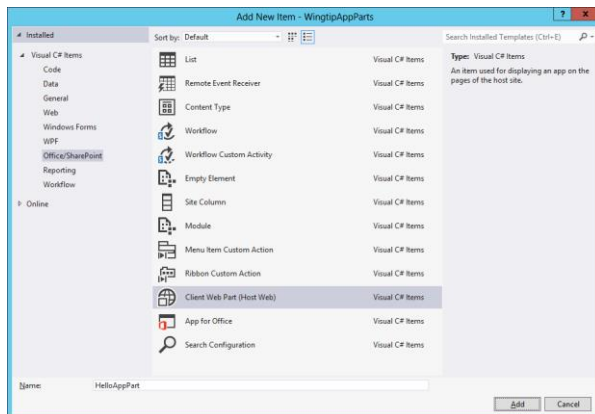
- d) Save and close **HelloAppPart.css**.
- e) Open **HelloAppPart.htm** and modify the HTML contents to look like the following HTML listing. Be sure to include a link to the CSS file named **HelloAppPart.css**.

```
<!DOCTYPE html>  
  
<html lang="en" xmlns="http://www.w3.org/1999/xhtml">  
  
  <head>  
    <meta charset="utf-8" />  
    <title></title>  
    <link href="../Content/HelloAppPart.css" rel="stylesheet" />  
  </head>  
  
  <body>  
  
    <h4>Hello App Part Content</h4>  
    <div>This content lives in the app web</div>  
  
  </body>  
</html>
```

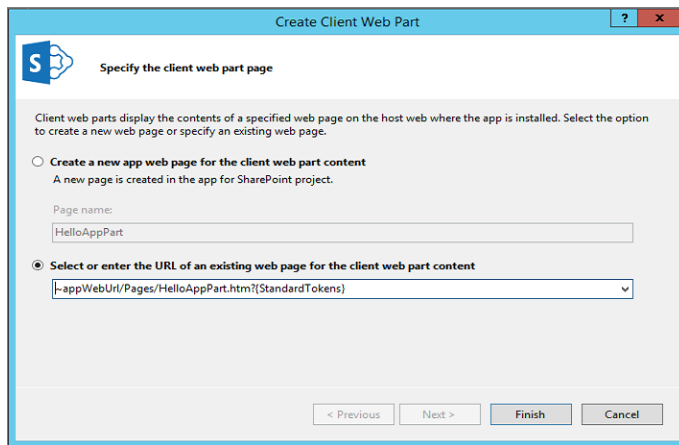
- f) Save and close **HelloAppPart.htm**.

10. Create a new app part which will use the page **HelloAppPart.htm** to display its contents.

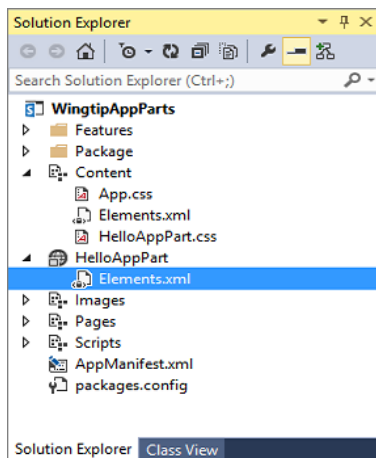
- a) In the Solution Explorer, right-click on the **WingtipAppParts** project and select the **Add New Item** command.
- b) In the **Add New Item** dialog, select the **Client Web Part (Host Web)** project item template and give it the name **HelloAppPart**.



- c) Click the **Add** button at the bottom right of the **Add New Item** dialog to add the new Client Web Part project item. When you click the Add button, you should then see the **Create Client Web Part** dialog.
- d) In the **Create Client Web Part** dialog, select the option **Select or enter a URL for an existing web page**. Then use the drop down list to select the **HelloAppPart.htm** page in the **Pages** folder.



- e) Click the **Finish** button in the **Create Client Web Part** dialog to complete the process of adding the new Client Web Part project item.
- f) Once the Client Web Part project item has been created, you can see that Visual Studio has created a folder for it in the project. This folder contains a file named **elements.xml**.



- g) Modify the **elements.xml** file for the new **HelloAppPart** Client Web Part to match the XML in the following code listing.

```
<?xml version="1.0" encoding="utf-8"?>
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
  <ClientWebPart
    Name="HelloAppPart"
    Title="The Hello App Part"
    Description="A simple little app part"
    Defaultwidth="600" DefaultHeight="200">

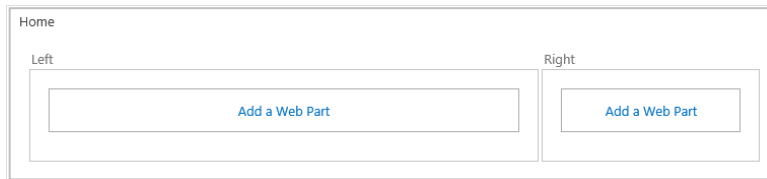
    <Content
      Type="html"
      Src="~appWebUrl/Pages/HelloAppPart.htm?{StandardTokens}" />

    <Properties>
    </Properties>

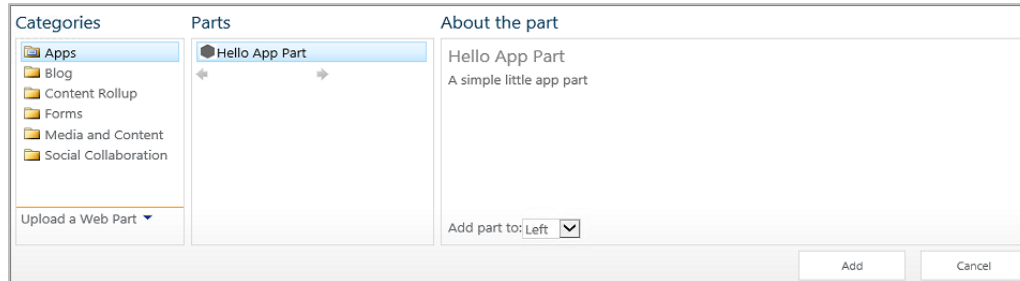
  </ClientWebPart>
</Elements>
```

- h) Save and close the **elements.xml** file.
11. Test your work by adding the **HelloAppPart** app part to a web part page in the host web.
    - a) Press **{F5}** to begin a debugging session.
    - b) When you see the app's start page, click the link to redirect to the home page of the host web. (Reminder: this link is in the top left corner of the page)

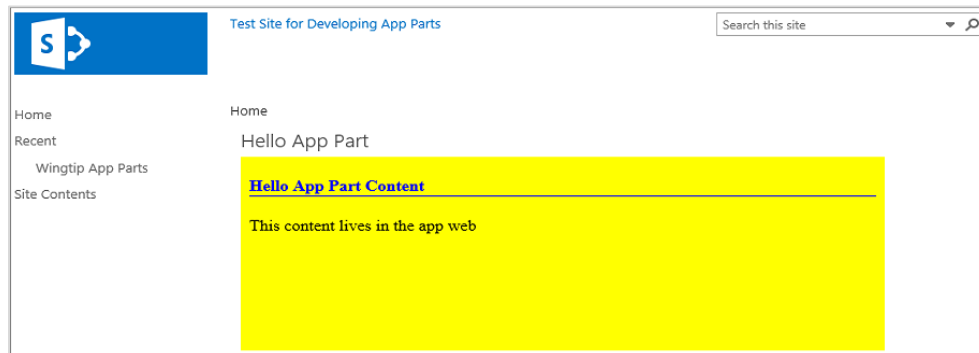
- c) Use the **Edit page** menu from the **Site Actions** menu to move the page into Edit Mode.
- d) Once you are in Edit Mode, click the **Add a Web Part** link in the left web part zone to display the web part catalog.



- e) Locate and select the app part with a title of **The Hello App Part** in the **Apps** category folder. Click the **Add** button on the bottom right-hand side of the web part catalog to add the app part to the home page of the host web.



- f) In the **Ribbon Bar Page** Tab click **Stop Editing**. Now click the **Browse** Tab in the Ribbon to see your completed page with the app part.
- g) After you have added the app part, you should be able to see it on the home page of the host web.



- h) Close the browser to end the debugging session and then return to Visual Studio.

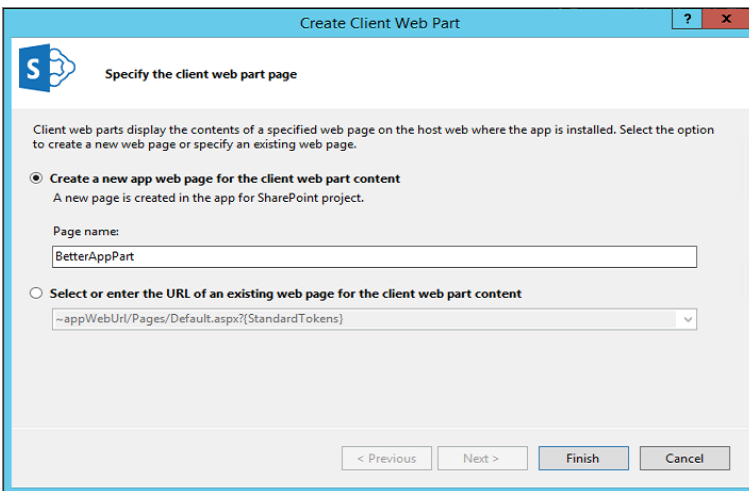
Now you have created and tested a simple app part based on an HTML page. Next, you will create a more complicated app part with custom app part properties which is implemented with an ASPX file instead of a simple HTML file.

## Exercise 4: Creating an App Part with Custom Properties

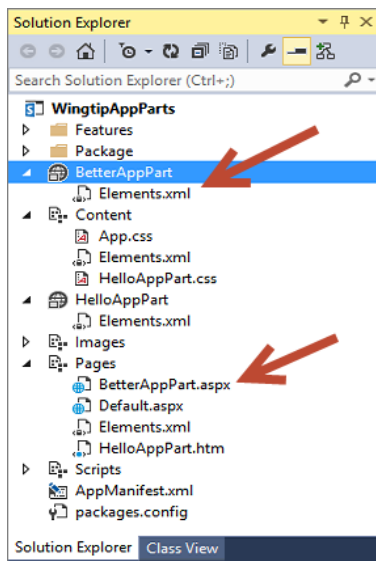
In this exercise you will create and test an app part with custom properties.

1. Continue working in the same **WingtipAppParts** project you created in the previous exercise.
2. Create a new app part named **BetterAppPart**,
  - a) Right-click on the **WingtipAppParts** project and select the **Add New Item** command.
  - b) In the **Add New Item** dialog, select the **Client Web Part (Host Web)** project item template (Located in the **Visual C# Items** → **Office/SharePoint** category) and give it a name of **BetterAppPart**. Click the **Add** button, you should then see the **Create Client Web Part** dialog.
  - c) In the **Create Client Web Part** dialog, accept the default settings and click **Finish**.





- d) Once the Client Web Part has been added, inspect what files have been added to the project. You should see that Visual Studio created a folder named **BetterAppPart** for the project item which contains an **elements.xml** file which defines the Client Web Part. In addition, an **aspx** page named **BetterAppPart** has been added to the **Pages** folder.



3. Open the **elements.xml** file in the **BetterAppPart** and modify its content to look like this.

```
<?xml version="1.0" encoding="utf-8"?>
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">

  <ClientWebPart
    Name="BetterAppPart"
    Title="Better App Part"
    Description="A really nice app part"
    Defaultwidth="600"
    DefaultHeight="200">

    <Content
      Type="html"
      Src="~appWebUrl/Pages/BetterAppPart.aspx?{StandardTokens}" />

    <Properties>
    </Properties>

  </ClientWebPart>
</Elements>
```

4. Save and close the **elements.xml** file.

5. Open **BetterAppPart.aspx** in Code View. Do not make any modifications to the **Page** directive, the **Register** directives or the **WebPartPages:AllowFraming** control at the top of the page. However, modify the HTML content below in the page to look like the code following listing. (i.e. this means you will remove all the <script> tags and associated script content (except for the jquery script tag) from the <head> section of the page in addition to adding content to the <body> section)

```
<%@ Page Language="C#" Inherits="Microsoft.SharePoint.WebPartPages.WebPartPage, BLAH BLAH BLAH %>

<%@ Register TagPrefix="SharePoint" Namespace="Microsoft.SharePoint.WebControls" BLAH BLAH BLAH %>
<%@ Register TagPrefix="Utilities" Namespace="Microsoft.SharePoint.Utilities" BLAH BLAH BLAH %>
<%@ Register TagPrefix="WebPartPages" Namespace="Microsoft.SharePoint.WebPartPages" BLAH BLAH BLAH %>

<WebPartPages:AllowFraming ID="AllowFraming" runat="server" />

<html>
<head>
  <title></title>

  <script type="text/javascript" src="../Scripts/jquery-1.9.1.min.js"></script>
</head>
<body>

  <h4>Better App Part Content</h4>
  <div id="results">default contents</div>

</body>
</html>
```

- a) Save your changes to the **BetterAppPart.aspx** file. (Note: Keep this file open we will need it later)
6. Add some JavaScript code for the app part.
- a) In Windows Explorer, look inside the folder at **C:\Student\Modules\SharePointHostedApps\Lab\StarterFiles** and locate the file named **wingtip.utilities.js**. Add this file into the **WingtipAppParts** project in the **Scripts** folder. (Note: you can accomplish this by dragging the file from the **Starter Files** source folder in File Explorer into the **Scripts** destination folder in the Solution Explorer in Visual Studio)
- b) Inspect what's inside of **wingtip.utilities.js**. As you can see, it is a JavaScript Module named **Wingtip.Utilities** that is very similar to the one you created in the JavaScript programming lab.

```
'use strict';

var wingtip = window.wingtip || {};

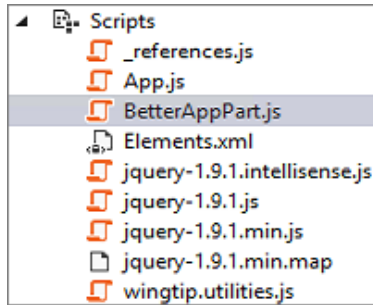
wingtip.Utilities = function () {

  var getQueryStringParameter = function (param) {
    var querystring = document.URL.split("?")[1];
    if (querystring) {
      var params = querystring.split("&");
      for (var index = 0; (index < params.length) ; index++) {
        var current = params[index].split("=");
        if (param.toUpperCase() === current[0].toUpperCase()) {
          return decodeURIComponent(current[1]);
        }
      }
    }
  }

  return {
    getQueryStringParameter: getQueryStringParameter,
  };
}();
```

- c) Close **wingtip.utilities.js**.
- d) Add a new JavaScript file into the **Scripts** folder named **BetterAppPart.js**.
- i) Right-click on the **Scripts** folder in the **WingtipAppParts** project in **Solution Explorer** and select the **Add New Item** command.
- e) In the **Add New Item** dialog, select the **JavaScript File** template (Located in the **Visual C# Items** → **Web** category) and give it a name of **BetterAppPart.js**. Click the **Add** button, you should then see the **Create New JavaScript File** dialog.

- f) In the **Create New JavaScript File** dialog, accept the default settings and click **Finish**



- g) Add the following JavaScript code to **BetterAppPart.js**.

```
$(function () {  
    $("#results").text("My dynamic content");  
  
    $("body").css({  
        "border": "2px solid #ccc",  
        "padding": "8px"  
    });  
  
    $(".header").css({"border-bottom": "1px solid black"});  
});
```

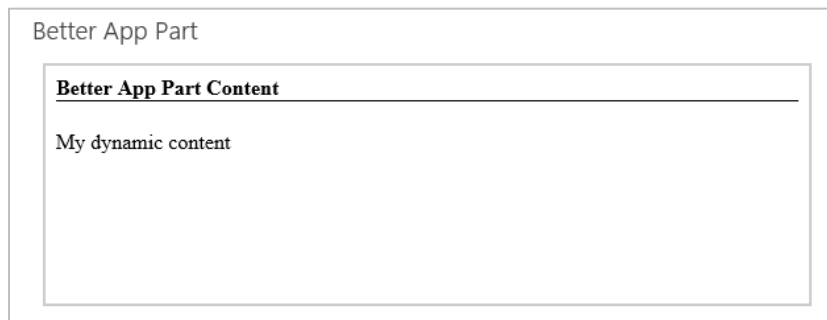
- h) Save and close the **BetterAppPart.js** file.

7. Open **BetterAppPart.aspx** and add the following script links into the **head** section of the page:  
(Note: you can do this quickly by dragging the JavaScript files from the Solution Explorer into the correct location in the BetterAppPart.aspx page)
- a) The jQuery library (verify that this is already there)  
(Note: the version number on this library may differ from the code below as it is frequently updated)
  - b) Wingtip.utilities.js
  - c) BetterAppPart.js

```
<head>  
  <title></title>  
  <script type="text/javascript" src="../../Scripts/jquery-1.9.1.min.js"></script>  
  <script src="../../Scripts/wingtip.utilities.js"></script>  
  <script src="../../Scripts/BetterAppPart.js"></script>  
</head>
```

- d) Save and close the **BetterAppPart.aspx** file.

8. Test your work by adding the **BetterAppPart** app part to a web part page in the host web.
- a) Press **{F5}** to begin a debugging session.
  - b) When you see the app's start page, click the link to redirect to the home page of the host web.
  - c) Use the **Edit page** menu from the **Site Actions** menu to move the page into Edit Mode.
  - d) Once you are in Edit Mode, click the **Add a Web Part** link in the left web part zone to display the web part catalog.
  - e) Locate and select the app part with a title of **Better App Part** in the **Apps** category folder. Click the **Add** button on the bottom right-hand side of the web part catalog to add the app part to the home page of the host web.
  - f) In the **Ribbon Bar Page Tab** click **Stop Editing**. Now click the **Browse** Tab in the Ribbon to see your completed page with the app part.
  - g) Once the app part is displayed, you should be able to verify that the JavaScript code executed properly to add the message "My dynamic content" and to add a bottom border on the heading **Better App Part Content**.



- h) Close the browser window to end the debugging session and return to Visual Studio.
9. Add two app part properties.
- a) Open the **elements.xml** file for the **BetterAppPart** app part. Add the two following property definitions.

```
<Properties>

  <Property
    Name="BackgroundColor"
    WebDisplayName="Add Background Color"
    Type="boolean"
    DefaultValue="false"
    WebCategory="Custom Wingtip Properties"
    RequiresDesignerPermission="true" >
  </Property>

  <Property
    Name="HeaderColor"
    WebDisplayName="Header Color"
    Type="enum"
    DefaultValue="Black"
    WebCategory="Custom Wingtip Properties"
    RequiresDesignerPermission="true" >
    <EnumItems>
      <EnumItem WebDisplayName="Black" Value="Black"/>
      <EnumItem WebDisplayName="Blue" Value="Blue"/>
      <EnumItem WebDisplayName="Green" Value="Green"/>
    </EnumItems>
  </Property>

</Properties>
```

- b) Inspect the Content element in **elements.xml**. Currently the Src attribute is defined as an URL which has a query string defined using only the dynamic token named **{StandardTokens}**.

```
<Content
  Type="html"
  Src="~appwebUrl/Pages/BetterAppPart.aspx?{StandardTokens}" />
```

- c) Modify the query string in the **elements.xml** file as shown here to pass the custom property values to **BetterAppPart.aspx**.

```
BetterAppPart.aspx?BackgroundColor=_BackgroundColor_&HeaderColor=_HeaderColor_&{StandardTokens}
```

- d) Save and close the **elements.xml** file.
- e) Return to **BetterAppPart.js** and add some code to read the two property values from the query string.

```
$(function () {

  $("#results").text("My dynamic content");

  $("body").css({
    "border": "2px solid #CCC",
    "padding": "8px"
  });

  $(".header").css({"border-bottom": "1px solid black"});

  var BackgroundColor = wingtip.Utilities.getQueryStringParameter("BackgroundColor");
```

```

if (BackgroundColor === "true") {
    $("body").css({ "background-color": "yellow" });
}

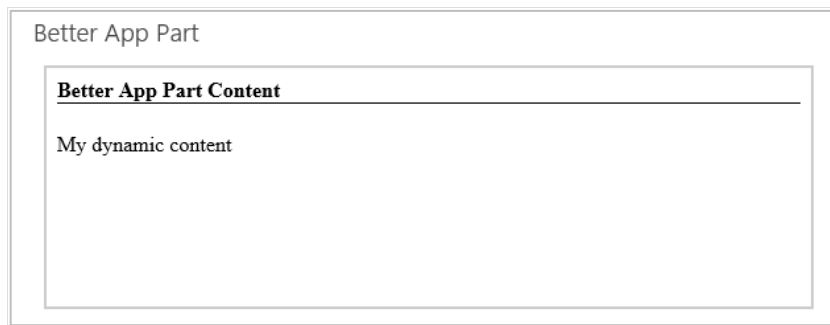
var HeaderColor = wingtip.Utilities.getQueryStringParameter("HeaderColor");
if (HeaderColor) {
    $(".header").css({ "color": HeaderColor });
    $(".header").css({ "border-bottom": "1px solid " + HeaderColor });
}
}
});

```

10. Save and close the **BetterAppPart.js** file.

11. Test your work.

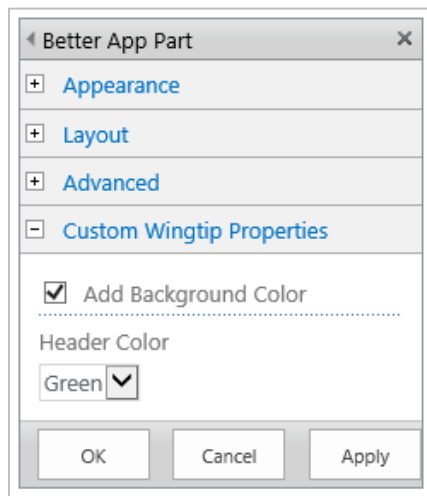
- Using the Visual Studio **Build** Menu select **Rebuild WingtipAppParts** to ensure the updated code will be deployed.
- Press **{F5}** to begin a debugging session.
- When you see the app's start page, click the link to redirect to the home page of the host web.
- Use the **Edit page** menu from the **Site Actions** menu to move the page into Edit Mode.
- Once you are in Edit Mode, click the **Add a Web Part** link in the left web part zone to display the web part catalog.
- Locate and select the app part with a title of **Better App Part** in the **Apps** category folder. Click the **Add** button on the bottom right-hand side of the web part catalog to add the app part to the home page of the host web.
- Once the app part is displayed, you should be able to verify that the JavaScript code executed properly to add the message "My dynamic content" and to add a bottom border on the heading **Better App Part Content**.



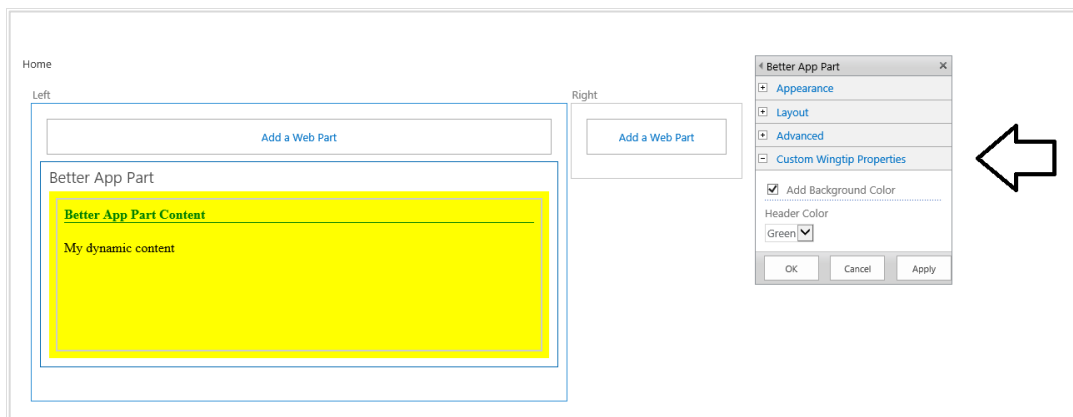
- As you are still in the Page Edit mode you can use the drop down app part menu in the top-right corner of the app part title bar to select the **Edit Web Part** menu. This will display the editor parts that make it possible for the user to modify app part properties.



- In the editor part for the **Better App Part**, locate and expand the **Custom Wingtip Properties** section.



- j) Enable the option to **Add Background Color**. Change the **Header Color** property to Green and then click the **Apply** button. You should see these changes affect the display the app part.

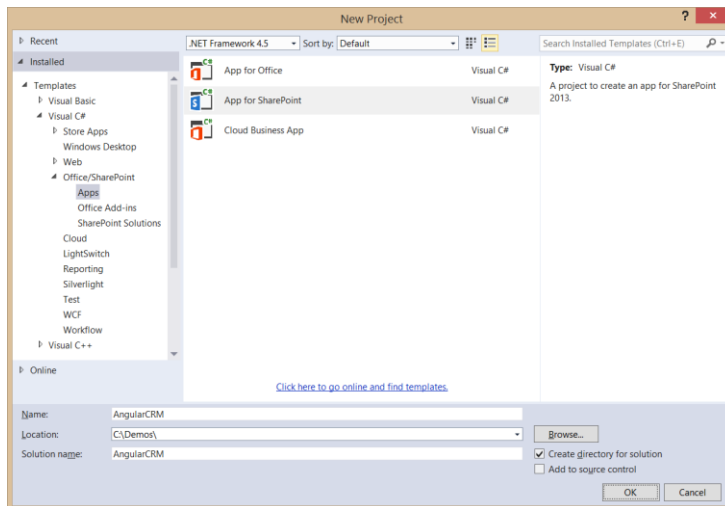


- k) When you are done with your testing, close the browser window to end the debugging session.

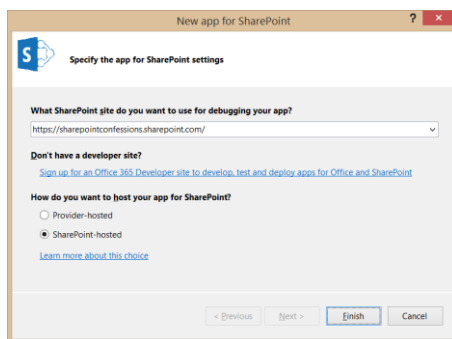
## Exercise 5: Creating a SharePoint-hosted App using Bootstrap and AngularJS

In this exercise you will create a SharePoint-hosted app and configure it to use Bootstrap and the AngularJS framework.

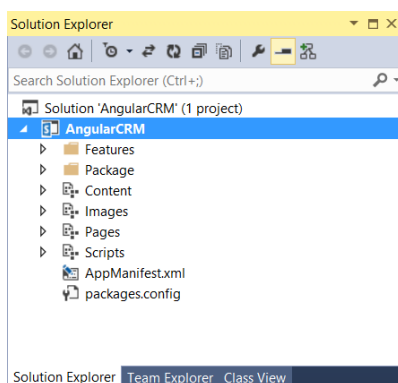
1. Launch Visual Studio as administrator.
2. Create a new project in Visual Studio by selecting the menu command **File > New > Project**.
3. In the **New Project** dialog
  - a) Select the **App for SharePoint** project template under the **Templates > Visual C# > Office / SharePoint > Apps** section.
  - b) Enter a name of **AngularCRM**
  - c) Enter a location of **C:\Student\Modules\Angular\Lab**.
  - d) Click the **OK** button.



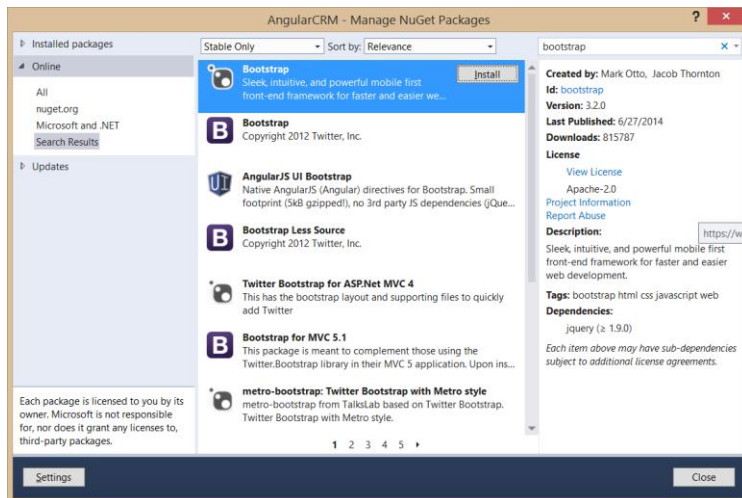
4. In the **New app for SharePoint wizard**, enter the URL for your SharePoint Developer site and select **SharePoint-hosted** for the app hosting model. When done, complete the wizard by clicking the **Finish** button.



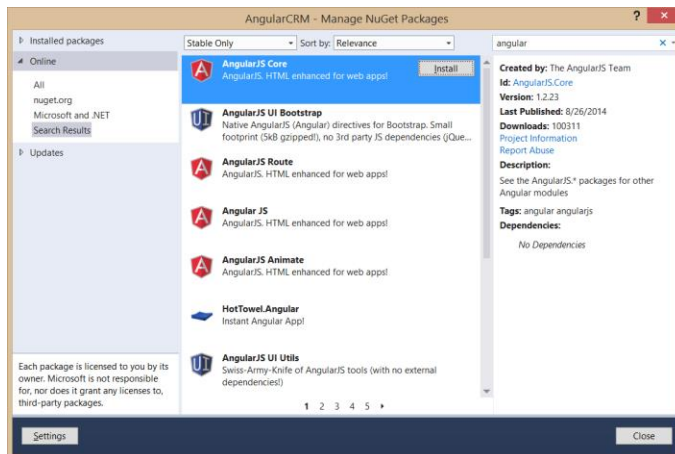
5. Examine the default project setup for the new SharePoint-Hosted app. As you can see, it is like a traditional SharePoint solution-based project because you have a **Features** and **Packages** node. Note that there are project folders named **Content**, **Images** & **Pages** are actually SharePoint Project Modules that will provision their contents to the respective folders in the app web when the app is installed.



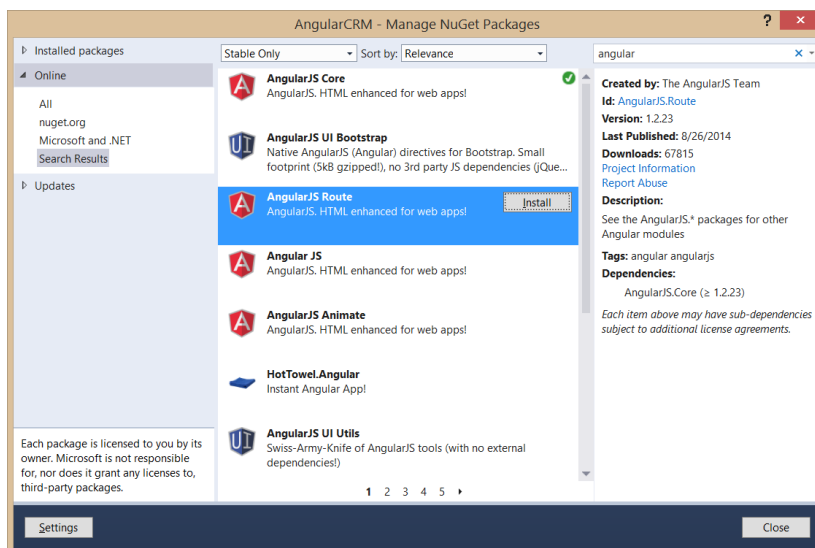
6. Add the NuGet Packages for Bootstrap and AngularJS.
  - a) Right-click on the **AngularCRM** project in the solution Explorer and select **Manage NuGet Packages**.
  - b) Using the **Manage NuGet Packages** dialog, install the NuGet package for **Bootstrap**.



c) Next, install the NuGet package for **AngularJS Core**.



d) Finally, install the NuGet package for AngularJS Route.

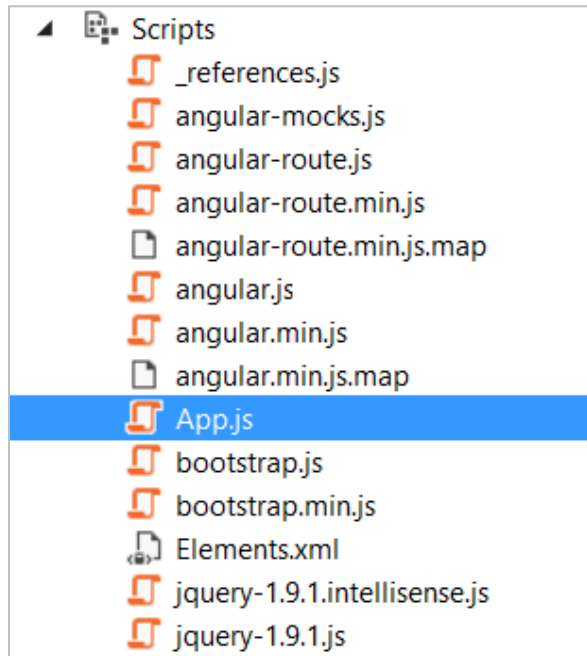


e) Close the **Manage NuGet Packages** dialog.

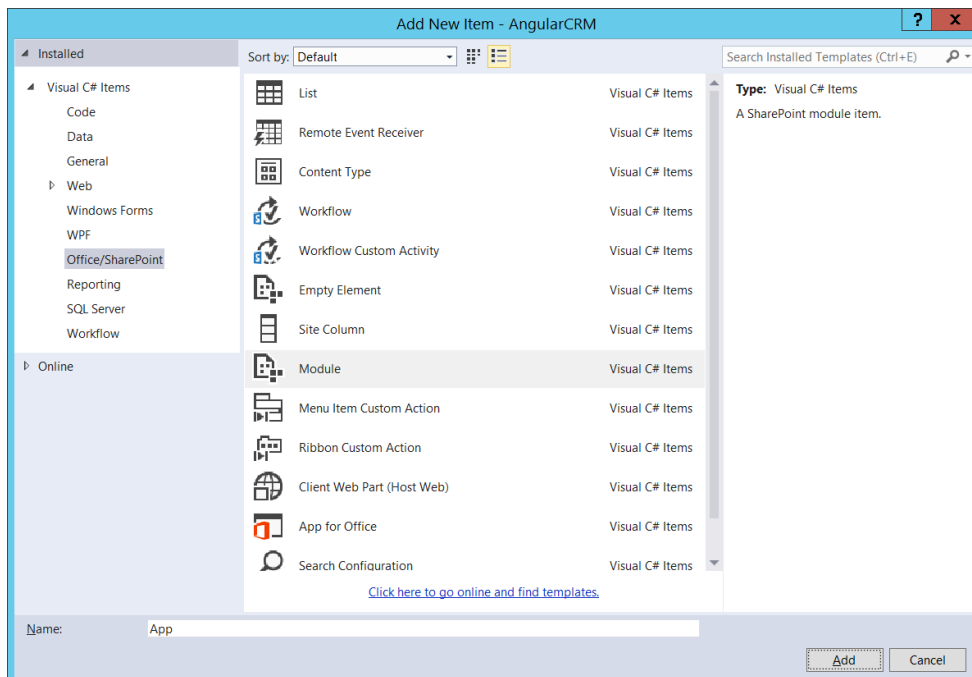
7. Your app will not be using anything from the **Pages** folder. Therefore, you should delete the **Pages** folder from your project by right-clicking on it in the Solution Explorer and selecting the **Delete** command.



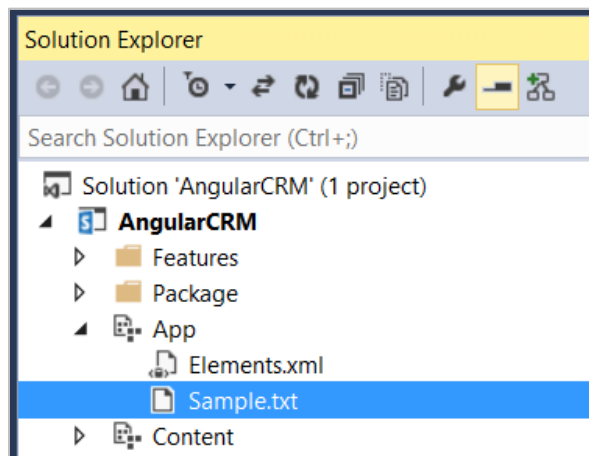
8. Delete the **App.js** file from the **Scripts** folder.
- Open the Scripts folder and inspect what's inside.
  - Note that there is a **App.js** file that you will not be using in this project. Delete the **App.js** file.



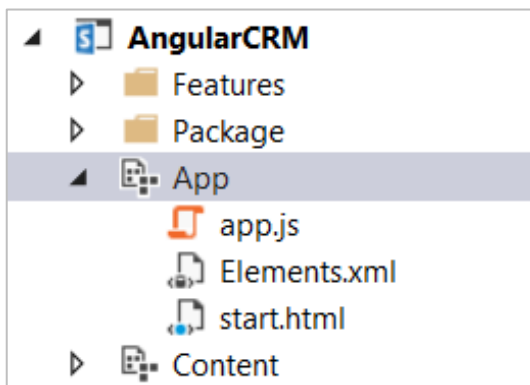
9. Add a new SharePoint module named **App**.
- In the solution Explorer, right-click on the top-level node of the **AngularCRM** project and click **Add > New Item**.
  - In the **Add New Item** dialog, select the **Module** template and enter the name **App** as shown in the following screenshot. Click the **Add** button when you are done.



- Once the **App** module has been created in the project, you should notice that it already contains a file named **Elements.xml** and a second file **Sample.txt**. Delete the file named **Sample.txt**.



10. Add a new HTML file named **start.html** into the folder for the **App** module.
11. Add a new JavaScript file named **App.js** into the folder for the **App** module.



12. Open **start.html** in an editor windows and modify the head section to match the following code listing.

```
<head>
  <meta charset="utf-8" />
  <meta http-equiv="X-UA-Compatible" content="IE=10" />
  <title>Angular CRM</title>
</head>
```

13. Inside the **head** section, add two links to the CSS files in the Content folder named **bootstrap.css** and **App.css**.

```
<head>
  <meta charset="utf-8" />
  <meta http-equiv="X-UA-Compatible" content="IE=10" />
  <title>Angular CRM</title>

  <link href="../../Content/bootstrap.css" rel="stylesheet" />
  <link href="../../Content/App.css" rel="stylesheet" />
</head>
```

14. Add in **script** links to the JavaScript files in the scripts folder for jQuery, as well as **bootstrap.js**, **angular.js** and **angular.route.js**. Also, add a link to the **App.js** file which is located in the **App** folder.

```
<head>
  <meta charset="utf-8" />
  <meta http-equiv="X-UA-Compatible" content="IE=10" />
  <title>Angular CRM</title>

  <link href="../../Content/bootstrap.css" rel="stylesheet" />
  <link href="../../Content/App.css" rel="stylesheet" />

  <script src="../../Scripts/jquery-1.9.1.js"></script>
```

```

<script src="../../Scripts/bootstrap.js"></script>
<script src="../../Scripts/angular.js"></script>
<script src="../../Scripts/angular-route.js"></script>
<script src="App.js"></script>
</head>

```

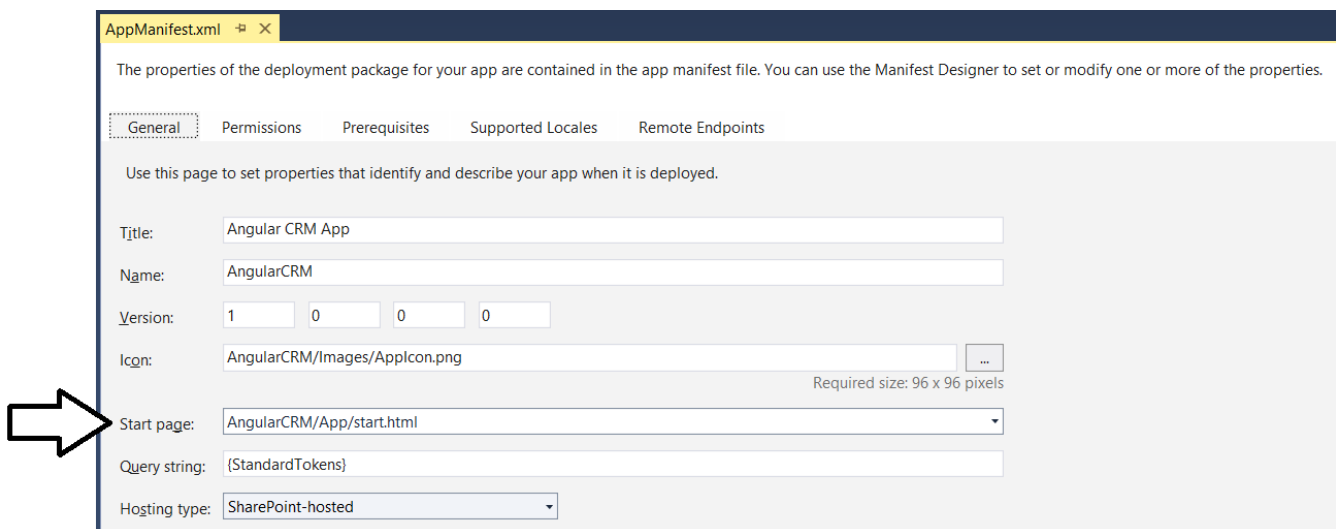
15. Update the **body** element by adding the following HTML layout which uses predefined Bootstrap styles such as **container**, **container-fluid** and **navbar**. If you'd rather note type all this code by hand, you can copy it from the **start.html.body.txt** files in the **StarterFiles** folder for this lab.

```

<body>
  <div class="container">
    <div class="navbar navbar-default" role="navigation">
      <div class="container-fluid">
        <div class="navbar-header">
          <a class="navbar-brand" href="#">Angular CRM</a>
        </div>
        <div class="navbar-collapse collapse">
          <ul class="nav navbar-nav">
            <li><a href="#">Home</a></li>
            <li><a href="#">Add Customer</a></li>
            <li><a href="#">About</a></li>
          </ul>
          <ul class="nav navbar-nav navbar-right">
            <li><a id="lnkHostWeb">Back to Host Web</a></li>
          </ul>
        </div>
      </div>
    </div>
    <div class="container">
      <div id="content-box" />
    </div>
  </body>

```

16. Save and close **start.html**.
17. Open up **AppManifest.xml** in the designer and update the app's **Start page** setting to point to **start.html**. Also, update the **Title** to something more readable such as **Angular CRM App**.



The properties of the deployment package for your app are contained in the app manifest file. You can use the Manifest Designer to set or modify one or more of the properties.

General Permissions Prerequisites Supported Locales Remote Endpoints

Use this page to set properties that identify and describe your app when it is deployed.

Title: Angular CRM App

Name: AngularCRM

Version: 1 0 0 0

Icon: AngularCRM/Images/AppIcon.png Required size: 96 x 96 pixels

Start page: AngularCRM/App/start.html

Query string: {StandardTokens}

Hosting type: SharePoint-hosted

18. Save and close **AppManifest.xml**.
19. Open **App.js** in an editor window and update its contents to match the following code listing.

```

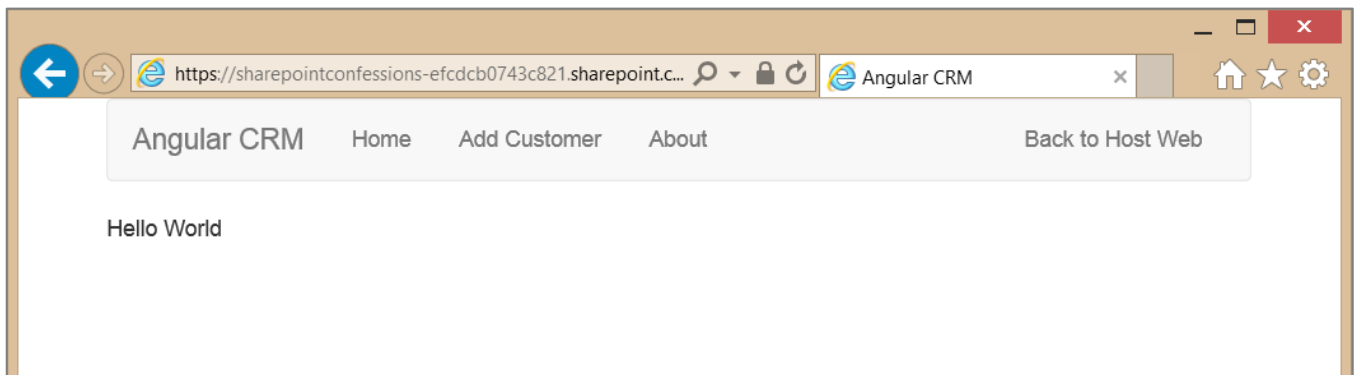
'use strict';

angular.element(document).ready( function () {
  $("#content-box").text("Hello world");
});

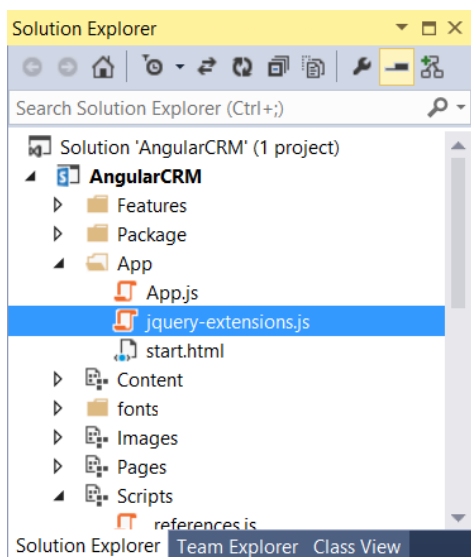
```

```
});
```

20. It's now time to test the app. Press **{F5}** to begin a new debugging sessions. Once the app has been installed, Visual Studio will start up Internet Explorer and redirect you to the app's start page. You should see the app's navbar and the "Hello World" test message as shown in the following screenshot.



21. Close the browser window to terminate the debugging session and return to Visual Studio.
22. Your app will need to read query string parameters. Therefore, you will create a simple jQuery extension to add a few helper functions. Begin by creating a new JavaScript file into the App folder named **jquery-extensions.js**.



23. Add the following code to **jquery-extensions.js** to extend the jQuery library with two helper methods named **getQueryStringValues** and **getQueryStringValue**. If you'd rather not type all this code by hand, you can copy it from the **jquery-extensions.js.txt** file in the **StarterFiles** folder for this lab.

```
$.extend({  
  getQueryStringValues: function () {  
    var vars = [], hash;  
    var hashes = window.location.href.slice(window.location.href.indexOf('?') + 1).split('&');  
    for (var i = 0; i < hashes.length; i++) {  
      hash = hashes[i].split('=');  
      vars.push(hash[0]);  
      vars[hash[0]] = hash[1];  
    }  
    return vars;  
  },  
  getQueryStringValue: function (name) {  
    return decodeURIComponent($.getQueryStringValues()[name]);  
  }  
});
```

```
});
```

24. Open **start.html** and add a new script link for named **jquery-extensions.js** right after the script link for the jQuery library.

```
<head>
  <meta charset="utf-8" />
  <meta http-equiv="X-UA-Compatible" content="IE=10" />
  <title>Angular CRM</title>
  <link href="../Content/bootstrap.css" rel="stylesheet" />
  <link href="../Content/App.css" rel="stylesheet" />
  <script src="../Scripts/jquery-1.9.1.js"></script>

  <script src="jquery-extensions.js"></script>

  <script src="../Scripts/bootstrap.js"></script>
  <script src="../Scripts/angular.js"></script>
  <script src="../Scripts/angular-route.js"></script>
  <script src="App.js"></script>
</head>
```

25. Save and close **start.html**.
26. Open **App.js** and update its contents to match the following code listing.

```
'use strict';

angular.element(document).ready( function () {
  var hostWeb = $.getQueryStringValue("SPHostUrl");
  $("#lnkHostWeb").attr("href", hostWeb);
});
```

27. It's time again to test the app. Press {F5} to begin a new debugging sessions. Once the app has been installed, Visual Studio will start up Internet Explorer and redirect you to the app's start page. At this point, you should be able to click the **Back to Host Web** link on the right-hand side of the navbar and successfully navigate back to the host web which you are using for your testing.
28. Close the Internet explorer to terminate the debugging session and return to Visual studio.
29. The final steps in this exercise will involve adding the **ng-app** directive to the **body** element of **start.html** and adding JavaScript code to properly initialize the app while the Angular framework is loading. Begin by opening **start.html** in an editor window.
30. Locate the opening tag of the body element and add the **ng-app** directive with an app name of **AngularCRM**.

```
<body ng-app="AngularCRM">
```

31. Save your changes to **start.html**.
32. Open **App.js**. Remove all the existing code and replace it with the code shown in the following code listing.

```
'use strict';

var crmApp = angular.module("AngularCRM", []);

crmApp.config(function () {
  var hostWeb = $.getQueryStringValue("SPHostUrl");
  $("#lnkHostWeb").attr("href", hostWeb);
});
```

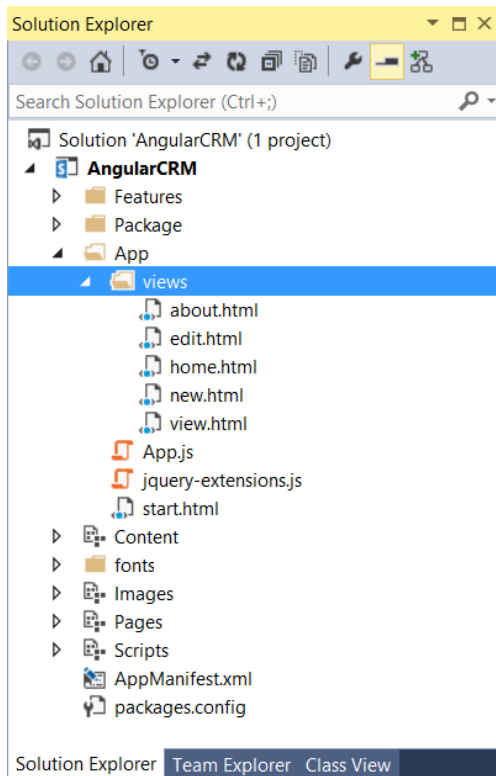
33. Test the app by pressing {F5} to begin a new debugging sessions. Once the app has been installed, Visual Studio will start up Internet Explorer and redirect you to the app's start page. Test to ensure you can still click the **Back to Host Web** link and successfully navigate back to the host web. It should work just as it did before, it's just now you are using a more angular-specific way of initializing the app.
34. Close the Internet explorer to terminate the debugging session and return to Visual studio.

## Exercise 6: Working with Views, Controllers and Routing

In this lab, you will continue working with the AngularCRM project you created in the previous lab exercise. You will extend this project by adding several new views and controllers and configuring the app's routing scheme.

1. Open the **AngularCRM** project in Visual Studio if it is not already open.
2. Create a new folder named **views** inside the **App** folder.

3. Create five new HTML files inside the **views** folder named **about.html**, **edit.html**, **home.html**, **new.html** and **view.html**.



4. Update the contents of **home.html** to match the following listing and save your changes.

```
<h3>Customer List</h3>
```

5. Update the contents of **new.html** to match the following listing and save your changes.

```
<h3>New Customer</h3>
```

6. Update the contents of **view.html** to match the following listing and save your changes.

```
<h3>View Customer</h3>
```

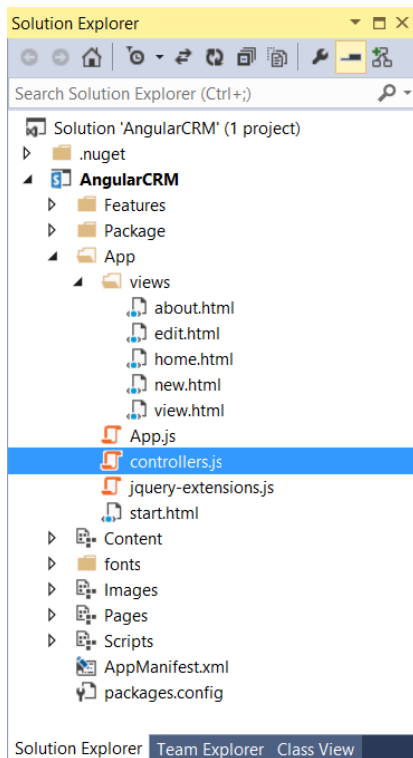
7. Update the contents of **edit.html** to match the following listing and save your changes.

```
<h3>Edit Customer</h3>
```

Update the contents of **about.html** to match the following listing and save your changes.

```
<h3 ng-bind="title"></h3>
<p ng-bind="description"></p>
```

8. Create a new JavaScript file named **controllers.js** in the App folder.



9. Add the following code into controllers.js to provide a controller starting point for each of the views. If you'd rather not type all this code by hand, you can copy it from the **controllers\_starter.js.txt** file in the **StarterFiles** folder for this lab.

```
'use strict';

var app = angular.module('AngularCRM');

app.controller('homeController',
  function ($scope) {
  }
);

app.controller('newController',
  function ($scope) {
  }
);

app.controller('viewController',
  function ($scope) {
  }
);

app.controller('editController',
  function ($scope) {
  }
);

app.controller('aboutController',
  function ($scope) {
  }
);
```

10. Implement the **aboutController** controller function to create a **title** property and a **description** property on the **\$scope** variable and to initialize their values using string literals.

```
app.controller('aboutController',
  function ($scope) {
    $scope.title = "About the Angular CRM App"
    $scope.description = "The Angular CRM App is a demo app which uses Bootstrap and AngularJS"
  }
);
```

```
);
```

11. Update the head section in **start.html** to include a script link for **controllers.js** just after the script link to App.js and then save your changes.

```
<script src="App.js"></script>
<script src="controllers.js"></script>
```

12. Now that you have created the views and controllers, it's time to configure the app's routing scheme. Begin by opening App.js in an editor window.

13. Locate the line of code which calls `angular.module` and update it to include a dependency on `ngRoute` module.

```
var crmApp = angular.module("AngularCRM", ['ngRoute']);
```

14. Locate the line of code which calls `angular.config` and modify the controller function parameter list to accept a parameter named `$routeProvider`.

```
crmApp.config(function ($routeProvider) {
```

15. At this point, the code in App.js should match the following code listing.

```
'use strict';

var crmApp = angular.module("AngularCRM", ['ngRoute']);

crmApp.config(function ($routeProvider) {

    var hostWeb = $.getQueryStringValue("SPHostUrl");
    $("#lnkHostWeb").attr("href", hostWeb);

});
```

16. Add the following code to configure the app's routing map just after the code which updates the href attribute of the anchor element with an id of **lnkHostWeb**. If you'd rather not type all this code by hand, you can copy it from the **routing\_map.js.txt** file in the **StarterFiles** folder for this lab.

```
$routeProvider.when("/", {
    templateUrl: 'views/home.html',
    controller: "homeController"
}).when("/view/:id", {
    templateUrl: 'views/view.html',
    controller: "viewController"
}).when("/edit/:id", {
    templateUrl: 'views/edit.html',
    controller: "editController"
}).when("/new", {
    templateUrl: 'views/new.html',
    controller: "newController"
}).when("/about", {
    templateUrl: 'views/about.html',
    controller: "aboutController"
}).otherwise({
    redirectTo: "/"
});
```

When you are done, the code you have written in App.js should match the following code listing.

```
'use strict';

var crmApp = angular.module("AngularCRM", ['ngRoute']);

crmApp.config(function ($routeProvider) {

    var hostWeb = $.getQueryStringValue("SPHostUrl");
    $("#lnkHostWeb").attr("href", hostWeb);

    // config route map
    $routeProvider.when("/", {
        templateUrl: 'views/home.html',
        controller: "homeController"
    }).when("/view/:id", {
```



```

        templateUrl: 'views/view.html',
        controller: "viewController"
    }).when("/edit/:id", {
        templateUrl: 'views/edit.html',
        controller: "editController"
    }).when("/new", {
        templateUrl: 'views/new.html',
        controller: "newController"
    }).when("/about", {
        templateUrl: 'views/about.html',
        controller: "aboutController"
    }).otherwise({
        redirectTo: "/"
    });
});

```

17. The last remaining task before testing is to update the navigation links on **start.html**. Begin by opening **start.html** and locating the elements with the links titled **Home**, **Add Customer** and **About**.
18. Update the href attribute for the Add Customer link to **#/new** and update the href attribute for the About link to **#/about**.

```

<div class="navbar-collapse collapse">
  <ul class="nav navbar-nav">
    <li><a href="#">Home</a></li>
    <li><a href="#/new">Add Customer</a></li>
    <li><a href="#/about">About</a></li>
  </ul>
</div>

```

19. Down in the body section of start.html, locate the div element with the id of content-box and add the ng-view directive.

```

<div class="container">
  <div id="content-box" ng-view ></div>
</div>

```

20. Save your changes to start.html.
21. Test the routing scheme of the app by pressing {F5} to begin a new debugging sessions. Once the app has been installed, Visual Studio will start up Internet Explorer and redirect you to the app's start page. At this point, you should be able to click on the navbar links titled Home, Add Customer and About to navigate between these three views.
22. Click on the About link to navigate to the app's About page. You should be able to verify that the about view is properly displaying the values of the title property and the description property that were written to the \$scope variable by aboutController.

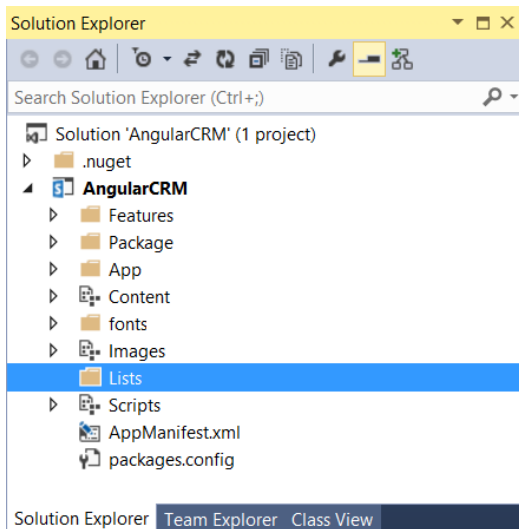


You have now successfully set up the routing scheme for the app. Close the Internet explorer to terminate the debugging session and return to Visual studio.

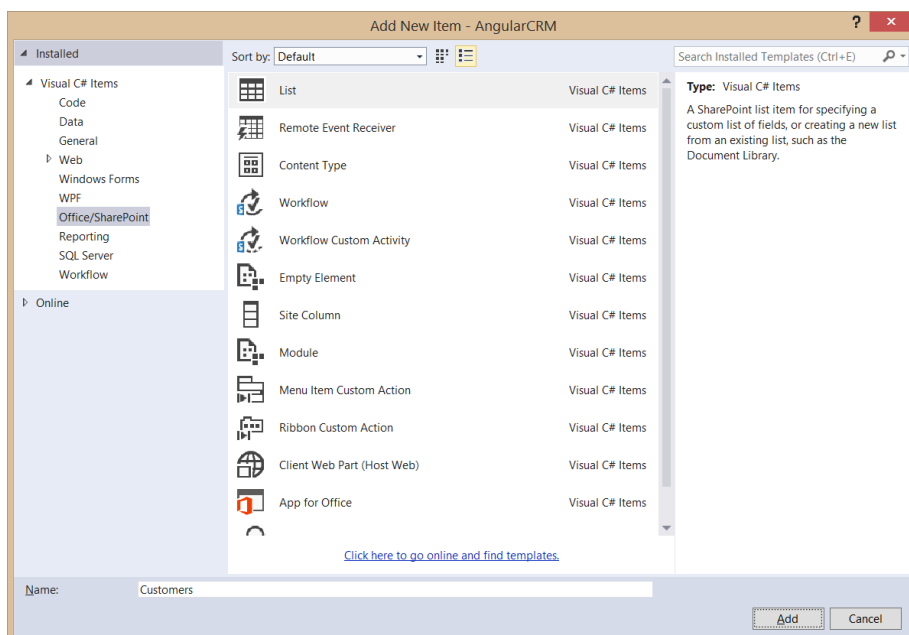
## Exercise 7: Extending the AngularCRM Project with a Custom Service

In this exercise you will extend the AngularCRM app by adding a SharePoint list and then creating a custom angular service to read and write items to and from the list.

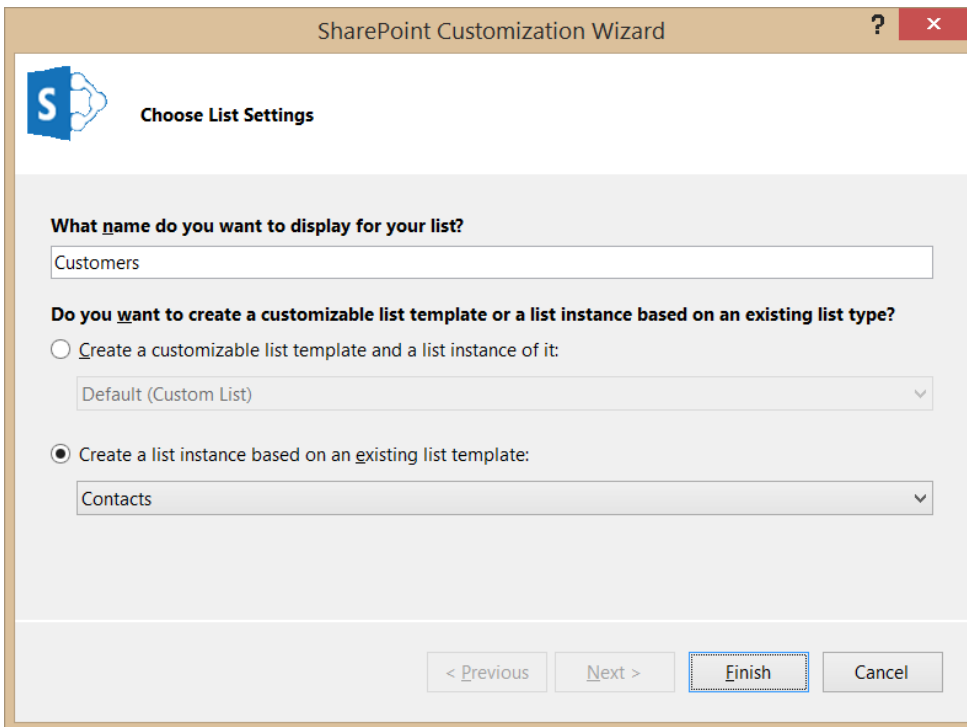
1. Open the AngularCRM project in Visual Studio if it is not already open.
2. Create a new top-level folder named Lists at the root of the AngularCRM project.



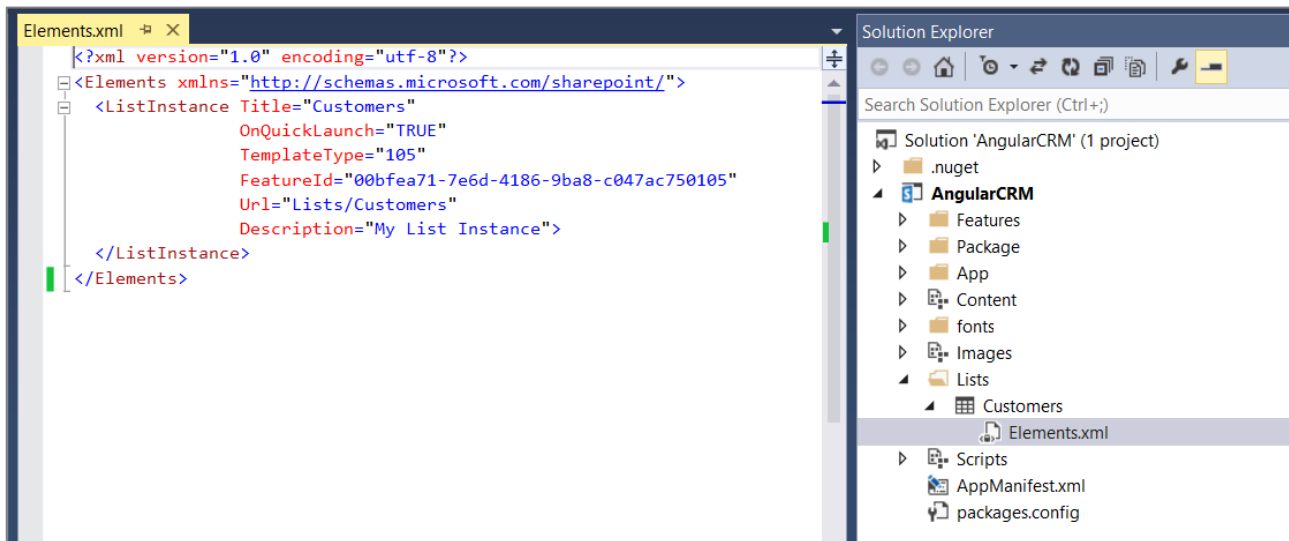
3. Right-click on the Lists folder and select Add > New Item.
4. In the Add New Item dialog, click the Office/SharePoint group on the left and then select the List template. Enter a name of Customers and click Add.



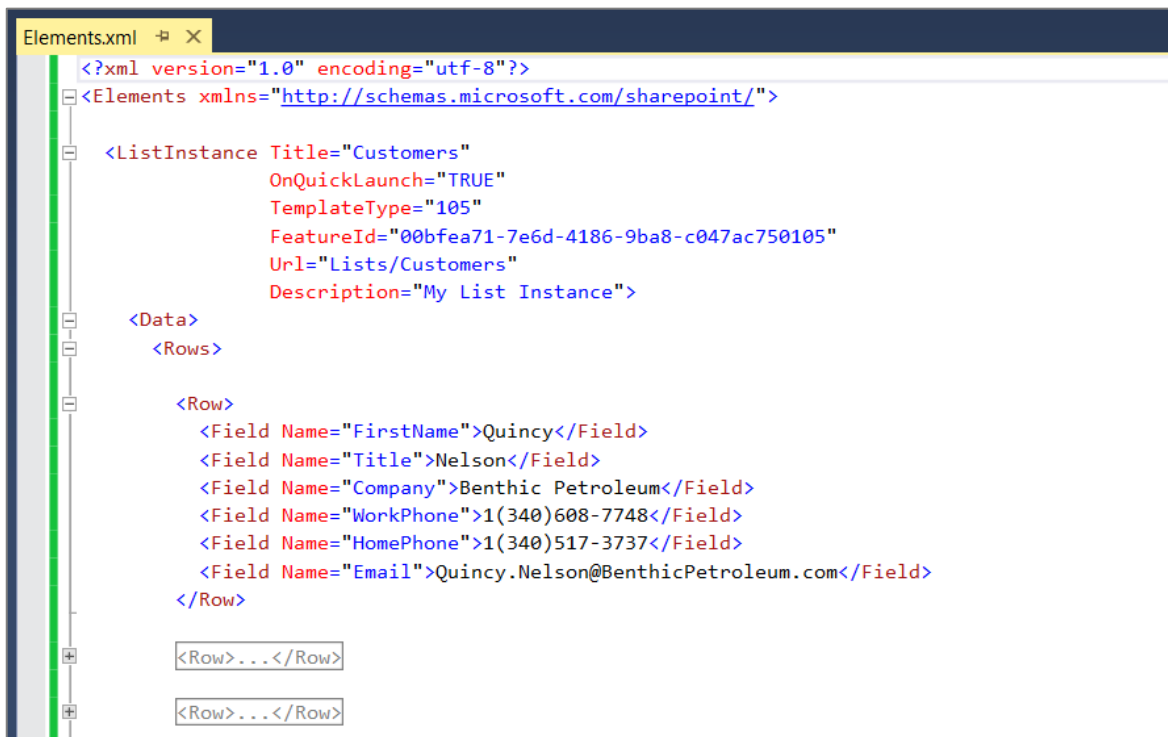
5. In the SharePoint Customization Wizard dialog, enter a list display name of Customers. Select the option Create a list instance based on an existing list template and set the list template type to Contacts.



6. Click the Finish button in the SharePoint Customization Wizard to create the new project item for the list. Inside the Lists folder, you should be able to see a Customers folder which contains an element manifest named elements.xml.



7. In this step you will modify the elements.xml so that the Customers list will be created with a pre-populated set of customer items.
8. Using Windows Explorer, look inside the Starter Files folder for this lab and locate the file named Customers\_Elements.xml.txt.
  - a) Open Customers\_Elements.xml.txt in NOTEPAD and copy all its contents into the Windows clipboard.
  - b) Return to Visual Studio and make sure that the elements.xml file for the Customers list is open in an editor window.
  - c) Delete all the existing content from elements.xml.
  - d) Paste in the contents of the clipboard.
  - e) Save your changes to elements.xml.
9. Look at the XML content inside elements.xml and examine how it uses a Data element with an inner Rows element to pre-populate the Customers list with a set of sample customer items to assist in your testing and debugging.



```
<?xml version="1.0" encoding="utf-8"?>
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">

  <ListInstance Title="Customers"
    OnQuickLaunch="TRUE"
    TemplateType="105"
    FeatureId="00bfea71-7e6d-4186-9ba8-c047ac750105"
    Url="Lists/Customers"
    Description="My List Instance">

    <Data>
      <Rows>

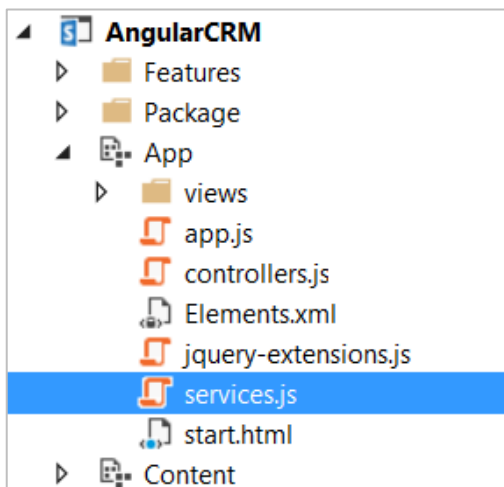
        <Row>
          <Field Name="FirstName">Quincy</Field>
          <Field Name="Title">Nelson</Field>
          <Field Name="Company">Benthic Petroleum</Field>
          <Field Name="WorkPhone">1(340)608-7748</Field>
          <Field Name="HomePhone">1(340)517-3737</Field>
          <Field Name="Email">Quincy.Nelson@BenthicPetroleum.com</Field>
        </Row>

        <Row>...</Row>

        <Row>...</Row>

      </Rows>
    </Data>
  </ListInstance>
</Elements>
```

10. Save and close elements.xml.
11. In the Apps folder, create a new JavaScript file named services.js.



12. Open start.html and add a script link for services.js just after the script link to controllers.js.

```
<script src="App.js"></script>
<script src="controllers.js"></script>
<script src="services.js"></script>
```

13. Save and close start.html.
14. Open **services.js** in an editor window.
15. Add the following code into **services.js** to provide a starting point for your service implementation.

```
'use strict';

var app = angular.module('AngularCRM');

app.factory("wingtipCrmService",
```

```
function ($http) {
    // create service object
    var service = {};

    // TODO: add behavior to service object

    // return service object to angular framework
    return service;
};
```

16. Add the following code just below the line with the comment **TODO: add behavior to service object**. This code will initialize the service by retrieving and caching a form digest value in a variable named **requestDigest**.

```
// retrieve and cache SharePoint form digest value
var requestDigest;

$http({
    method: 'POST',
    url: "../_api/contextinfo",
    headers: { "Accept": "application/json; odata=verbose" }
}).success(function (data) {
    requestDigest = data.d.GetContextWebInformation.FormDigestValue
});
```

17. Just below the code you added in the previous step, add the following code to add a function to the service named **getCustomers**.

```
service.getCustomers = function () {
    var restQueryUrl = "../_api/web/lists/getByTitle('Customers')/items/" +
        "?$select=ID,Title,FirstName,WorkPhone,HomePhone,Email";
    return $http({
        method: 'GET',
        url: restQueryUrl,
        headers: { "Accept": "application/json; odata=verbose" }
    })
}
```

18. Open **controllers.js** in an editor window.

19. Locate the code which defines the controller function for **homeController**. In the function parameter list, add a second parameter named **wingtipCrmService** in addition to the **\$scope** parameter that is already defined.

```
app.controller('homeController',
    function ($scope, wingtipCrmService) {
    }
);
```

20. Implement the **homeController** function as shown in the following code listing.

```
app.controller('homeController',
    function ($scope, wingtipCrmService) {
        wingtipCrmService.getCustomers().success(function (data) {
            $scope.customers = data.d.results;
        });
    }
);
```

21. Open **Home.html** in an editor window. Copy the following HTML layout and paste it into **Home.html** to replace the existing content. If you'd rather not type all this code by hand, you can copy it from the **home.html.txt** file in the **StarterFiles** folder for this lab.

```
<h3>Customer List</h3>

<table class="table table-striped table-hover table-responsive ">
    <thead>
        <tr>
            <td>ID</td>
            <td>First Name</td>
            <td>Last Name</td>
            <td>Work Phone</td>
            <td>Home Phone</td>
            <td>Email Address</td>
            <td>&nbsp;</td>
            <td>&nbsp;</td>
            <td>&nbsp;</td>
```

```

</tr>
</thead>
<tbody ng-repeat="customer in customers">
  <tr>
    <td>{{customer.Id}}</td>
    <td>{{customer.FirstName}}</td>
    <td>{{customer.Title}}</td>
    <td>{{customer.WorkPhone}}</td>
    <td>{{customer.HomePhone}}</td>
    <td>{{customer.Email}}</td>
    <td><a href="#/view/{{customer.Id}}" class="btn-mini">View</a></td>
    <td><a href="#/edit/{{customer.Id}}" class="btn-mini">Edit</a></td>
    <td><a href="#" data-ng-click="deleteCustomer(customer.Id)" class="btn-mini">Delete</a></td>
  </tr>
</tbody>
</table>

```

22. Test your work by pressing {F5} to begin a debugging session. The app should initialize the start page using the view defined in Home.html which should display a table of customers as shown in the following screenshot.

Angular CRM

Home

Add Customer

About

Back to Host Web

Customer List

ID	First Name	Last Name	Work Phone	Home Phone	Email Address			
1	Quincy	Nelson	1(340)608-7748	1(340)517-3737	Quincy.Nelson@BenthicPetroleum.com	<a href="#">View</a>	<a href="#">Edit</a>	<a href="#">Delete</a>
2	Jude	Mason	1(203)408-0466	1(203)411-0071	Jude.Mason@CyberdyneSystems.com	<a href="#">View</a>	<a href="#">Edit</a>	<a href="#">Delete</a>
3	Sid	Stout	1(518)258-6571	1(518)376-8576	Sid.Stout@Roxxon.com	<a href="#">View</a>	<a href="#">Edit</a>	<a href="#">Delete</a>
4	Gilberto	Gillespie	1(270)510-1720	1(270)755-7810	Gilberto.Gillespie@ShinraElectricPowerCompany.com	<a href="#">View</a>	<a href="#">Edit</a>	<a href="#">Delete</a>
5	Diane	Strickland	1(407)413-4851	1(407)523-5411	Diane.Strickland@Izon.com	<a href="#">View</a>	<a href="#">Edit</a>	<a href="#">Delete</a>
6	Jacqueline	Zimmerman	1(844)234-0550	1(844)764-3522	Jacqueline.Zimmerman@ZorgIndustries.com	<a href="#">View</a>	<a href="#">Edit</a>	<a href="#">Delete</a>
7	Naomi	Schroeder	1(204)355-6648	1(204)356-2831	Naomi.Schroeder@ComTron.com	<a href="#">View</a>	<a href="#">Edit</a>	<a href="#">Delete</a>
8	Lynne	Stephens	1(407)787-7308	1(407)732-1700	Lynne.Stephens@TradeFederation.com	<a href="#">View</a>	<a href="#">Edit</a>	<a href="#">Delete</a>
9	Luther	Sullivan	1(323)755-3404	1(323)684-7814	Luther.Sullivan@Metacortex.com	<a href="#">View</a>	<a href="#">Edit</a>	<a href="#">Delete</a>
10	Rose	Parsons	1(802)357-5583	1(802)727-0246	Rose.Parsons@HansoFoundation.com	<a href="#">View</a>	<a href="#">Edit</a>	<a href="#">Delete</a>

23. Close Internet Explorer to terminate your debugging session and return to Visual Studio.
24. Now it is time to implement the remaining functionality needed for the service. Begin by opening **services.js** and positioning your cursor just below the **getCustomers** function you added earlier.
25. Add the following function implementation to the service for the **getCustomer** function.

```

service.getCustomer = function (id) {
  var restQueryUrl = "../_api/web/lists/getByTitle('Customers')/items(" + id + ")/" +
    "?$select=ID,Title,FirstName,WorkPhone,HomePhone,Email";
  return $http({
    method: 'GET',
    url: restQueryUrl,
    headers: { "Accept": "application/json; odata=verbose" }
  })
}

```

26. Add the following function implementation to the service for the **deleteCustomer** function.

```

service.deleteCustomer = function (id) {
  var restQueryUrl = "../_api/web/lists/getByTitle('Customers')/items(" + id + ")/";
  return $http({
    method: 'DELETE',
    url: restQueryUrl,
    headers: {
      "Accept": "application/json; odata=verbose",
      "X-RequestDigest": requestDigest,
      "If-Match": "*"
    }
  })
}

```

```

    }
  });
}

```

27. Add the following function implementation to the service for the `addCustomer` function. If you'd rather not type all this code by hand, you can copy it from the `addCustomer.js.txt` file in the **StarterFiles** folder for this lab.

```

service.addCustomer = function (FirstName, LastName, WorkPhone, HomePhone, Email) {
  var restQueryUrl = "../_api/web/lists/getByTitle('Customers')/items";

  var customerData = {
    __metadata: { "type": "SP.Data.CustomersListItem" },
    Title: LastName,
    FirstName: FirstName,
    WorkPhone: WorkPhone,
    HomePhone: HomePhone,
    Email: Email
  };

  var requestBody = JSON.stringify(customerData);

  return $http({
    method: 'POST',
    url: restQueryUrl,
    contentType: "application/json;odata=verbose",
    data: requestBody,
    headers: {
      "Accept": "application/json; odata=verbose",
      "X-RequestDigest": requestDigest,
      "content-type": "application/json;odata=verbose"
    }
  });
}

```

28. Add the following function implementation to the service for the `updateCustomer` function. If you'd rather not type all this code by hand, you can copy it from the `updateCustomer.js.txt` file in the **StarterFiles** folder for this lab.

```

service.updateCustomer = function (id, FirstName, LastName, WorkPhone, HomePhone, Email, etag) {
  var restQueryUrl = "../_api/web/lists/getByTitle('Customers')/items(" + id + ")";

  var customerData = {
    __metadata: { "type": "SP.Data.CustomersListItem" },
    Title: LastName,
    FirstName: FirstName,
    WorkPhone: WorkPhone,
    HomePhone: HomePhone,
    Email: Email
  };

  var requestBody = JSON.stringify(customerData);

  return $http({
    method: 'POST',
    url: restQueryUrl,
    contentType: "application/json;odata=verbose",
    data: requestBody,
    headers: {
      "Accept": "application/json; odata=verbose",
      "X-RequestDigest": requestDigest,
      "content-type": "application/json;odata=verbose",
      "If-Match": etag,
      "X-HTTP-METHOD": "PATCH"
    }
  });
}

```

29. Save your changes to `services.js`.
30. Open `controllers.js` in an editor window.
31. Replace the code with creates `homeController` to add the `deleteCustomer` function to the `$scope` variable. You can either type the following code or copy and paste it from the `homeController.js.txt` file in the **Starter Files** folder.

```

app.controller('homeController',

```

```

function ($scope, wingtipCrmService) {
    wingtipCrmService.getCustomers().success(function (data) {
        $scope.customers = data.d.results;
        // add behavior function for view to call
        $scope.deleteCustomer = function (id) {
            wingtipCrmService.deleteCustomer(id).success(function (data) {
                $scope.$apply();
            });
        };
    });
}
};

```

32. Replace the code which creates the **newController** with the following code. If you'd rather not type all this code by hand, you can copy it from the **newController.js.txt** file in the **StarterFiles** folder for this lab.

```

app.controller('newController',
    function ($scope, $location, wingtipCrmService) {

        $scope.customer = {};
        $scope.customer.FirstName = "";
        $scope.customer.Title = "";
        $scope.customer.WorkPhone = "";
        $scope.customer.HomePhone = "";
        $scope.customer.Email = "";

        $scope.addCustomer = function () {
            var firstName = $scope.customer.FirstName;
            var lastName = $scope.customer.Title;
            var workPhone = $scope.customer.WorkPhone;
            var homePhone = $scope.customer.HomePhone;
            var email = $scope.customer.Email;
            wingtipCrmService.addCustomer(firstName, lastName, workPhone, homePhone, email)
                .success(function (data) {
                    $location.path("/");
                });
        };
    }
);

```

33. Replace the code which creates the **viewController** with the following code.

```

app.controller('viewController',
    function ($scope, $routeParams, wingtipCrmService) {
        var id = $routeParams.id;
        wingtipCrmService.getCustomer(id).success(function (data) {
            $scope.customer = data.d;
        });
    }
);

```

34. Replace the code which creates the **editController** with the following code. If you'd rather not type all this code by hand, you can copy it from the **editController.js.txt** file in the **StarterFiles** folder for this lab.

```

app.controller('editController',
    function ($scope, $routeParams, $location, wingtipCrmService) {
        var id = $routeParams.id;
        wingtipCrmService.getCustomer(id).success(function (data) {
            $scope.customer = data.d;

            $scope.updateCustomer = function () {
                var firstName = $scope.customer.FirstName;
                var lastName = $scope.customer.Title;
                var workPhone = $scope.customer.WorkPhone;
                var homePhone = $scope.customer.HomePhone;
                var email = $scope.customer.Email;
                var etag = $scope.customer.__metadata.etag;
                wingtipCrmService.updateCustomer(id, firstName, lastName, workPhone, homePhone, email, etag)
                    .success(function (data) {
                        $location.path("/");
                    });
            };
        });
    }
);

```



```
}  
);
```

35. Save your changes to **controllers.js**.
36. Open the view file in the **views** folder named **new.html** and replace the contents with the following HTML layout. If you'd rather not type all this code by hand, you can copy it from the **new.html.txt** file in the **StarterFiles** folder for this lab.

```
<h3>New Customer</h3>  
  
<div class="form-horizontal">  
  <fieldset>  
    <div class="form-group">  
      <label for="txtFirstName" class="col-lg-2 control-label">First Name:</label>  
      <div class="col-lg-6">  
        <input id="txtFirstName" type="text" class="form-control" ng-model="customer.FirstName">  
      </div>  
    </div>  
    <div class="form-group">  
      <label for="txtLastName" class="col-lg-2 control-label">Last Name:</label>  
      <div class="col-lg-6">  
        <input id="txtLastName" type="text" class="form-control" ng-model="customer.Title">  
      </div>  
    </div>  
    <div class="form-group">  
      <label for="txtWorkPhone" class="col-lg-2 control-label">Work Phone:</label>  
      <div class="col-lg-6">  
        <input id="txtWorkPhone" type="text" class="form-control" ng-model="customer.WorkPhone">  
      </div>  
    </div>  
    <div class="form-group">  
      <label for="txtHomePhone" class="col-lg-2 control-label">Home Phone:</label>  
      <div class="col-lg-6">  
        <input id="txtHomePhone" type="text" class="form-control" ng-model="customer.HomePhone">  
      </div>  
    </div>  
    <div class="form-group">  
      <label for="txtEmailAddress" class="col-lg-2 control-label">Email Addresss:</label>  
      <div class="col-lg-6">  
        <input id="txtEmailAddress" type="text" class="form-control" ng-model="customer.Email">  
      </div>  
    </div>  
    <div class="form-group">  
      <div class="col-lg-offset-2">  
        <input id="cmdSave" type="button" class="button" value="Save" data-ng-  
click="addCustomer()" />  
      </div>  
    </div>  
  </fieldset>  
</div>  
  
<hr />  
  
<a href="#">Return to customers list</a>
```

37. Save and close **new.html**.
38. Open the view file in the **views** folder named **view.html** and replace the contents with the following HTML layout. If you'd rather not type all this code by hand, you can copy it from the **view.html.txt** file in the **StarterFiles** folder for this lab.

```
<h3>View Customer</h3>  
  
<div class="form-horizontal">  
  <fieldset>  
    <div class="form-group">  
      <label for="txtID" class="col-lg-2 control-label">ID:</label>  
      <div class="col-lg-6">  
        <input id="txtID" type="text" readonly class="form-control" ng-model="customer.ID">  
      </div>  
    </div>  
    <div class="form-group">  
      <label for="txtFirstName" class="col-lg-2 control-label">First Name:</label>  
      <div class="col-lg-6">
```

```

        <input id="txtFirstName" type="text" readonly class="form-control"
            ng-model="customer.FirstName">
    </div>
</div>
<div class="form-group">
    <label for="txtLastName" class="col-lg-2 control-label">Last Name:</label>
    <div class="col-lg-6">
        <input id="txtLastName" type="text" readonly class="form-control"
            ng-model="customer.Title">
    </div>
</div>
<div class="form-group">
    <label for="txtWorkPhone" class="col-lg-2 control-label">Work Phone:</label>
    <div class="col-lg-6">
        <input id="txtWorkPhone" type="text" readonly class="form-control"
            ng-model="customer.WorkPhone">
    </div>
</div>
<div class="form-group">
    <label for="txtHomePhone" class="col-lg-2 control-label">Home Phone:</label>
    <div class="col-lg-6">
        <input id="txtHomePhone" type="text" readonly class="form-control"
            ng-model="customer.HomePhone">
    </div>
</div>
<div class="form-group">
    <label for="txtEmailAddress" class="col-lg-2 control-label">Email Addresss:</label>
    <div class="col-lg-6">
        <input id="txtEmailAddress" type="text" readonly class="form-control"
            ng-model="customer.Email">
    </div>
</div>
</fieldset>
</div>
<hr />
<a href="#/edit/{{customer.Id}}">Edit this Customer</a>
<br />
<a href="#/">Return to Customers List</a>

```

39. Save and close **view.html**.

40. Open the view file in the **views** folder named **edit.html** and replace the contents with the following HTML layout. If you'd rather not type all this code by hand, you can copy it from the **edit.html.txt** file in the **StarterFiles** folder for this lab.

```

<h3>Edit Customer</h3>
<div class="form-horizontal">
    <fieldset>
        <div class="form-group">
            <label for="txtID" class="col-lg-2 control-label">ID:</label>
            <div class="col-lg-6">
                <input id="txtID" type="text" readonly class="form-control"
                    ng-model="customer.ID">
            </div>
        </div>
        <div class="form-group">
            <label for="txtFirstName" class="col-lg-2 control-label">First Name:</label>
            <div class="col-lg-6">
                <input id="txtFirstName" type="text" class="form-control"
                    ng-model="customer.FirstName">
            </div>
        </div>
        <div class="form-group">
            <label for="txtLastName" class="col-lg-2 control-label">Last Name:</label>
            <div class="col-lg-6">
                <input id="txtLastName" type="text" class="form-control"
                    ng-model="customer.Title">
            </div>
        </div>
        <div class="form-group">
            <label for="txtWorkPhone" class="col-lg-2 control-label">Work Phone:</label>

```

```

        <div class="col-lg-6">
            <input id="txtWorkPhone" type="text" class="form-control"
                ng-model="customer.WorkPhone">
        </div>
    </div>
    <div class="form-group">
        <label for="txtHomePhone" class="col-lg-2 control-label">Home Phone:</label>
        <div class="col-lg-6">
            <input id="txtHomePhone" type="text" class="form-control"
                ng-model="customer.HomePhone">
        </div>
    </div>
    <div class="form-group">
        <label for="txtEmailAddress" class="col-lg-2 control-label">Email Address:</label>
        <div class="col-lg-6">
            <input id="txtEmailAddress" type="text" class="form-control"
                ng-model="customer.Email">
        </div>
    </div>
    <div class="form-group">
        <label for="txtID" class="col-lg-2 control-label">ETag:</label>
        <div class="col-lg-6">
            <input id="txtID" type="text" readonly class="form-control"
                ng-model="customer.__metadata.etag">
        </div>
    </div>
    <div class="form-group">
        <div class="col-lg-offset-2">
            <input id="cmdSave" type="button" class="button" value="Save"
                data-ng-click="updateCustomer()" />
        </div>
    </div>
</fieldset>
</div>

<hr />

<a href="#/">Return to customers list</a>

```

41. Save and close **edit.html**.
42. Test your work by pressing **{F5}** to begin a debugging session. The app should initialize the start page using the view defined in **Home.html** which should display a table of customers. However, now the app should support full CRUD functionality.
43. Click on the **Add Customer** link and make sure you are able to add a new customer item.
44. Test the links in the table in the home view. You should be able to click **View** and navigate to a view which displays the details of a single item.
45. You should be able to click **Edit** and navigate to a view which allows you to edit an existing item and save your changes.
46. You should be able to click **Delete** and delete a customer item from the list.

You have now completed this lab.

a)

You have now completed this lab where you have created and tested an app part with custom properties.