## Getting Started with SharePoint App Development

**Lab Time**: 45 minutes

**Lab Folder**: C:\Student\Modules\AppModel\Lab

**Lab Overview**: In this lab you will get hands-on experience working with the new SharePoint App model. Through the exercises in this lab you will learn how to create and test a SharePoint-hosted app as well as a provider-hosted app.

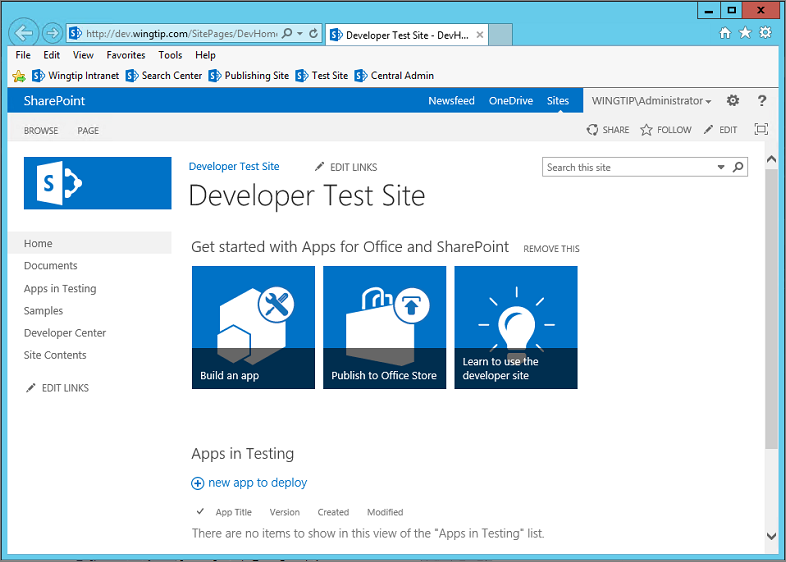
### Exercise 1: Creating a New Developer Site for Testing

In this exercise you will prepare your development environment by creating a new Developer site at http://dev.wingtip.com.

1. Create 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 **SetupLab.ps1** and select **Run with PowerShell**. This file can be found in the files associated with this lab:  
         (Note: In order to run PowerShell Scripts in this environment, you may be prompted for an Execution Policy Change. If prompted, type **Y** and press **Enter.**)

C:\Student\Modules\AppModel\Lab\SetupLab.ps1

* 1. When the script completes, it will launch a new browser and navigate to the new Developer site at **http://dev.wingtip.com**.



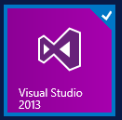
* 1. Close the PowerShell console window.

You have now completed the first step by creating a new test site for SharePoint app development.

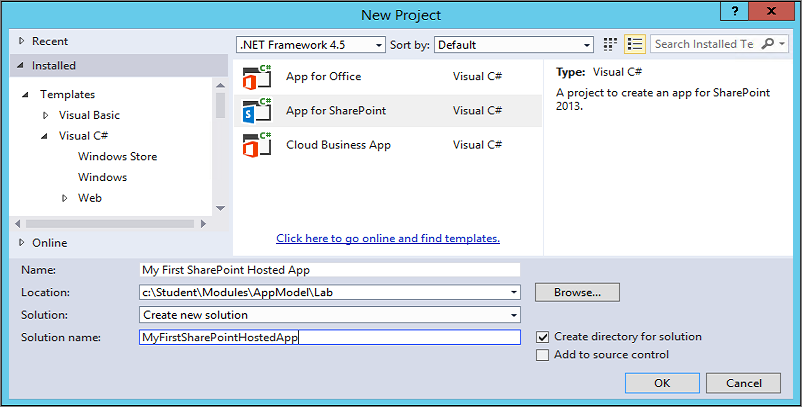
### Exercise 2: Creating and Debugging a SharePoint-Hosted App

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

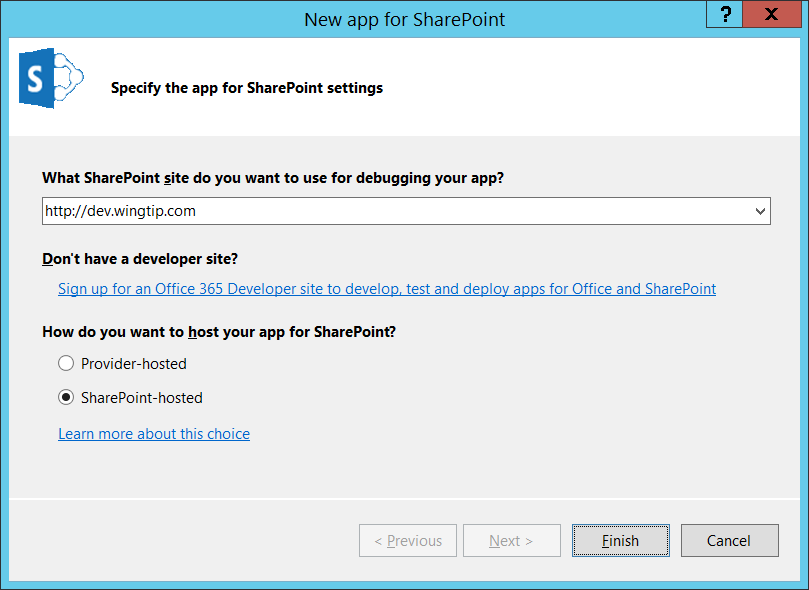
1. Create a new project in Visual Studio 2013:
   1. Launch **Visual Studio 2013** as administrator:
      1. Windows Keyboard Key 🡪 Right click on the Visual Studio 2013 tile and select **Run as administrator**.



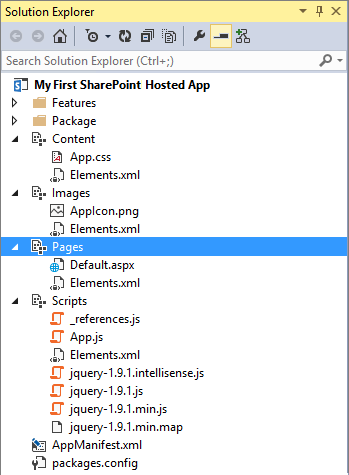
* 1. In Visual Studio select **File 🡪 New 🡪 Project**.
  2. In the **New Project** dialog:
     1. Find the **App for SharePoint 2013** template under the **Templates 🡪 Visual C# 🡪 Office / SharePoint 🡪 Apps** section.
     2. **Name**: My First SharePoint Hosted App **(Note:** the name is used for the **Project Name** and also for the **default App Title**. Either put spaces in the name or after creating the project edit the **AppManifest.xml** file and add spaces to the Title field.)
     3. **Location:** C:\Student\Modules\AppModel\Lab
     4. **Solution name:** MyFirstSharePointHostedApp
     5. Click **OK.**



* 1. In the **New App for SharePoint** wizard, use the following values to complete the wizard and click **Finish**.
     1. **What site do you want to use for debugging?** <http://dev.wingtip.com>
     2. **How do you want to host your app for SharePoint?** SharePoint-hosted



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 notice triple commented references to JavaScript files which Visual Studio 2013 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 minified and non-minified 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 about the app such as:
      1. Name, Product ID, App Version Number and minimum version for the SharePoint host environment.
      2. Security configuration and permissions.
      3. App Title to display on app launcher tile on Site Contents page of the host web.
      4. The URL of the app’s start page.



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 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:

<input type="button" value="Push Me" onclick="hello();" />

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

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>";

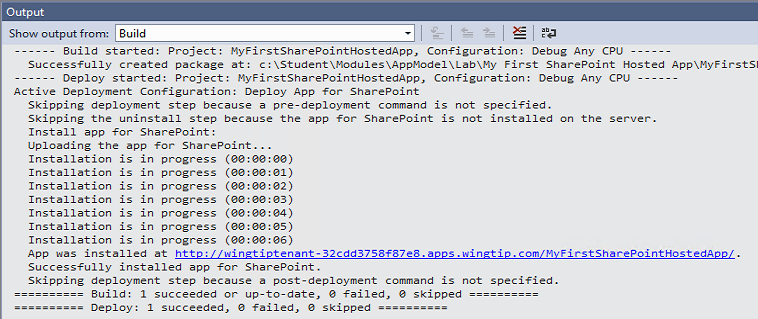
}

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

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

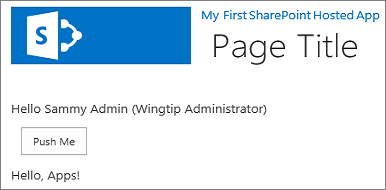
#### Build and Test the Project

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 2013: **View 🡪 Output**):



* 1. What you see in the screenshot is 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.

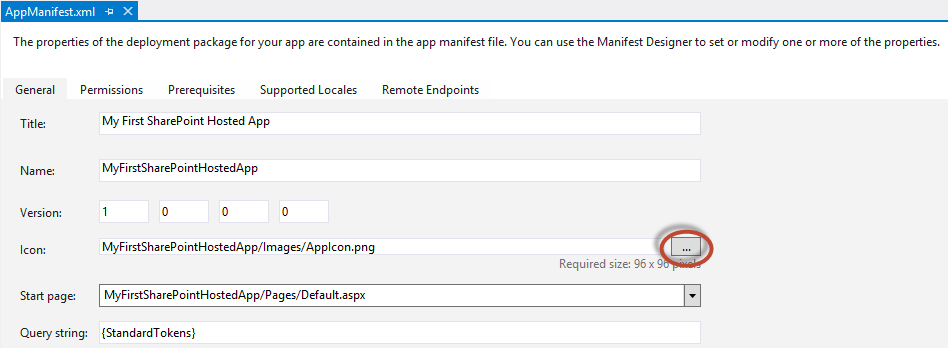
1. Once the solution has been deployed, Internet Explorer will launch and navigate to the app’s **default.aspx** page.
2. When the page loads, click the **Push me!** button to see your text get written to the page:



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

#### Customize the SharePoint-Hosted App

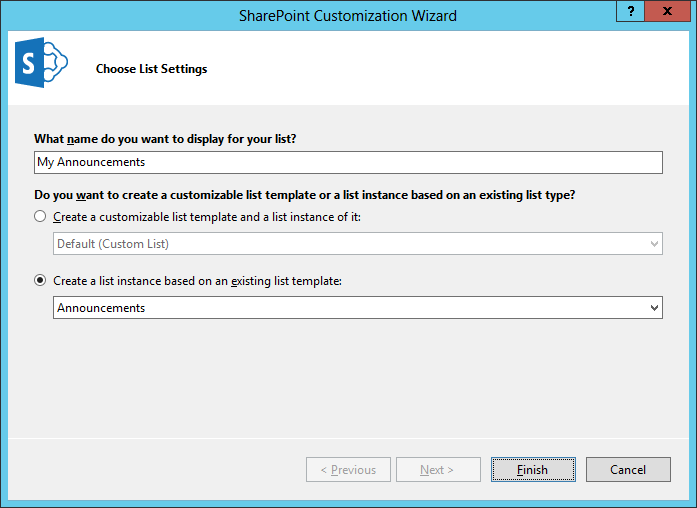
1. Use a new icon for the app:
   1. Using the **Solution Explorer** tool window, right-click the **AppManifest.xml** file and select **Open**.
   2. Click the **Browse** button  for the **Icon**:



* 1. Select the **MyAppIcon.png** image found in the **StarterFiles** folder for this lab and click **OK**:

C:\Student\Modules\AppModel\Lab\StarterFiles\MyAppIcon.png

1. Add a list to the app:
   1. Using the **Solution Explorer** tool window, right-click the **My First SharePoint Hosted App** 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 and click **Finish**:
      1. **What name do you want to display for your list?** My Announcements
      2. **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**



1. Save all changes: **File 🡪 Save All**.
2. Open the file default.aspx and add the following HTML link to after the <div id=”displayDiv”></div> ta.

<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. At the top of the page click the **Developer Test Site** link in the breadcrumb navigation to navigate out of the App back to the site.
5. On the Sites Quick Launch navigation (on the left side of the screen) click on **Site Contents**.
6. On the Site Contents page notice the icon for the app we just deployed My First SharePoint Hosted App.



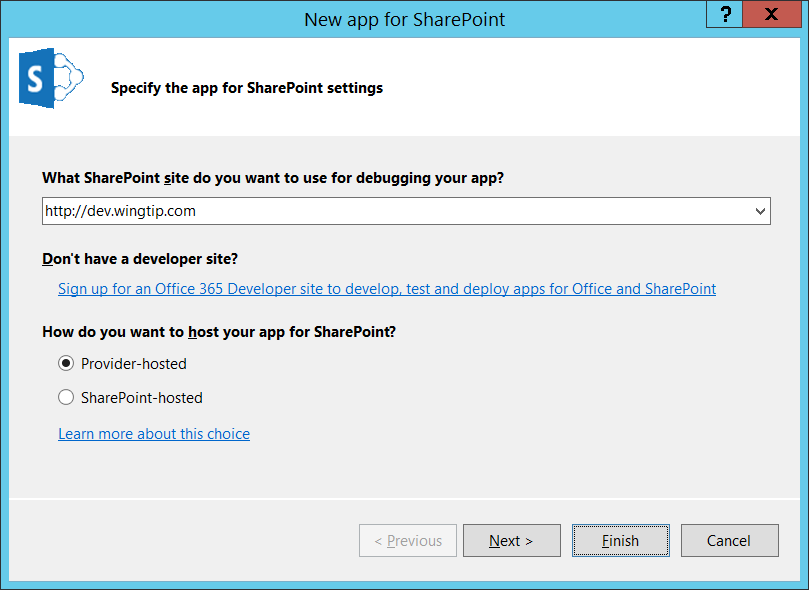
1. Close the browser to stop the debugger and go back to Visual Studio.

In this exercise you created a simple SharePoint-Hosted app and made some basic customizations to it. Later modules and labs will 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 and Debugging a Provider-Hosted App

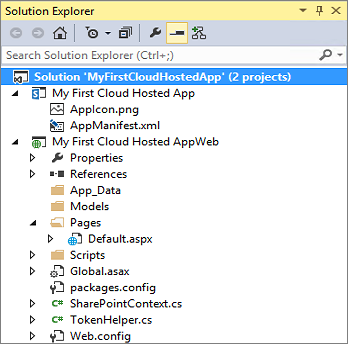
In this exercise you will create and test a very simple Provider-Hosted App. This will give you opportunity to observe the basic differences between developing SharePoint-hosted apps and cloud-hosted apps using Visual Studio 2013.

1. Create a new project in Visual Studio 2013:
   1. Launch **Visual Studio 2013** as administrator if it is not already running.
   2. In Visual Studio select **File 🡪 New 🡪 Project**.
   3. In the **New Project** dialog:
      1. Find the **App for SharePoint 2013** template under the **Templates 🡪 Visual C# 🡪 Office / SharePoint 🡪 Apps** section.
      2. **Name**: My First Cloud Hosted App
      3. **Location:** C:\Student\Modules\AppModel\Lab\
      4. **Solution name:** MyFirstCloudHostedApp
      5. Click **OK**
   4. On the **Specify the App for SharePoint Settings** page, use the following values and click **Next**:
      1. **What site do you want to use for debugging?** <http://dev.wingtip.com>
      2. **How do you want to host your app for SharePoint?** Provider-hosted.



* 1. In the **New App for SharePoint** wizard, on the **Specify the web project type** page select the **ASP.NET Web Forms Application** setting and click **Next >**
  2. In the **New App for SharePoint** wizard, on the **Configure authentication Settings** page, use the following value to complete the wizard and click **Finish**:
     1. **How do you want your app to authenticate?** Use Windows Azure Access Control Service (for SharePoint cloud apps)

1. Examine the default project setup for a SharePoint-Hosted app:
   1. As you can see, the Visual Studio solution created for a Provider-Hosted app has two projects and is very different from the Visual Studio solution for a SharePoint-Hosted app which has only one project:



* 1. Notice the **My First Cloud Hosted App** project contains only two files: the **AppManifest.xml** and **AppIcon.png**.
     1. This effectively means the app will not install any resources into the SharePoint host such as pages. This project only contains app metadata and an image file that get added to the SharePoint host when the app gets installed.
  2. There is a new project, **My First Cloud Hosted AppWeb**, 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**: (Inside the **Pages** folder expand out the **Default.aspx** file to see this) 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. **SharePointContext.cs**: This is a code file provided by Microsoft to encapsulate all the information from SharePoint. You will ignore this for now.
     4. **Scripts**: A common folder to place JavaScript files.

#### Update Security Configuration for the App

By default a Provider-Hosted app is expecting to use external authentication with either OAuth or S2S which are topics covered later in the course. For this lab, you will disable the configuration for external authentication to eliminate security requirements which would complicate building and testing our first Provider-Hosted app:

1. First, modify the app manifest to use internal app authentication:
   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 app 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. Delete the contents of the **Page\_PreInit()** method (Note: this is necessary, along with replacing the contents of the Page\_Load() method, as we are using Internal App Authentication)
2. 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 2013 may prompt you with a Security Alert to trust a self-signed certificate. You are not using a certificate in this solution, so just click **Yes** (and 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 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. As in the last exercise, you didn’t do much in this exercise beyond creating and testing the simplest cloud-hosted app possible. In later labs you will build on this foundation to add more capabilities to SharePoint apps.