

# Developing and Debugging a Provider-hosted Add-in

**Lab Time:** 60 minutes

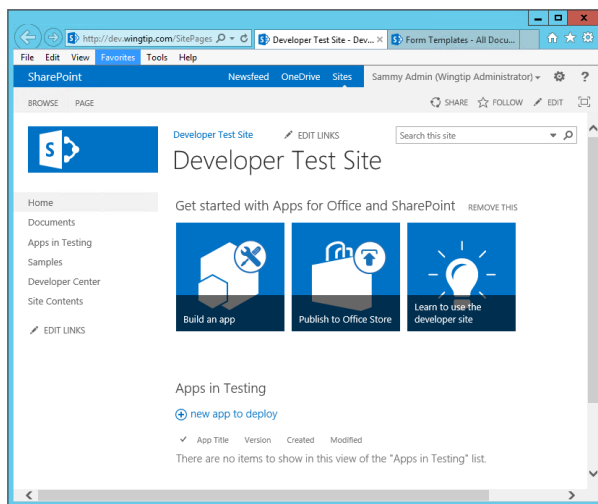
**Lab Folder:** C:\Student\Modules\ProviderHostedAddins\Labs

**Lab Overview:** In this lab you will create and develop a provider-hosted app project. This will give you experience testing and debugging a provider-hosted app as well as using the SharePoint app Chrome control to create a user interface that is similar to the host web. You will also create a user interface experience with allows user to read and write customer data from the SQL Server database named **WingtipCRM** which you created in an earlier lab.

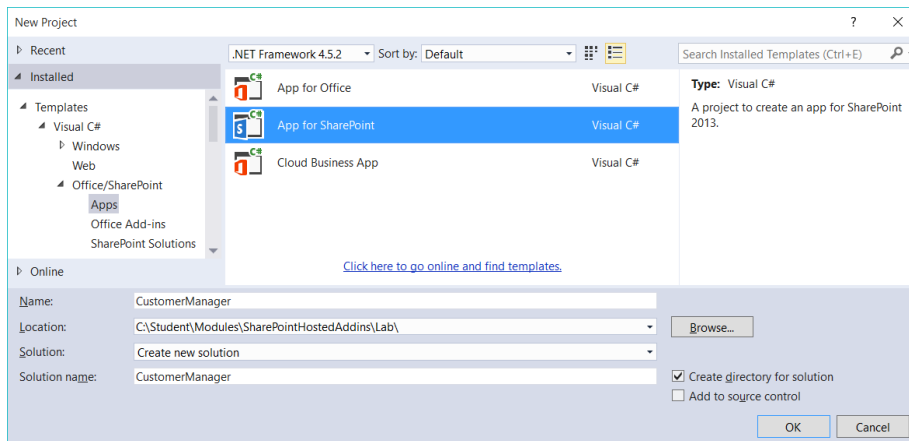
## Exercise 1: Creating a Provider-hosted Add-in Project in Visual Studio 2015

In this exercise you will create a new Provider-Hosted App.

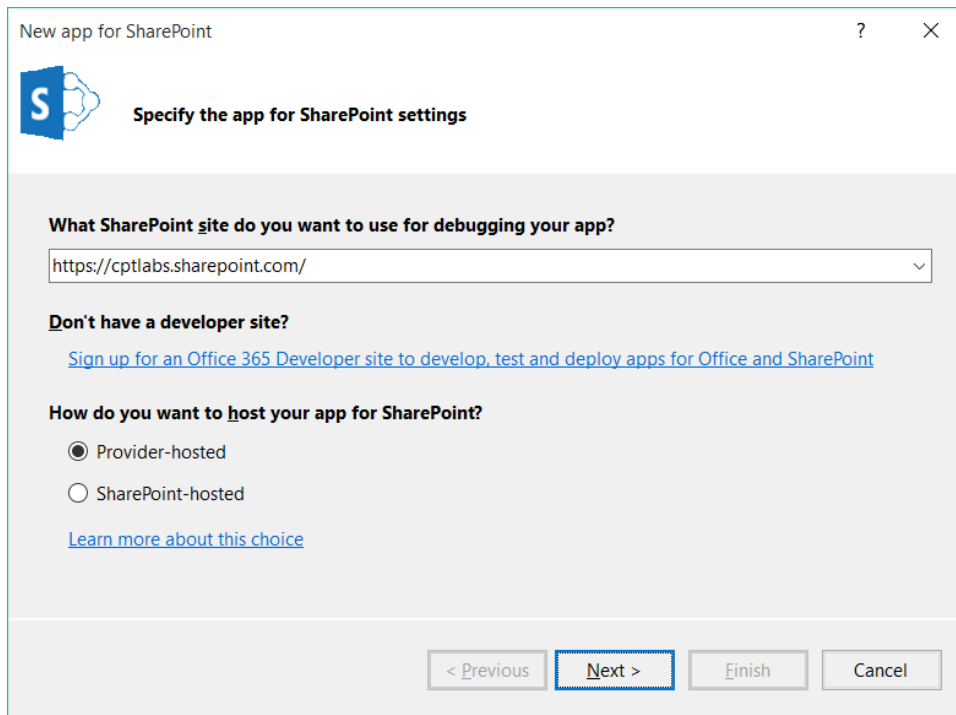
1. Make sure your developer test site is up and running so you have a place to debug the app you are going to create.
  - a) Using the browser, navigate to your Office 365 developer test site.



2. Launch **Visual Studio 2015**.
3. Create a new SharePoint provider-hosted app project in Visual Studio 2013.
  - a) In Visual Studio select **File → New → Project**.
  - b) Fill out the **New Project** dialog as follows...
    - i) Find the **App for SharePoint 2013** template under the **Templates → Visual C# → Office / SharePoint → Apps** section.
    - ii) **Name:** CustomerManager
    - iii) **Location:** C:\Student\Modules\ProviderHostedAddins\Lab
    - iv) Click **OK**.




- c) In the **New App for SharePoint** wizard, use the following values to complete the wizard and click **Finish**.
- What site do you want to use for debugging? **Your developer site URL**
  - How do you want to host your app for SharePoint? **Provider-hosted**
  - Click **Next**.



- d) On the **Specify the target SharePoint version** page, select **SharePoint Online** and click **Next**.

New app for SharePoint ? X

 **Specify the target SharePoint version**

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.

What's the earliest version of SharePoint that you want your app to target?


☐ SharePoint 2013

☒ SharePoint Online


< Previous Next > Finish Cancel

- e) On the **Specify the web project type** page, select **ASP.NET Web Forms Application** and click **Next**.

New app for SharePoint ? X

 **Specify the web project type**

A cloud app for SharePoint consists of an app for SharePoint that is deployed directly to a SharePoint site and a separately deployed web application.

 Which type of web application project do you want to create?

☒ ASP.NET Web Forms Application

☐ ASP.NET MVC Web Application

< Previous Next > Finish Cancel

- f) On the **Configure authentication settings** page, select Use **Windows Azure Control Service** and click **Finish**.

New app for SharePoint

**Configure authentication settings**

**How do you want your app to authenticate?**

☒ Use Windows Azure Access Control Service (for SharePoint cloud apps)

☐ Use a certificate (for SharePoint on-premises apps using high-trust)

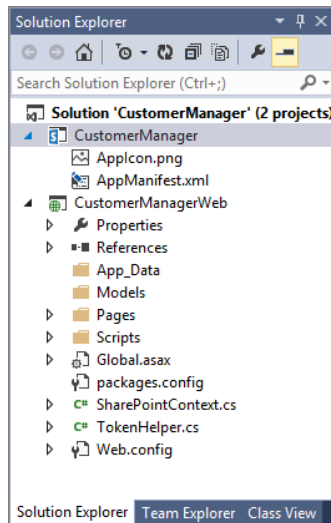
Certificate location:

Password:

Issuer ID:

< Previous   Next >   **Finish**   Cancel

4. Once the **New app for SharePoint** wizard completes, take a moment to inspect what has been created.
  - a) You should be able to see that there is new Visual Studio solution with two projects.

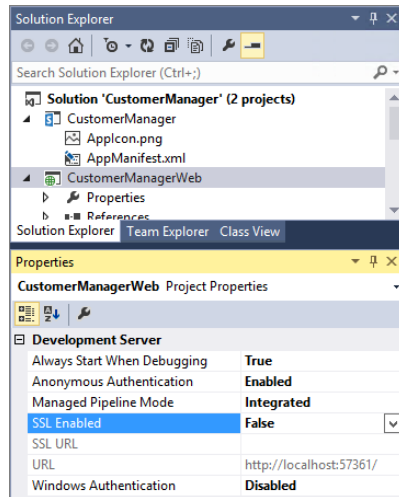


- b) The top project named **CustomerManager** is the project for the app itself. This project contains the **AppManifest.xml** file.
  - c) The second project below named **CustomerManagerWeb** is an ASP.NET Web Forms project which will be used to implement the remote web of this provider hosted app.
  - d) The **Web Project** property of the top project named **CustomerManager** references the **CustomerManagerWeb** project.
5. Change the **CustomerManager** app project's authentication method to use internal authentication:
  - a) In the project **CustomerManager** project, locate the **AppManifest.xml** file.
  - b) Open the **AppManifest.xml** file in Code View by right-clicking it and selecting **View Code**.
  - c) First find the **<Title>** element. Add a space in the Title so that the name looks like this **Customer Manager**
  - d) Now find the **<AppPrincipal>** element. Remove the **<RemoteWebApplication>** element and replace it with **<Internal/>** so the **<AppPrincipal>** element now looks like this:

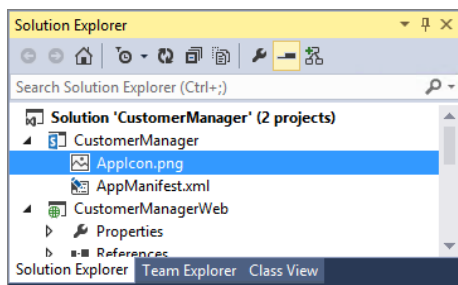
```
<AppPrincipal>
  <Internal/>
```

```
</AppPrincipal>
```

- e) Save and close the **AppManifest.xml** file
- 6. Configure the projects to disable the use of SSL so everything happens over HTTP rather than HTTPS:
  - a) Select the project **CustomerManagerWeb** in the **Solution Explorer** tool window.
  - b) Look in the **Properties** tool window. Look at the **SSL Enabled** property and make sure it is set to **False**.



- 7. Add a custom image to the project to use as the startup icon for the app.
  - a) Look inside the **CustomerManager** project and locate the file named **AppIcon.png**.



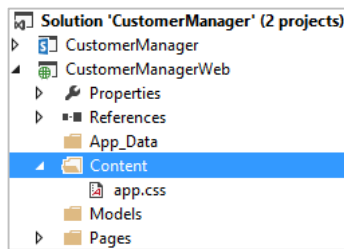
- b) Delete the file named **AppIcon.png** from the project and replace with a file with the same name located at the following path. (Hint: Right Click on **CustomerManager** Project and select **Add → Existing Item...**)

```
C:\Student\Modules\ProviderHostedApps\Labs\StarterFiles\AppIcon.png
```

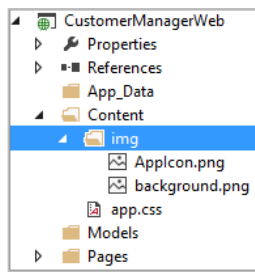
- c) Open the **AppManifest.xml** file by double clicking on it in the Solution Explorer
    - d) Click the button to the right of the Icon: Text box (...)
    - e) Select the **AppIcon.png** file located in the default location.
    - f) Save and close the **AppManifest.xml** file
- 8. Add CSS styles and image files to the project.
  - a) Within the **CustomerManagerWeb** project, create a new folder named **Content** at the root of the project.
  - b) Using the Windows Explorer, locate the CSS file named **app.css** at the following path.

```
C:\Student\Modules\ProviderHostedApps\Lab\StarterFiles\app.css
```

- c) Add a copy of the **app.css** file into the **Content** folder of the **CustomerManagerWeb** project. When you have completed this step, the structure of your project should look like the one in the following screenshot.

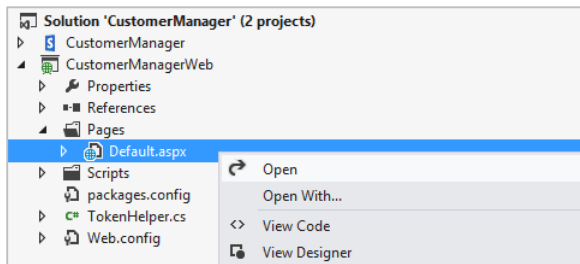


- d) Create a child folder in the **Content** folder named **img** and then add a copy of the two images files in the **StarterFiles** folder named **AppIcon.png** and **BackGround.png**.



9. Modify the HTML for the app's start page:

- a) Within the **CustomerManagerWeb** project, right-click the **Pages\Default.aspx** file and select **Open**.



- b) Inspect (*but do not modify*) the ASP.NET **Page** directive at the top of **Default.aspx**.

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"
    Inherits="CustomerManagerWeb.Pages.Default" %>
```

- c) Replace the existing DOCTYPE declaration with the simpler DOCTYPE declaration used in HTML5.

```
<!DOCTYPE html>
```

- d) Modify the **title** element within the **head** section to contain the text of **Customer Manager**.

```
<head runat="server">
  <title>Customer Manager</title>
</head>
```

- e) Add a link to the CSS file named **app.css**.  
(Hint: the easiest way to add a link to a css file is to drag it from the Solution Explorer and drop it into the .aspx page)

```
<head id="Head1" runat="server">
  <title>Customer Manager</title>
  <link href=" ../Content/app.css" rel="stylesheet" />
</head>
```

- f) Modify the **body** section of the page to match the following HTML fragment. Be sure add the IDs for **pageWidth**, **topnav**, **topHeader** and **contentBody** exactly as you see them in the following code. This is important because these IDs are referenced in **app.css**.

```
<body>
  <form id="form1" runat="server">
```

```

<div id="pagewidth">
    <nav id="topnav">
        <asp:HyperLink ID="lnkHostWeb" runat="server">Host Web</asp:HyperLink>
    </nav>

    <header id="topHeader">
        <h2>Customer Manager Start Page</h2>
    </header>

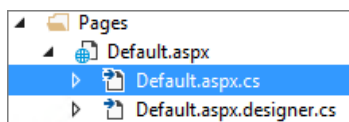
    <div id="contentBody">
        <asp:PlaceHolder ID="pageContent" runat="server"></asp:PlaceHolder>
    </div>
</div>
</form>
</body>

```

g) When you are done, save your changes and close **Default.aspx**.

10. Remove the starter C# code that Visual Studio has added to **Default.aspx.cs**.

a) Inside the **Pages** folder of the **CustomerManagerWeb** project, open the C# source files named **Default.aspx.cs**.



b) Inspect the code that Visual Studio has added to **Default.aspx.cs**. This code presents a problem because it will fail in a project like the **CustomerManager** project which has an authentication setting of **Internal**.

```

namespace CustomerManagerWeb {
    public partial class Default : System.Web.UI.Page {
        protected void Page_PreInit(object sender, EventArgs e) {
            Uri redirectUrl;
            switch (SharePointContextProvider.CheckRedirectionStatus(Context, out redirectUrl)) {
                case RedirectionStatus.Ok:
                    return;
                case RedirectionStatus.ShouldRedirect:
                    Response.Redirect(redirectUrl.AbsoluteUri, endResponse: true);
                    break;
                case RedirectionStatus.CanNotRedirect:
                    Response.Write("An error occurred while processing your request.");
                    Response.End();
                    break;
            }
        }

        protected void Page_Load(object sender, EventArgs e) {
            // The following code gets the client context and Title property by using TokenHelper.
            // To access other properties, the app may need to request permissions on the host web.
            var spContext = SharePointContextProvider.Current.GetSharePointContext(Context);

            using (var clientContext = spContext.CreateUserClientContextForSPHost()) {
                clientContext.Load(clientContext.Web, web => web.Title);
                clientContext.ExecuteQuery();
                Response.Write(clientContext.Web.Title);
            }
        }
    }
}

```

c) Remove all the code that has been added to the **Page\_Load** and **Page\_Preinit** methods. When you have completed this step, **Page\_Load** and **Page\_Preinit** should be empty methods.

```

namespace CustomerManagerWeb {
    public partial class Default : System.Web.UI.Page {
        protected void Page_PreInit(object sender, EventArgs e) {

        }

        protected void Page_Load(object sender, EventArgs e) {

        }
    }
}

```

```
}
}
```

- d) Now add code to the **Page\_Load** method

```
protected void Page_Load(object sender, EventArgs e) {

    // configure link back to host web
    lnkHostWeb.NavigateUrl = Request.QueryString["SPHostUrl"];

    // add content to page
    pageContent.Controls.Add( new LiteralControl("Hello from server-side C# code"));

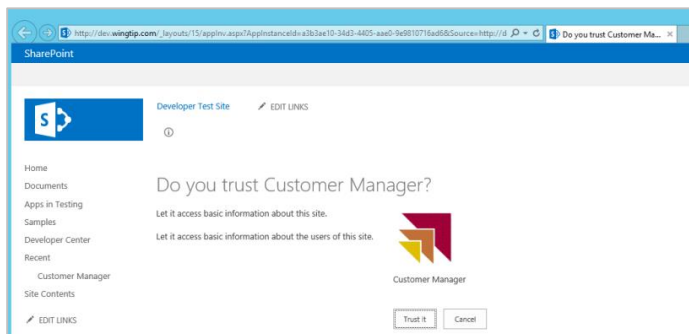
}
```

- e) When you are done, save your changes and close **Default.aspx.cs**.

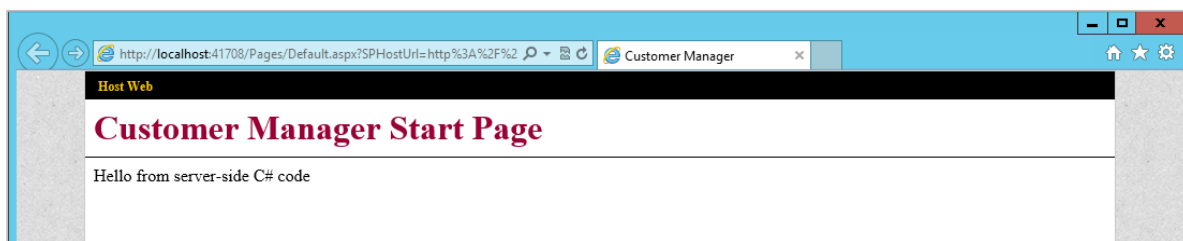
Now you are at a point when you can build and test the app using the Visual Studio debugger.

## 11. Test your application using the Visual Studio debugger.

- Begin a debugging session in Visual Studio by pressing the **{F5}** key or running the command **Debug → Start Debugging**.
- When Visual Studio installs the app, SharePoint might prompt you with a page asking you whether you trust the app which will be required for the app to be installed. If you are prompted, click the **Trust It** button.

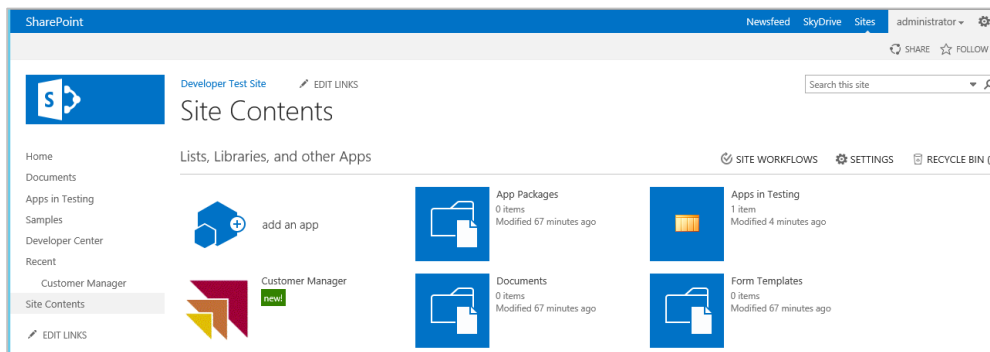


- c) Once the solution has been deployed, Internet Explorer will launch and navigate to the start page which is **Default.aspx**.



- Test the **Host Web** hyperlink in the top left corner of the start page which links back to the host web. When you click on this hyperlink, you should navigate back to the developer test site at <http://dev.wingtip.com>.
- Once you have navigated to the developer test site at <http://dev.wingtip.com>, click on the **Site Contents** link on the left-hand navigation. You should see your app listed in the **Lists, Libraries and other Apps** section of the page.



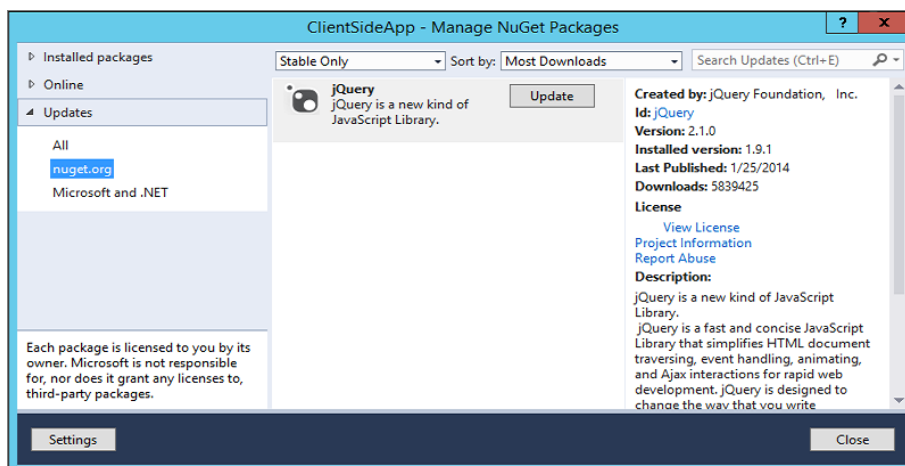


12. Close the browser to stop the debugger and go back to Visual Studio.

## Exercise 2: Creating a Multi-page User Interface using the SharePoint Chrome Control

In this exercise, you will build out the user interface as a multi-page app. You will accomplish this in the **CustomerManager** project using a custom ASP.NET master page and the SharePoint Chrome Control.

1. Return to Visual Studio.
2. If the Visual Studio debugger is still running, stop the current debugging session.
3. Select the **CustomerManagerWeb** project.
4. Update the jQuery library in the **CustomerManagerWeb** project to the most recent version.
  - a) Right-click on the **CustomerManagerWeb** project and click **Manage NuGet Packages....**
  - b) Select **Updates** on the left-hand side of the **Manage NuGet Packages** dialog.
  - c) Select the **jQuery** library and then click the **Update** button.

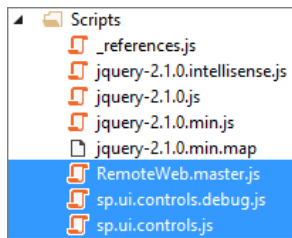


Note that the jQuery library version number you see might be more recent than the one shown in the previous screenshot.

5. Add a few more JavaScript files into the Scripts folder of the **CustomerManagerWeb** project.
  - a) Right-click the **Scripts** folder and select **Add → Existing Item**
  - b) In the **Add Existing Item** dialog, browse to the **StarterFiles** folder at the following path:

**C:\Student\Modules\ProviderHostedApps\Lab\StarterFiles**

- c) Add the following files:
  - i) **RemoteWeb.master.js** - custom JavaScript code used behind the master page to initialize the Chrome control.
  - ii) **sp.ui.controls.js** - this is the JavaScript library which contains the Chrome control.
  - iii) **sp.ui.controls.debug.js** - this is the debug version of the JavaScript library which contains the Chrome control.



6. Next, add a master page to the **Pages** folder.

- a) Right-click the **Pages** folder and select **Add → Existing Item**.
- b) In the **Add Existing Item** dialog, browse to the **StarterFiles** folder at the following path:

**C:\Student\Modules\ProviderHostedApps\Lab\StarterFiles**

- c) Select the **RemoteWeb.master** file and click **Add**

7. Modify the master page to link to CCS and JavaScript files.

- a) Open **RemoteWeb.master** in Code View and examine the **head** section. There is an HTML comment which lets you know where you can add links to CSS files and JavaScript files.

```
<head>
<title>Customer Manager App</title>
<!-- add links to CSS and JavaScript here -->
<asp:ContentPlaceHolder ID="PlaceholderAdditionalPageHead" runat="server" />
</head>
```

- b) Add a link to the CSS file named **app.css**.  
(Hint: the easiest way to add these is by dragging them from the Solution Explorer into the location you wish in the .Master File

```
<head>
<title>Customer Manager App</title>
<link href="../Content/app.css" rel="stylesheet" />
<asp:ContentPlaceHolder ID="PlaceholderAdditionalPageHead" runat="server" />
</head>
```

- c) Add a script link for the jQuery library.
- d) Add a script link to the JavaScript library with the Chrome Control which is named **sp.ui.controls.js**.
- e) Add a script link to **RemoteWeb.master.js**.

```
<head>
<title>Customer Manager App</title>
<link href="../Content/app.css" rel="stylesheet" />
<script src="../Scripts/jquery-2.1.0.js"></script>
<script src="../Scripts/sp.ui.controls.js"></script>
<script src="../Scripts/RemoteWeb.master.js"></script>
<asp:ContentPlaceHolder ID="PlaceholderAdditionalPageHead" runat="server" />
</head>
```

8. Open **RemoteWeb.master.js**.

- a) Examine the JavaScript code inside. Note there is no need for you to modify this file.

```
// determine URL back to host web
var hostWebUrl = decodeURIComponent(getQueryStringParameter("SPHostUrl"));

// create settings object for Chrome control
var options = {
    siteUrl: hostWebUrl,
    siteTitle: "Back to Host Web",
    appHelpPageUrl: "help.aspx?SPHostUrl=" + hostWebUrl,
    appIconUrl: "/Content/img/AppIcon.png",
    appTitle: "Customer Manager App",
    settingsLinks: [
        { linkUrl: "start.aspx?SPHostUrl=" + hostWebUrl, displayName: "Home" },
        { linkUrl: "about.aspx?SPHostUrl=" + hostWebUrl, displayName: "About" },
        { linkUrl: "contact.aspx?SPHostUrl=" + hostWebUrl, displayName: "Contact" }
    ]
}
```

```
};

// create Chrome control instance
var nav = new SP.UI.Controls.Navigation("chrome_ctrl_container", options);
nav.setVisible(true);
```

- b) This JavaScript code in **n RemoteWeb.master.js** is written to go through these steps
    - i) Determine the URL back to host web
    - ii) Create a settings object used to initialize Chrome Control
    - iii) Create the Chrome Control instance
    - iv) Call **setVisible** method to display on the hosting page
  - c) Once you have looked at the JavaScript code and understand it, close **RemoteWeb.master.js**.
9. Add a few page files to the **Pages** folder to create the multi-page user interface experience:
- a) Create a new start page named **start.aspx**:
    - i) Right-click the **Pages** folder and select **Add → New Item**.
    - ii) In the **Add New Item** dialog, select the **Web Form with Master Page** template within the **Visual C# \ Web** category and give the new file a name of **Start.aspx**. Click **Add**
    - iii) When prompted to select a master page, select **RemoteWeb.master** in the **Pages** folder. Click **OK**
    - iv) Once the page has been created, replace the contents of the new page **start.aspx**, except the **@Page** directive at the top of the file, with the following markup:

```
<asp:Content ContentPlaceHolderID="PlaceholderMain" runat="server">
  <h2>Customer Manager Start Page</h2>
</asp:Content>
```

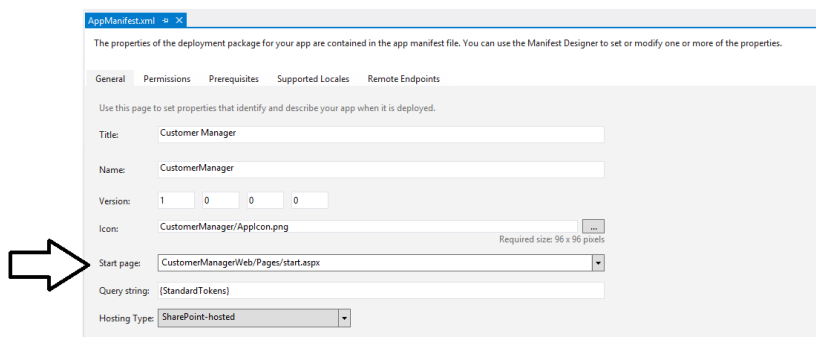
- b) Repeat the previous step to create the following pages & include the appropriate **<h1>** tag:
  - i) About.aspx
  - ii) Contact.aspx
  - iii) Help.aspx

Be sure to enter the names of these page files exactly as shown above. That's because the names of these page files have been hardcoded into the code that initializes the Chrome control inside **RemoteWeb.master.js**. Also, be sure to update the **<h2>** tag in each page so that it displays an appropriate message (i.e. for About.aspx **<h2>** should be set to **About Page**)

10. Ensure all four of the pages you just created reference the new master page you added to the **Pages** folder previously. Each of these pages should have the **MasterPageFile** attribute in the **@Page** directive that looks like this:

```
MasterPageFile="~/Pages/RemoteWeb.Master"
```

11. Now, change the start page of the app to use one of the new pages you just created:
  - a) Using the **Solution Explorer** tool window, within the **CustomerManager** project, right-click the **AppManifest.xml** file and select **Open**.
  - b) On the **General** tab of the app manifest designer, locate the **Start page** setting and modify it to point to **CustomerManagerWeb/Pages/Start.aspx**.



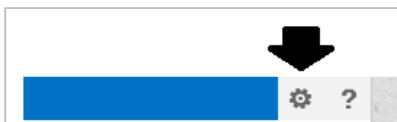
12. Save all changes: **File → Save All**.

## Build and Test the Project

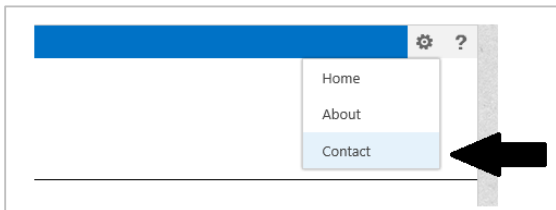
13. Build and test your application by pressing **[F5]** or **Debug → Start Debugging**.
14. Once the solution has been deployed, Internet Explorer will launch and navigate to the **start.aspx** page of the remote web.



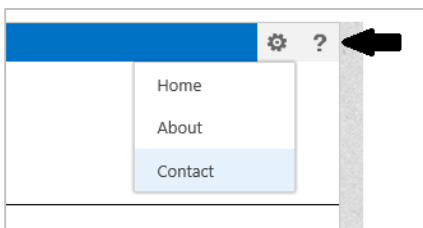
15. Now it is time for experiment by moving from page to page to ensure that all pages in this app have a similar look and feel:
  - a) Locate the Chrome Controls navigation menu which displays as a gear icon located in the top right corner of the page just to the left of the Help icon.



- b) Click on the navigation menu to drop down its list of pages to which you can navigate. Use this navigation menu to move between the **Home** page, the **About** page and **Contact** page.



- c) Click on the Help link and make sure you can navigate to the Help page as well.

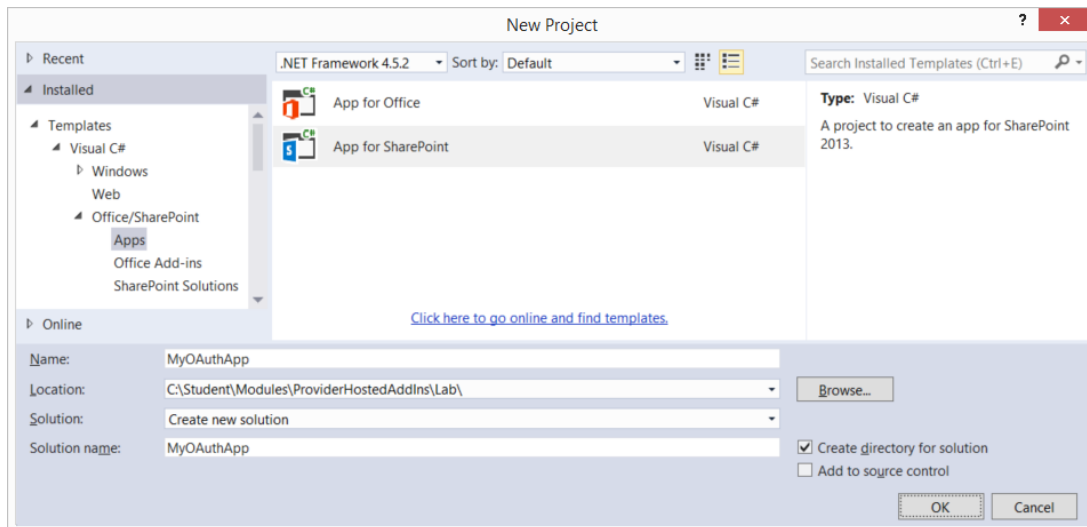


16. Close the browser to stop the debugger and go back to Visual Studio.

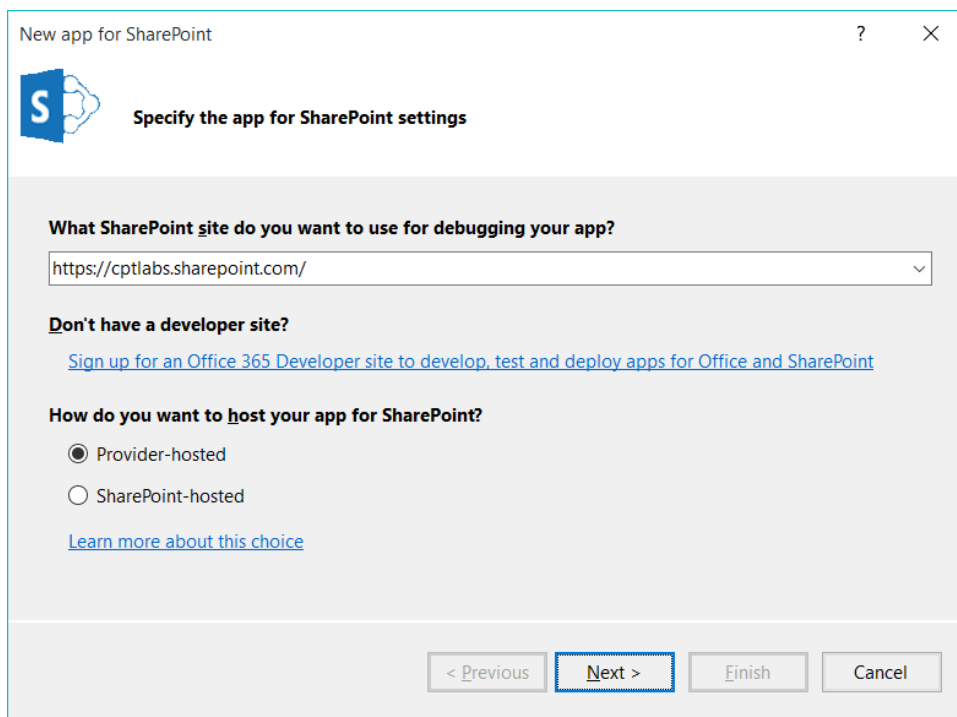
## Exercise 3: Create a SharePoint Provider-Hosted Add-in that Uses OAuth 2.0

In this exercise you will create a new SharePoint Provider-Hosted app that will use OAuth 2.0 authentication.

17. Launch **Visual Studio 2015** (if it is not already open).
18. Create a new project in Visual Studio 2015 for hosting your app:
  - a) 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:** MyOAuthApp
    - iii) **Location:** C:\Student\Modules\SharePointHostedAddins\Lab\
    - iv) When the **New Project** dialog looks like the screenshot below, click the **OK** button.




- c) In the **New App for SharePoint** wizard, use the following values to complete the wizard and click **Finish**.
- What site do you want to use for debugging? **Your developer site URL**
  - How do you want to host your app for SharePoint? **Provider-hosted**
  - Click **Next**.



- d) On the Specify the target SharePoint version page, select SharePoint Online and click Next.

New app for SharePoint ? X

 **Specify the target SharePoint version**

**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.**

**What's the earliest version of SharePoint that you want your app to target?**


☐ SharePoint 2013

☒ SharePoint Online


< Previous Next > Finish Cancel

- e) On the Specify the web project type page, select ASP.NET Web Forms Application and click Next.

New app for SharePoint ? X

 **Specify the web project type**

**A cloud app for SharePoint consists of an app for SharePoint that is deployed directly to a SharePoint site and a separately deployed web application.**

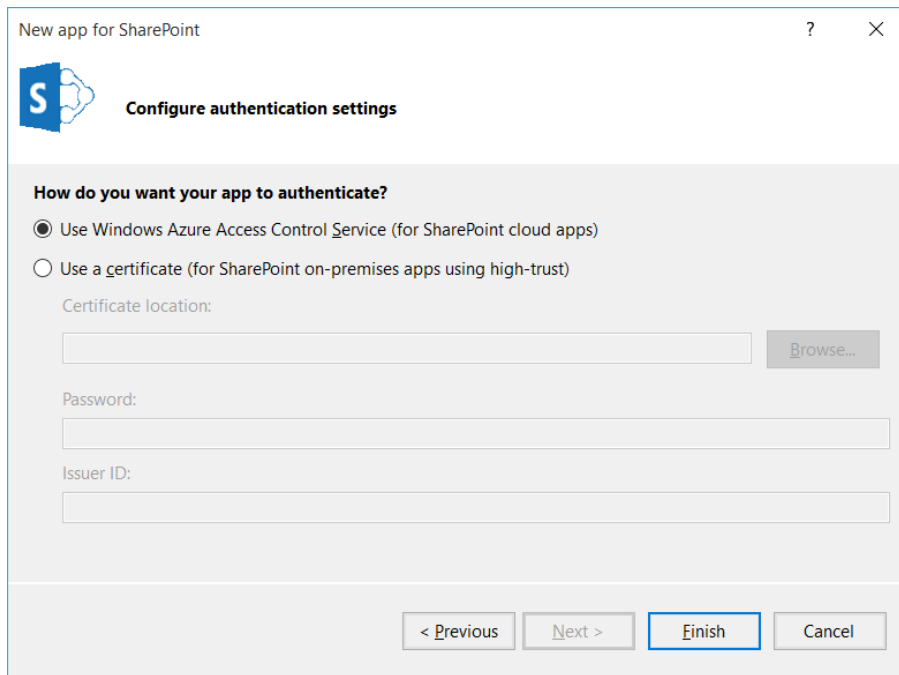
 **Which type of web application project do you want to create?**

☒ ASP.NET Web Forms Application

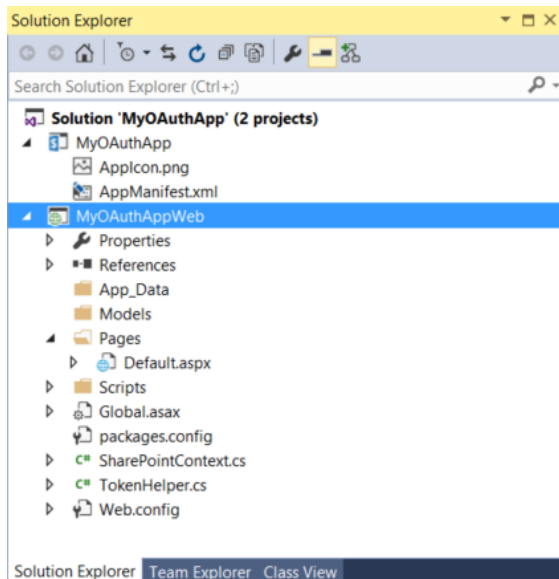
☐ ASP.NET MVC Web Application

< Previous Next > Finish Cancel

- f) On the Configure authentication settings page, select Use Windows Azure Control Service and click Finish.



19. Once the **New app for SharePoint** wizard completes, take a moment to inspect what has been created.
- You should be able to see that there is new Visual Studio solution with two projects.



## Make some modification to the App project named MyOAuthApp Project

20. Replace the **AppIcon.png** file with a custom image.
- In the Solution Explorer, look inside the **MyOAuthApp** project and locate the image file **AppIcon.png**.
  - Using Windows Explorer, look inside the **StarterFiles** folder for this lab at the following path.
- C:\Student\Modules\ProviderHostedAddins\Lab\StarterFiles**
- Locate the custom image file named **AppIcon.png**.
  - Use the **AppIcon.png** file in the **StarterFiles** folder to replace the **AppIcon.png** file in the **MyOAuthApp** project.
21. Modify the **AppManifest.xml** file.
- In Solution Explorer, double-click on **AppManifest.xml** open it in Visual Studio's App Manifest Designer.

- b) Update the Title property from **MyOAuthApp** to **My OAuth App**.
- c) Click on the **Permissions** tab and then add a permission with a **Scope** of **Web** and a **Permission** level of **Read**.

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: My OAuth App

Name: MyOAuthApp

Version: 1 0 0

Icon: MyOAuthApp/AppIcon.png Required size: 96 x 96 pixels

Start page: MyOAuthAppWeb/Pages/Default.aspx

Query string: (StandardTokens)

Hosting type: Provider-hosted

- d) Save your changes and close **AppManifest.xml**.

## Implement the Web Project named **MyOAuthAppWeb**

22. In Solution Explorer, move down to the Web Project named **MyOAuthAppWeb**.

23. Add a CSS file and an image file for the start page.

- a) Inside the **MyOAuthAppWeb** project, add a new top-level folder named **Content**.
- b) Using Windows Explorer, locate the CCS file in the **StarterFiles** folder at the following path.

**C:\Student\Modules\ProviderHostedAddins\Lab\StarterFiles\Content\App.css**

- c) Add the **App.css** file from the **StarterFiles** folder into the new **Content** folder in the **MyOAuthApp** project.
- d) Create a child folder named **img** inside the **Content** folder.
- e) Using Windows Explorer, locate the CCS file in the **StarterFiles** folder at the following path.

**C:\Student\Modules\ProviderHostedAddins\Lab\StarterFiles\Content\img\AppIcon.gif**

- f) Add the **AppIcon.gif** file from the **StarterFiles** folder into the new **img** folder in the **MyOAuthApp** project.
- g) Open the CSS file named **App.css** and have a quick look at the set of CSS rules inside.
- h) When you are done, close the **App.css** file without saving any changes.

24. Modify the app's start page named **Default.aspx**.

- a) Using the Solution Explorer, look inside the **Pages** folder of the **MyOAuthAppWeb** project and locate **Default.aspx**.
- b) Double click on **Default.aspx** to open it in the Web Forms Editor of Visual Studio.

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs" Inherits="MyFir" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1" %>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title></title>
</head>
<body>
<form id="form1" runat="server">
<div>
</div>
</form>
</body>
</html>
```

- c) Delete all contents from **Default.aspx** except for the first line with the **<@Page>** directive.
- d) Using Windows Explorer, locate the file named **Default.aspx.txt** in the **StarterFiles** folder at the following path.

**C:\Student\Modules\ProviderHostedAddins\Lab\StarterFiles\Default.aspx.txt**



- e) Open the file named **Default.aspx.txt** in **Notepad.exe** and copy its entire contents to the Windows clipboard.
- f) Return to Visual Studio.
- g) Navigate to **Default.aspx** which should still be open in the Web Forms Editor.
- h) Position your cursor in **Default.aspx** right below the first line with the **<@Page>** directive
- i) Paste the content of the Windows clipboard into **Default.aspx**.
- j) Save your changes to **Default.aspx**.

25. Examine the layout of the HTML you just pasted into **Default.aspx**.

- a) Make sure you are looking at **Default.aspx** in Code View.
- b) Note in the **head** section that there is already a link to the CSS file named **App.css**.
- c) The **body** section contains a **div** with an **id** of **page\_width** which contains child elements that make up the user interface.

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs" ... >

<!DOCTYPE html>

<html>

<head runat="server">
  <meta charset="utf-8" />
  <meta http-equiv="X-UA-Compatible" content="IE=10" />
  <title>My First S2S App</title>
  <link href="../../Content/App.css" rel="stylesheet" />
</head>

<body>
  <form id="form1" runat="server">

    <div id="page_width">
      <!-- more inside -->
    </div>

  </form>
</body>
</html>
```

26. Examine the server-side controls inside the **div** with the **id** of **page\_width**.

- a) First, you should see there is an ASP.NET **HyperLink** control with an **ID** of **HostWebLink**.

```
<div id="nav_bar">
  <asp:HyperLink ID="HostWebLink" runat="server" />
</div>
```

- b) Next, you should see two ASP.NET **Button** controls with **ID** values of **cmdGetTitleCSOM** and **cmdGetTitleREST**.

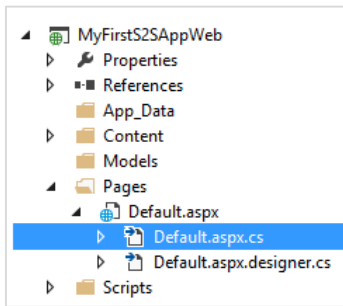
```
<nav id="toolbar">
  <asp:Button ID="cmdGetTitleCSOM" runat="server" Text="Get Title using CSOM" />
  <asp:Button ID="cmdGetTitleREST" runat="server" Text="Get Title using REST" />
</nav>
```

- c) Finally, there is an ASP.NET **Literal** control with an **ID** of **placeholderMainContent**.

```
<div id="content_box">
  <asp:Literal ID="placeholderMainContent" runat="server" ></asp:Literal>
</div>
```

27. Modify the server-side C# code behind the start page.

- a) In Solution Explorer, locate the code-behind file for **Default.aspx** named **Default.aspx.cs**.



- b) Open **Default.aspx.cs** and inspect the code inside that was generated by Visual Studio. You should be able to see that there is a code-behind class named **Default** for the page named **Default.aspx**. This Default code-behind class contains two methods in the **Default** code-behind class named **Page\_PreInit** and **Page\_Load**.

```
public partial class Default : System.Web.UI.Page {
    protected void Page_PreInit(object sender, EventArgs e) {
        // implementation details
    }

    protected void Page_Load(object sender, EventArgs e) {
        // implementation details
    }
}
```

- c) Delete the entire method named **Page\_PreInit**.  
d) Leave the **Page\_Load** method but delete all to code inside it so that it is an empty method.

```
public partial class Default : System.Web.UI.Page {
    protected void Page_Load(object sender, EventArgs e) {
    }
}
```

- e) Implement the **Page\_Load** method using the following code.

```
protected void Page_Load(object sender, EventArgs e) {
    SharePointContext spContext =
        SharePointContextProvider.Current.GetSharePointContext(Context);

    this.HostWebLink.NavigateUrl = spContext.SPHostUrl.AbsoluteUri;
    this.HostWebLink.Text = "Back to Host Web";
}
```

28. Test out your work by running the app in the Visual Studio Debugger.

- Press the **{F5}** key in Visual Studio to begin a debugging session for the app project.
- As Visual Studio installs the app in the test site, you will be prompted whether you trust the app. Click **Trust It**.
- Here we are
- Test it out. Make sure the link works.
- Close the browser and quit the debugging session.

29. Add event handlers behind the buttons

- Return to Visual Studio and the Web Project named **MyOAuthAppWeb**.
- Open the page named **Default.aspx** in the Web Forms Editor.
- Switch **Default.aspx** into Design View.
- When in Design View, double-click on the **Get Title with CSOM** button to generate an event handler in **Default.aspx.cs**.
- Return to Design View in **Default.aspx** and double-click on the **Get Title with REST** button to generate another event handler.
- When you're done, there should be two new event handlers named **cmdGetTitleCSOM\_Click** and **cmdGetTitleREST\_Click**.

```
public partial class Default : System.Web.UI.Page {
    protected void Page_Load(object sender, EventArgs e) {
        // ...
    }

    protected void cmdGetTitleCSOM_Click(object sender, EventArgs e) {
    }

    protected void cmdGetTitleREST_Click(object sender, EventArgs e) {
    }
}
```

## Make a server-side CSOM Call using C# and OAuth Authentication

30. Add the following code to **cmdGetTitleCSOM\_Click** to execute a CSOM call using S2S authentication.

```
protected void cmdGetTitleCSOM_Click(object sender, EventArgs e) {

    SharePointContext spContext =
        SharePointContextProvider.Current.GetSharePointContext(Context);

    using (var clientContext = spContext.CreateUserClientContextForSPHost()) {
        clientContext.Load(clientContext.Web);
        clientContext.ExecuteQuery();
        placeholderMainContent.Text = "Host web title (CSOM): " + clientContext.Web.Title;
    }

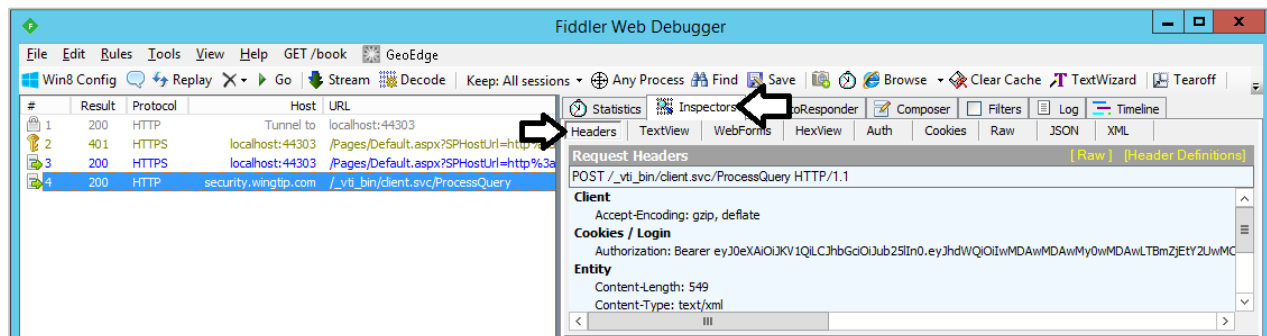
}
```

31. Test your work to ensure the CSOM call executes successful without any errors.

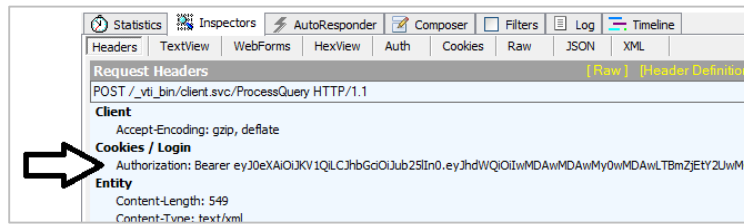
- Press the **{F5}** key to start another debugging session.
- If you are prompted to trust the app, click the **Trust It** button to complete the app installation.
- After the app has been installed, the Visual Studio debugger should redirect you to the app's start page in the remote web.
- Click the **Get Title using CSOM** button. When you click this button, the page should display a message with the title of the host web as shown in the following screenshot.
- Keep the browser window open because you will use it in the next step.

32. Inspect an S2S CSOM call using the Fiddler utility.

- Launch Fiddler (Press the **Windows** Key then Type **Fiddler** and click on the **Fiddler2** tile)
- Return to the app's Start page in the browser.
- Click the **Get Title using CSOM** button to execute a new CSOM.
- Return to Fiddler and inspect the requests.
- Select the CSOM request with the URL of **/\_vti\_bin/client.svc/ProcessQuery**.
- On the right-hand side of the Fiddler window, click the **Inspectors** tab and then the **Headers** tab in the row directly below.



- g) You should be able to verify that there is a request header named **Authorization**. The value of the **Authorization** header is the string value of *"Bearer "* parsed together with the S2S access token.



- h) When you have finished examining the call in Fiddler, close Internet Explorer and stop your Visual Studio debugging session.

## Make a server-side REST Call using C# and S2S Authentication

33. Return to Visual Studio and the Web Project named **MyOAuthAppWeb**.

34. Open the C# file named of **default.aspx.cs** if it is not already open.

35. Add the following using statements at the top of **default.aspx.cs**.

```
using System.Net;
using System.Xml.Linq;
```

36. Add the following code to **cmdGetTitleREST\_Click** to execute a REST call using S2S authentication.

```
protected void cmdGetTitleREST_Click(object sender, EventArgs e) {
    SharePointContext spContext =
        SharePointContextProvider.Current.GetSharePointContext(Context);

    string restUri = spContext.SPHostUrl + "_api/web/title";

    HttpWebRequest request = WebRequest.Create(restUri) as HttpWebRequest;
    request.Accept = "application/atom+xml";

    string spAccessToken = spContext.UserAccessTokenForSPHost;
    request.Headers["Authorization"] = "Bearer " + spAccessToken;

    HttpWebResponse response = request.GetResponse() as HttpWebResponse;
    XDocument responseBody = XDocument.Load(response.GetResponseStream());

    XNamespace nsDataService = "http://schemas.microsoft.com/ado/2007/08/dataservices";
    string hostWebTitle = responseBody.Descendants(nsDataService + "Title").First().Value;

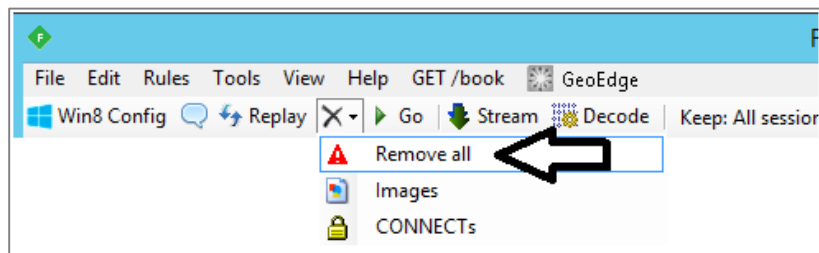
    placeholderMainContent.Text = "Host web title (REST): " + hostWebTitle;
}
```

37. Test your work to ensure the REST call executes successful without any errors.

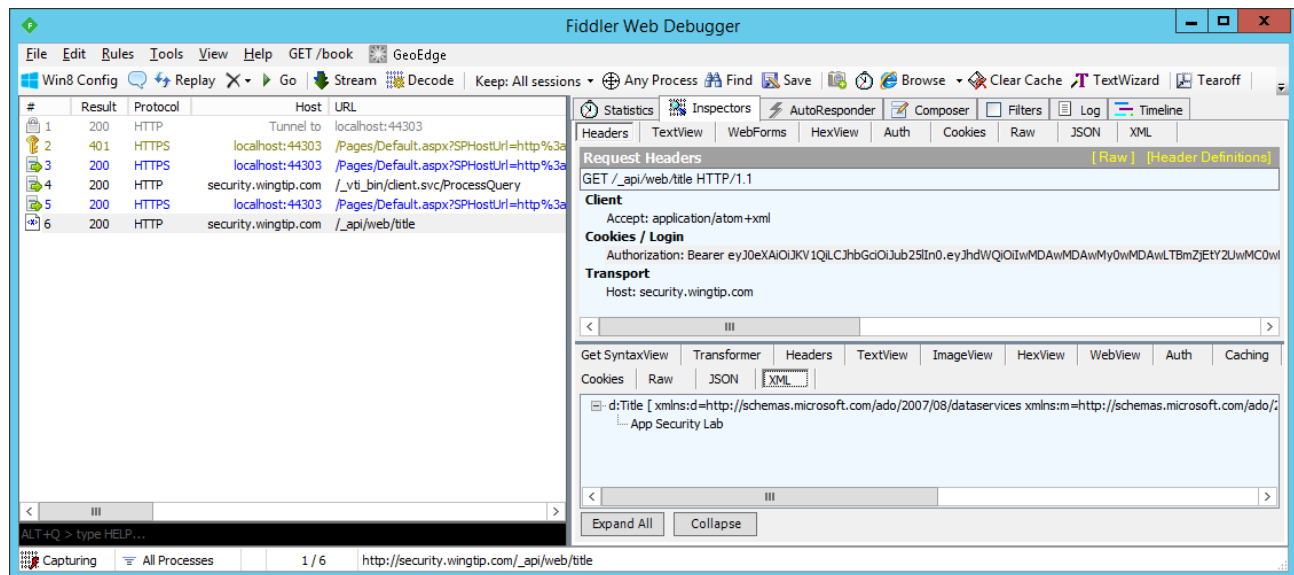
- Press the **{F5}** key to start another debugging session.
- If you are prompted to trust the app, click the **Trust It** button to complete the app installation.
- After the app has been installed, the Visual Studio debugger should redirect you to the app's start page in the remote web.
- Click the **Get Title using REST** button. When you click this button, the page should display a message with the title of the host web as shown in the following screenshot.
- Keep the browser window open because you will use it in the next step.

38. Inspect the REST call using the Fiddler utility.

- Return to Fiddler (or launch Fiddler if it's not already running).
- On the Fiddler toolbar, drop down the menu with X on it and select the Remove all command. This will clear the Fiddler window of all existing requests.



- Return to the app's Start page in the browser.
- Click the **Get Title using REST** button to execute a new REST call.
- Return to Fiddler and inspect the requests.
- Select the REST request with the URL of `/_api/web/title`.
- Verify that the request carries the Authorization header and a value just like the CSOM call did.



39. You are now done with the lab. You can close Internet Explorer and stop the debugging session.

In this exercise you created a new SharePoint Provider-Hosted App that uses S2S security. Using S2S security is both easy and fun!