## Extending Developing SharePoint Apps

**Lab Time**: 60 minutes

**Lab Folder**: [[StudentFolder]]\DevApps

**Lab Overview**: In this lab you will get your first hands-on experience working with the new SharePoint App model. Through the various exercises in this lab you will learn how to create the different types of apps such as a SharePoint-hosted app and a cloud-hosted app.

### Exercise 1: Setup Lab Environment

In this exercise you will setup your environment.

All exercises in this lab assume you will work in a new site collection, http://appdev.wingtip.com.

1. Setup a new site collection for this lab:
   1. Ensure you are logged into the **WingtipServer** server as **WINGTIP\Administrator**.
   2. Run a PowerShell script, found in the root lab folder for this module:
      1. Right-click **SetupModule.ps1** and select **Run with PowerShell**. This file can be found in the files associated with this lab:

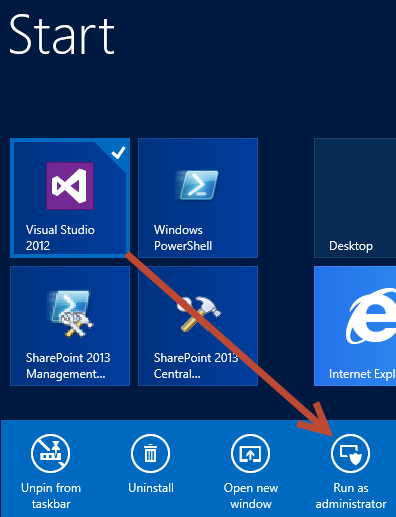
[..]\DevApps

* 1. When the script completes, it will launch a new browser and navigate to the lab site collection.
  2. Close the PowerShell console window.

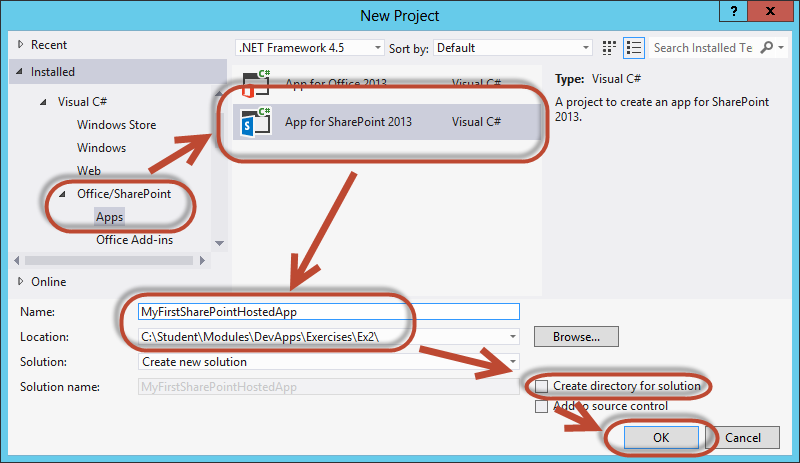
### Exercise 2: Create a SharePoint-Hosted App

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

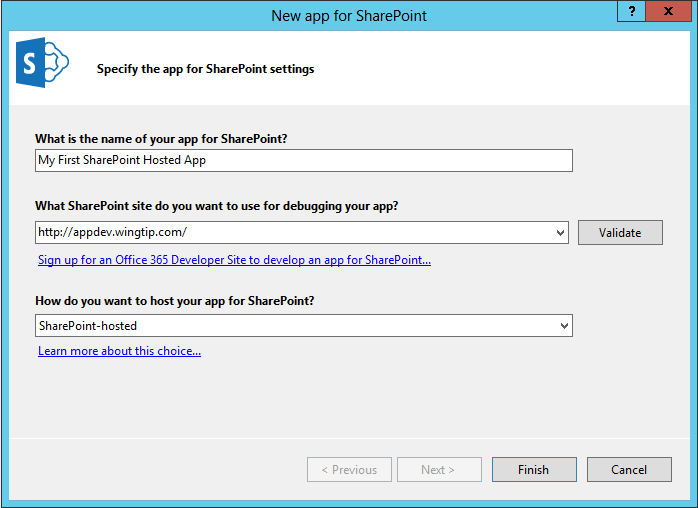
1. Create a new project in Visual Studio 2012:
   1. Launch **Visual Studio 2012** as administrator:
      1. Windows Keyboard Key 🡪 Right click on the Visual Studio 2012 tile and select Run as administrator.



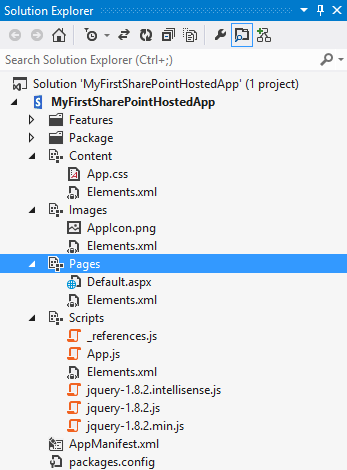
* 1. In Visual Studio select **File 🡪 New 🡪 Project**.
  2. In the **New Project** dialog:
     1. Find the **App for SharePoint 2013** template under the **Templates 🡪 Visual C# 🡪 Office / SharePoint 🡪 Apps** section.
     2. **Name**: MyFirstSharePointHostedApp
     3. **Location:** [..]\DevApps\Exercises\Ex2  
        (Where [..] represents the location of the student files (e.g. c:\student\..)
     4. Uncheck the **Create directory for solution** checkbox
     5. Click **OK** to create the project.



* 1. In the **New App for SharePoint** wizard, use the following values to complete the wizard:
     1. **What is the name of your App for SharePoint?** My First SharePoint Hosted App
     2. **What site do you want to use for debugging?** <http://appdev.wingtip.com>
     3. **How do you want to host your app for SharePoint?** SharePoint-hosted
  2. Click **Finish**.



1. Examine the default project setup for a SharePoint-Hosted app:
   1. Like a traditional SharePoint solution-based project you have a **Features** and **Packages** node. These work the same way they do in a solution-based project.
   2. **Content**, **Images** & **Pages**: These are actually SharePoint Project Items (SPI) that are Modules and will provision their contents to the respective folders in the app web that will be generated upon installing the app.
      1. **Content 🡪 App.css:** this is the Cascading Style Sheet used for the app.
      2. **Images 🡪 AppIcon.png:** this is the default image used for the app.
      3. **Pages 🡪 Default.aspx:** this is the default homepage for the app.
   3. **Scripts:** This is also a SPI Module that provisions its contents to the site. Because SharePoint-Hosted apps cannot use any server-side code, all business logic is handled in the client using JavaScript.
      1. **Scripts 🡪 \_references.js**: this file is not provisioned… it is a little trick file that Visual Studio uses to provide IntelliSense when writing JavaScript code. If you look inside this file you will triple commented references to JavaScript files which Visual Studio 2012 uses for IntelliSense.
      2. **Scripts 🡪 App.js**: this file is the default logic file for your app. It is referenced from the Default.aspx file. You do not have to use this file for your business logic… you can replace its contents or create your own.
      3. **Scripts 🡪 jquery[..].js**: because all logic is implemented in the client, the popular jQuery library is added to the project. The mimified and non-mimified versions of the library are include as well as the Visual Studio IntelliSense file (\*.vsdoc.js). You are free to replace this jQuery library with a more recent release if you like.
   4. **AppManifest.xml**: every app must have this file. It tells SharePoint the basic information it needs to know about the app such as:
      1. Name, Unique Identifier, Description & Version of the app.
      2. Icon for the app.
      3. Security configuration and permissions.
      4. Start page URL and any lifecycle event endpoints.



1. Examine the default SharePoint-Hosted app template:
   1. Using the **Solution Explorer** tool window, right-click the **Pages / Default.aspx** file and select **Open**.
      1. This file has a few JavaScript references in it that are added to the head part of the page using the ASP.NET content placeholder **PlaceHolderAdditionalPageHead**.
      2. There are references to the jQuery library and the **App.js** file.
      3. There is a reference to the **App.css** file as well.
   2. Using the **Solution Explorer** tool window, right-click the **Scripts / App.js** file and select **Open**.
      1. This file has four functions and a few globally scoped variables.
      2. The function **$(document).ready(function()){ … }** gets a reference to the client object model (CSOM) **ClientContext** object and then gets a reference to the current site.
      3. The **getUserName()** function is one that will usually be deleted from the project when you get more experience with SharePoint-Hosted apps. It uses the CSOM to get the name of the current user logged in.
      4. The last two functions are used as the success and failed callback when the CSOM request completes.
2. Update the app homepage:
   1. Using the **Solution Explorer** tool window, right-click the **Pages / Default.aspx** file and select **Open**.
   2. After the existing **<div>**, add the following markup:

<div id="displayDiv"></div>

<button onclick="hello();">Push me!</button>

1. Update the app script file:
   1. Using the **Solution Explorer** tool window, right-click the **Scripts / App.js** file and select **Open**.
   2. Add the following function to the bottom of the file that will be called when you click the button:

function hello() {

$get("displayDiv").innerHTML = "<p>Hello, Apps!</p>";

return false;

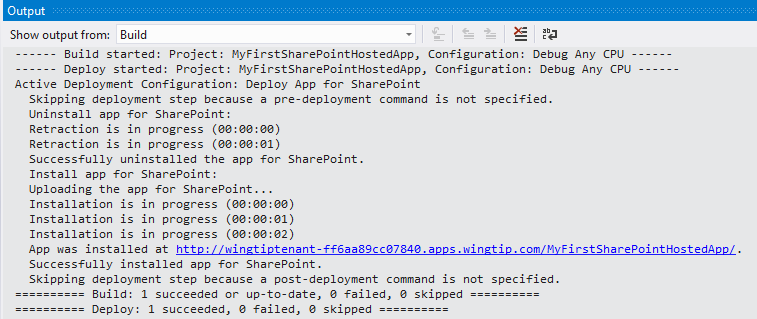
}

1. Save all changes: **File 🡪 Save All**.

This is the simplest SharePoint-Hosted app you can create with some business logic in it. Let’s just see how things turn out before we add some more stuff to it.

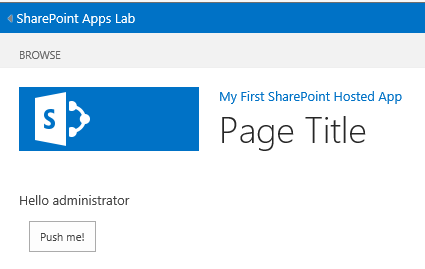
#### Build and Test the Project

1. Build and test your application by pressing **[F5]** or **Debug 🡪 Start Debugging**.
2. The installation process for an app will take a moment to complete. If you watch the lower-left corner of Visual Studio, it will tell you what it is currently doing. If you want more information, click the **Output** tab at the bottom of Visual Studio to see a log of what is going on (if the **Output** tab isn’t present, select the window from the menu in Visual Studio 2012: **View 🡪 Output**):



* 1. What you see in the screenshot is that the app was compiled first and then the installation process started. Visual Studio will write a message to the Output window every second while the app is being installed.

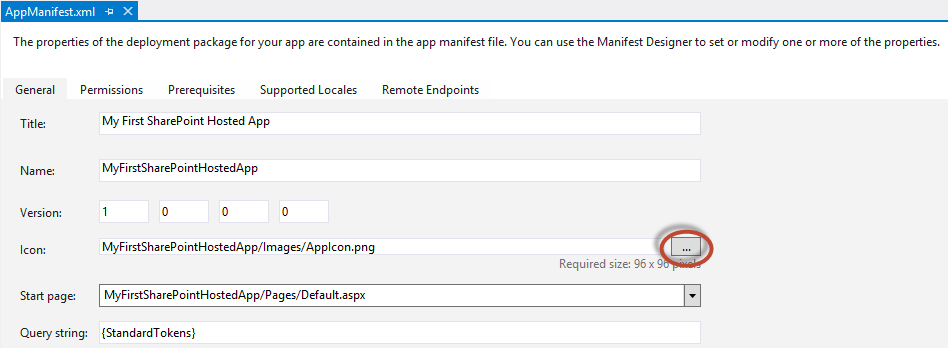
1. Once the solution has been deployed, Internet Explorer will launch and navigate to the app’s **default.aspx** page.
2. When the page loads, click the **Push me!** button to see your text get written to the page:



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

#### Customize the SharePoint-Hosted App

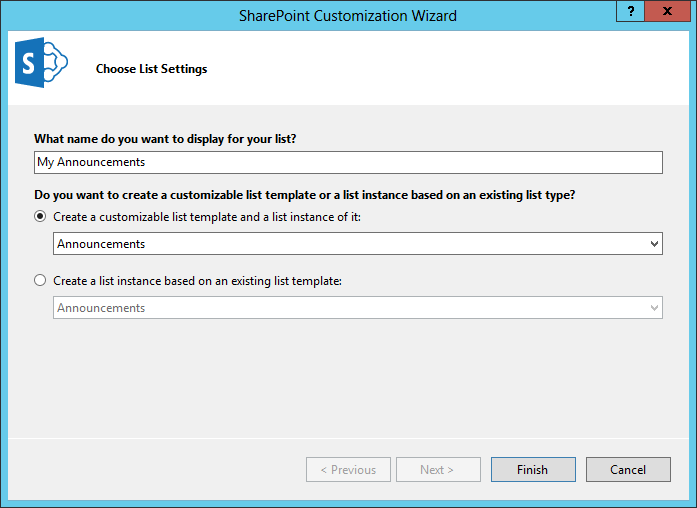
1. Use a new icon for the app:
   1. Using the **Solution Explorer** tool window, right-click the **AppManifest.xml** file and select **Open**.
   2. Click the **Browse** button for the **Icon**:



* 1. Select the **CPTAppIcon96x96.png** image found in the extra files associated with this course and click **OK**:

[..]\ExtraStudentFiles\Images\CPT

1. Add a list to the app:
   1. Using the **Solution Explorer** tool window, right-click the **MyFirstSharePointHostedApp** project and select **Add 🡪 New Item**.
   2. In the **Add New Item** dialog, select the **List** template found in the **Visual C# Items 🡪 Office / SharePoint** category.
      1. **Name:** MyAnnouncements
      2. Click **Add**
   3. In the **SharePoint Customization Wizard** dialog, use the following values to complete the form:
      1. **What name do you want to display for your list?** My Announcements
      2. **Do you want to create a customizable or non-customizable list based on an existing list type:** Create a customizable list based on: **Announcements**
   4. Click **Finish**.



1. Save all changes: **File 🡪 Save All**.
2. Open the file default.aspx and add the following HTML link to the bottom of the page.

<div>

<a href="../Lists/My Announcements">My Announcements</a>

</div>

#### Build and Test the Project

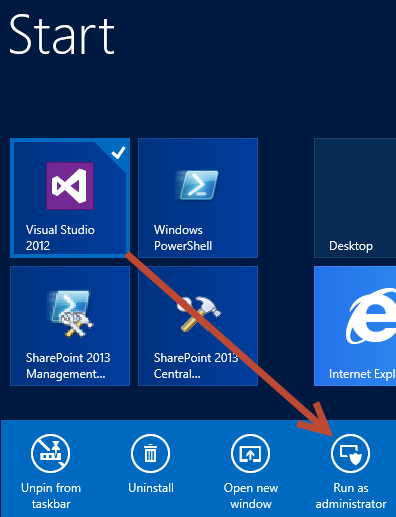
1. Build and test your application by pressing **[F5]** or **Debug 🡪 Start Debugging**.
2. Once the solution has been deployed, Internet Explorer will launch and navigate to the app’s homepage.
3. Notice there is nothing in the user interface of the app to link to the Announcements list except for the link that you added. Click that link to navigate to the Announcements list.
4. Close the browser to stop the debugger and go back to Visual Studio.

In this exercise you created a simple SharePoint-Hosted app and made some basic customizations to it. Later modules and labs will use this foundation to build on such as working with the CSOM and REST APIs, customizing the user interface and doing robust client-side development with additional permissions.

### Exercise 3: Create a Provider-Hosted App

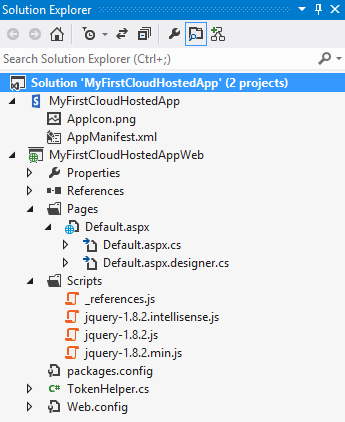
In this exercise you will create a new Provider-Hosted App.

1. Create a new project in Visual Studio 2012:
   1. Launch **Visual Studio 2012** as administrator:
      1. Windows Keyboard Key 🡪 Right click on the Visual Studio 2012 tile and select Run as administrator.



* 1. In Visual Studio select **File 🡪 New 🡪 Project**.
  2. In the **New Project** dialog:
     1. Find the **App for SharePoint 2013** template under the **Templates 🡪 Visual C# 🡪 Office / SharePoint 🡪 Apps** section.
     2. **Name**: MyFirstCloudHostedApp
     3. **Location:** [..]\DevApps\Exercises\Ex3  
        (Where [..] represents the location of the student files (e.g. c:\student\..)
     4. Uncheck the **Create directory for solution** checkbox
  3. Click **OK** to create the project.
  4. In the **New App for SharePoint** wizard, on the **Specify the App for SharePoint Settings** page, use the following values:
     1. **What is the name of your App for SharePoint?** My First Cloud Hosted App
     2. **What site do you want to use for debugging?** <http://appdev.wingtip.com>
     3. **How do you want to host your app for SharePoint?** Provider-hosted
  5. Click **Next**.
  6. In the **New App for SharePoint** wizard, on the **Configure authentication Settings** page, use the following value to complete the wizard:
     1. **How do you want your app to authenticate?** Use a client secret (requires a SharePoint farm connected to ACS)
  7. Click **Finish**.

1. Examine the default project setup for a SharePoint-Hosted app:
   1. The default Provider-Hosted app is very different from the SharePoint-Hosted app:



* 1. Notice the **MyFirstCloudHostedApp** project contains only two files: the **AppManifest.xml** and **AppIcon.png**.
     1. This effectively means the app will do nothing in SharePoint; you simply need to register it.
  2. There is a new project, **MyFirstCloudHostedAppWeb**, which will serve the role of the remote web. It is a standard ASP.NET Web application, but it contains a little extra stuff in it:
     1. **TokenHelper.cs**: This is a code file provided by Microsoft to make it easier to obtain the user identity, the OAuth token or the token provided by highly trusted apps. You will ignore this for now.
     2. **Default.aspx.cs**: The code behind file for the page contains logic to call back into SharePoint to obtain the title of the host web. The code is written to assume this app will use OAuth authentication.
     3. **Scripts**: A common folder to place JavaScript files.

#### Update Security Configuration for the App

By default a Provider-Hosted app is expecting to use either OAuth security or S2S, also known as server-to-server or a highly trusted app. For this lab, let’s remove security from the complication of building our first Provider-Hosted app:

1. First, modify the security configuration of the app to use internal security and not OAuth or a special S2S token:
   1. Using the **Solution Explorer** tool window, within the **MyFirstCloudHostedApp** project, right-click the **AppManifest.xml** file and select **View Code**.
   2. Locate the **<AppPrincipal>** node.
   3. Replace the contents with **<Internal />** so it looks like the following markup:

<AppPrincipal>

<Internal />

</AppPrincipal>

#### Code the Provider-Hosted App

Update the homepage for the application to run something using server-side code, something that isn’t possible in a SharePoint-Hosted app.

1. Using the **Solution Explorer** tool window, within the **MyFirstCloudHostedAppWeb** project, right-click the **Default.aspx** file and select **Open**.
   1. Replace the existing **<div>** on the page with a ASP.NET literal control so the body of the page looks like the following markup:

<body>

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

<asp:Literal ID="Message" runat="server" />

<p><asp:HyperLink ID="HostWebLink" runat="server" /></p>

</form>

</body>

1. Using the **Solution Explorer** tool window, within the **MyFirstCloudHostedAppWeb** project, right-click the **Default.aspx 🡪 Default.aspx.cs** file and select **Open**.
   1. Replace the contents of the **Page\_Load()** method with the following code:

this.Message.Text = "My first SharePoint Provider-Hosted App!";

var hostWeb = Page.Request["SPHostUrl"];

this.HostWebLink.NavigateUrl = hostWeb;

this.HostWebLink.Text = "&raquo; Go back to host web in SharePoint...";

1. Save all changes: **File 🡪 Save All**.

#### Build and Test the Project

1. Build and test your application by pressing **[F5]** or **Debug 🡪 Start Debugging**.
2. Visual Studio 2012 may prompt you with a Security Alert to trust a self-signed certificate. You are not using a certificate in this solution, so click **Yes**—and click **Yes** again to get past the extra confirmation prompt—to continue.
3. Once the solution has been deployed, Internet Explorer will launch and navigate to the start page of the app, located in the remote web.
   1. Notice when the page loads it is just a plain white page with the text you added and a link back to the hosting site.
   2. Close the browser to stop the debugger and go back to Visual Studio.

In this exercise you created a simple SharePoint Provider-Hosted app. Much like Exercise 2, in this exercise you didn’t do much beyond the simplest app possible. In later labs you will build on this foundation to add more capabilities to the app.