## Developing and Debugging a Provider-hosted App

**Lab Time**: 60 minutes

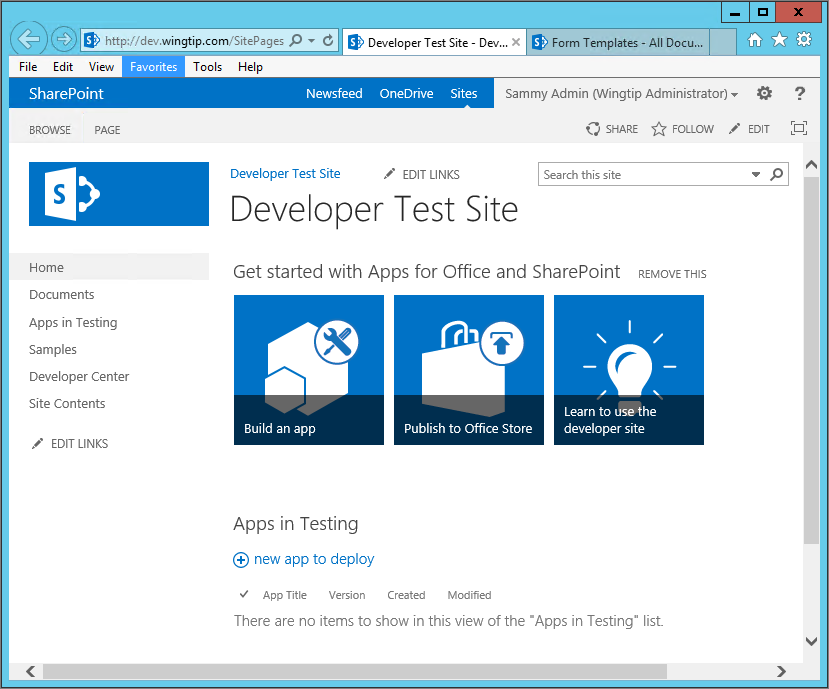
**Lab Folder**: C:\Student\Modules\ProviderHostedApps\Labs

**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 an Provider-hosted App Project in Visual Studio 2013

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

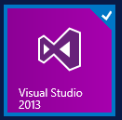
1. Make sure your developer test site is up and running so you have a place to debug the app you are going to create.
   1. Using the browser, navigate to the test site at <http://dev.wingtip.com>.



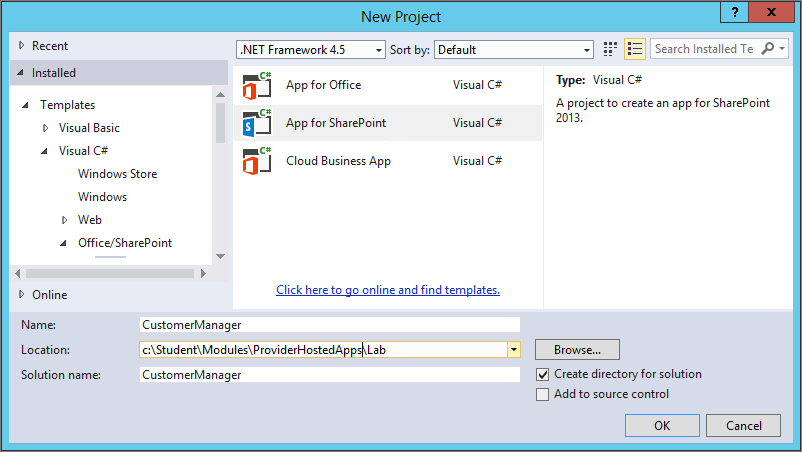
* 1. If this site does not yet exist, create it by running the PowerShell script named **CreateSharePointDeveloperSite.ps1** which is located at the following path. Also, keep in mind that you can also run the script to delete and recreate the developer site to return it to its original state whenever you would like.

C:\Student\Setup\CreateSharePointDeveloperSite.ps1

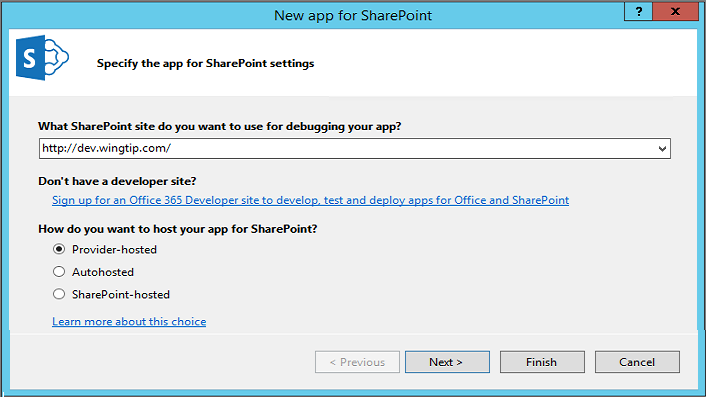
1. Launch **Visual Studio 2013** as administrator:
   1. Windows Keyboard Key 🡪 Right click on the **Visual Studio 2013** tile and select **Run as administrator**.



1. Create a new SharePoint provider-hosted app project in Visual Studio 2013.
   1. In Visual Studio select **File 🡪 New 🡪 Project**.
   2. Fill out the **New Project** dialog as follows…
      1. Find the **App for SharePoint 2013** template under the **Templates 🡪 Visual C# 🡪 Office / SharePoint 🡪 Apps** section.
      2. **Name**: CustomerManager
      3. **Location:** C:\Student\Modules\ProviderHostedApps\Lab
      4. Click **OK.**

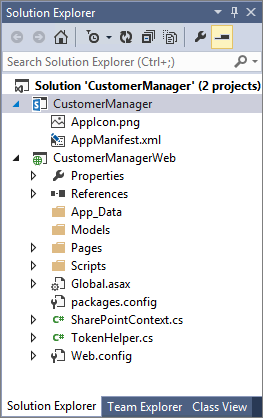


* 1. In the **New App for SharePoint** wizard, use the following values to complete the wizard and click **Finish**.
     1. What site do you want to use for debugging? <http://dev.wingtip.com>
     2. How do you want to host your app for SharePoint? Provider-hosted
     3. Click **Finish**.



When you click the **Finish** button instead of the **Next** button, you are accepting the defaults for the remaining settings. For example, the type of web application project will be **ASP.NET Web Forms Application**; and the type of Authentication will be **Use Windows Azure Access Control Service (for SharePoint cloud apps)**.

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



* 1. The top project named **CustomerManager** is the project for the app itself. This project contains the **AppManifest.xml** file.
  2. 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.
  3. The **Web Project** property of the top project named **CustomerManager** references the **CustomerManagerWeb** project.

1. Change the **CustomerManager** app project’s authentication method to use internal authentication:
   1. In the project **CustomerManager** project, locate the **AppManifest.xml** file.
   2. Open the **AppManifest.xml** file in Code View by right-clicking it and selecting **View Code**.
   3. First find the **<Title>** element. Add a space in the Title so that the name looks like this **Customer Manager**
   4. Now find the **<AppPrincipal>** element. Remove the **<RemoteWebApplication>** element and replace it with **<Internal/>** so the **<AppPrincipal>** element now looks like this:

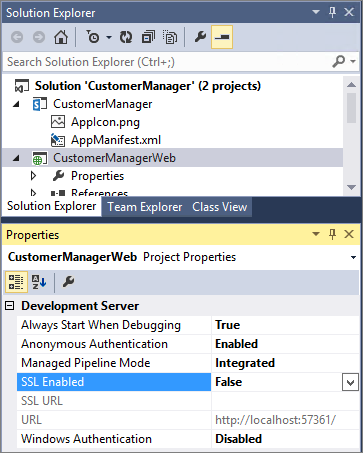
<AppPrincipal>

<Internal/>

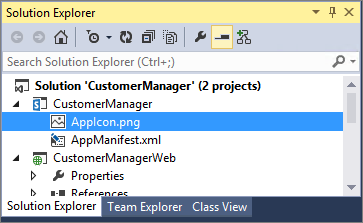
</AppPrincipal>

* 1. Save and close the **AppManifest.xml** file

1. Configure the projects to disable the use of SSL so everything happens over HTTP rather than HTTPS:
   1. Select the project **CustomerManagerWeb** in the **Solution Explorer** tool window.
   2. Look in the **Properties** tool window. Look at the **SSL Enabled** property and make sure it is set to **False**.



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



* 1. 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…**)

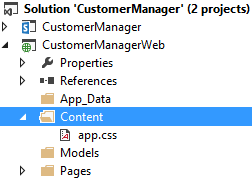
C:\Student\Modules\ProviderHostedApps\Labs\StarterFiles\AppIcon.png

* 1. Open the **AppManifest.xml** file by double clicking on it in the Solution Explorer
  2. Click the button to the right of the Icon: Text box (…)
  3. Select the **AppIcon.png** file located in the default location.
  4. Save and close the **AppManifest.xml** file

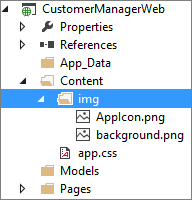
1. Add CSS styles and image files to the project.
   1. Within the **CustomerManagerWeb** project, create a new folder named **Content** at the root of the project.
   2. Using the Windows Explorer, locate the CSS file named app.css at the following path.

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

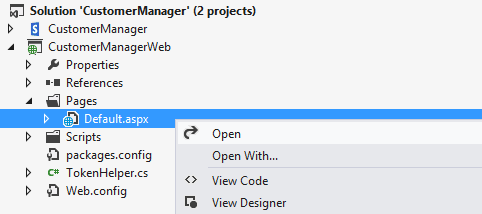
* 1. Add a copy of the app.css file into the **Content** folder of the **CustomerManagerWeb** project. When you have completed this step, the structure of your project should look like the one in the following screenshot.



* 1. 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**.



1. Modify the HTML for the app’s start page:
   1. Within the **CustomerManagerWeb** project, right-click the **Pages\Default.aspx** file and select **Open**.



* 1. 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" %>

* 1. Replace the existing DOCTYPE declaration with the simpler DOCTYPE declaration used in HTML5.

<!DOCTYPE html>

* 1. Modify the **title** element within the **head** section to contain the text of **Customer Manager**.

<head runat="server">

<title>Customer Manager</title>

</head>

* 1. 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>

* 1. 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>

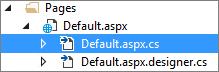
</div>

</form>

</body>

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

1. Remove the starter C# code that Visual Studio has added to **Default.aspx.cs**.
   1. Inside the **Pages** folder of the **CustomerManagerWeb** project, open the C# source files named **Default.aspx.cs**.



* 1. 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);

}

}

}

}

* 1. 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) {

}

}

}

* 1. 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

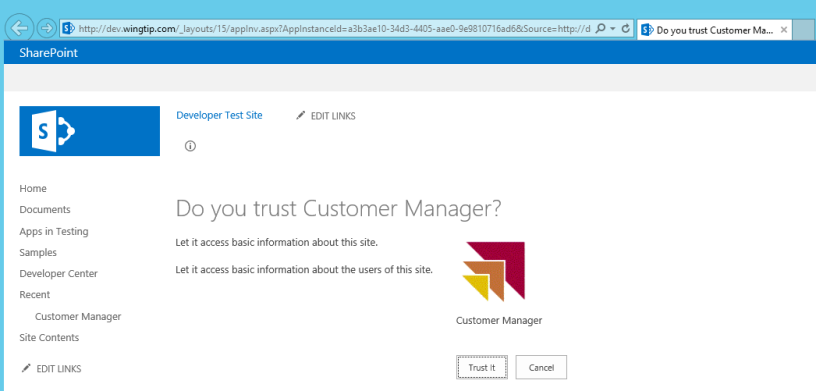
pageContent.Controls.Add( new LiteralControl("Hello from server-side C# code"));

}

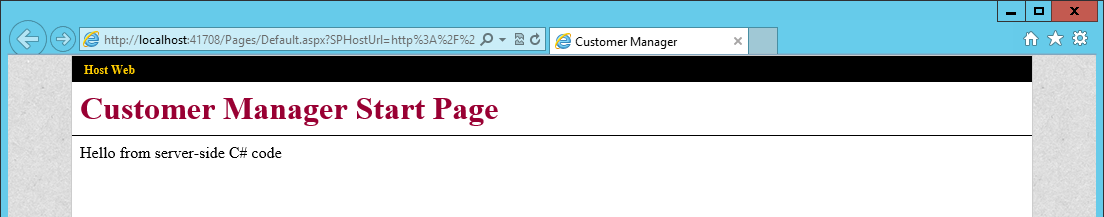
* 1. When you are done, save your changes and close **Default.aspx.cs**.

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

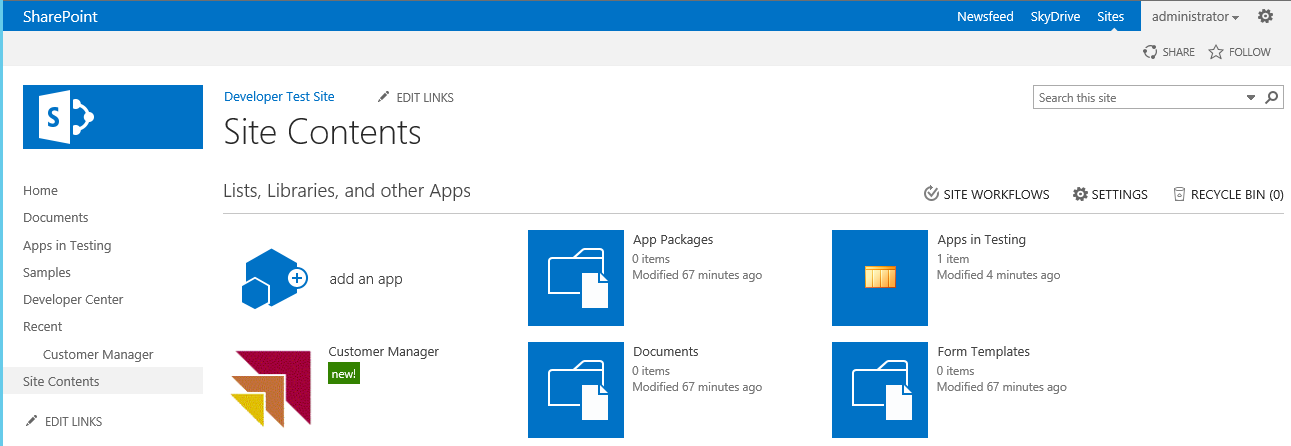
1. Test your application using the Visual Studio debugger.
   1. Begin a debugging session in Visual Studio by pressing the **{F5}** key or running the command **Debug 🡪 Start Debugging**.
   2. When Visual Studio installs the app, SharePoint might prompt you with a page asking you whether you trust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



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



* 1. 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](http://providerhostedapp.wingtip.com).
  2. 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.

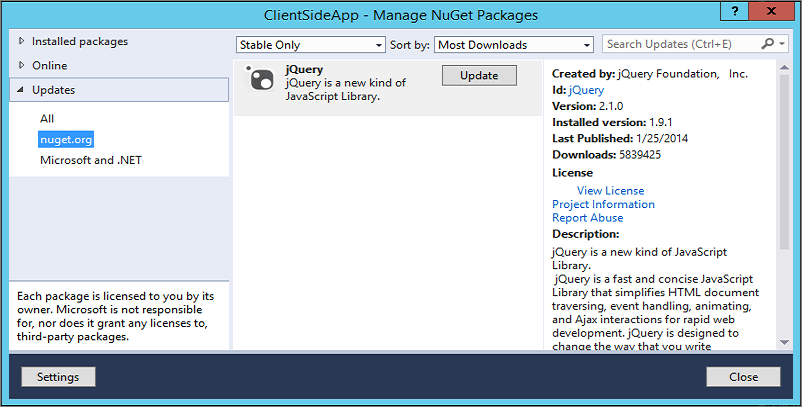


1. Close the browser to stop the debugger and go back to Visual Studio.

### 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.
   1. Right-click on the **CustomerManagerWeb** project and click **Manage NuGet Packages…**.
   2. Select **Updates** on the left-hand side of the **Manage NuGet Packages** dialog.
   3. Select the **jQuery** library and then click the **Update** button.

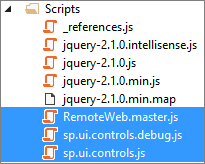


Note that the jQuery library version number you see might be more recent than the one shown in the previous screenshot.

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

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

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



1. Next, add a master page to the **Pages** folder.
   1. Right-click the **Pages** folder and select **Add 🡪 Existing Item**.
   2. In the **Add Existing Item** dialog, browse to the **StarterFiles** folder at the following path:

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

* 1. Select the **RemoteWeb.master** file and click **Add**

1. Modify the master page to link to CCS and JavaScript files.
   1. 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>

* 1. 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>

* 1. Add a script link for the jQuery library.
  2. Add a script link to the JavaScript library with the Chrome Control which is named **sp.ui.controls.js**.
  3. 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>

1. Open **RemoteWeb.master.js.**
   1. 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);

* 1. This JavaScript code in n **RemoteWeb.master.js** is written to go through these steps
     1. Determine the URL back to host web
     2. Create a settings object used to initialize Chrome Control
     3. Create the Chrome Control instance
     4. Call **setVisible** method to display on the hosting page
  2. Once you have looked at the JavaScript code and understand it, close **RemoteWeb.master.js**.

1. Add a few page files to the **Pages** **folder** to create the multi-page user interface experience:
   1. Create a new start page named **start.aspx**:
      1. Right-click the **Pages** folder and select **Add 🡪 New Item**.
      2. 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**
      3. When promoted to select a master page, select **RemoteWeb.master** in the **Pages** folder. Click **OK**
      4. 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>

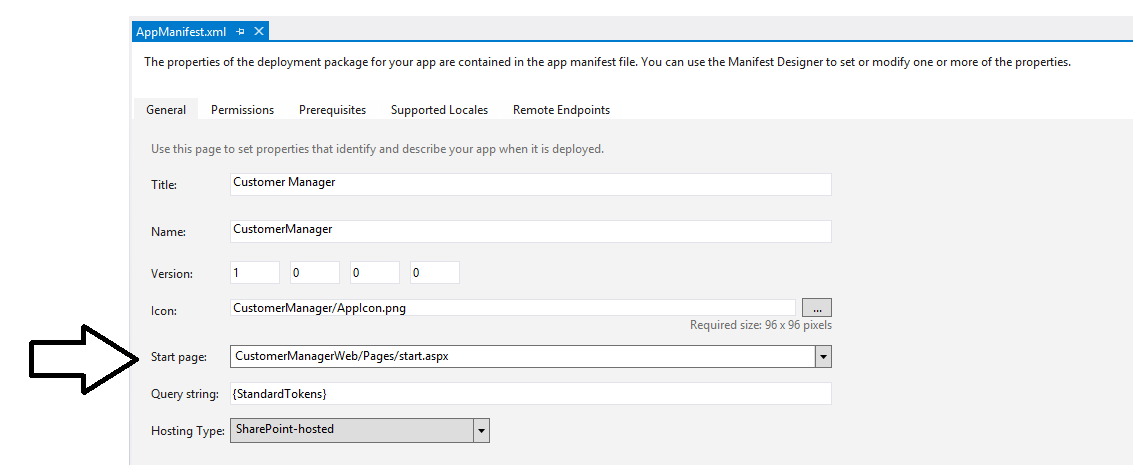
* 1. Repeat the previous step to create the following pages & include the appropriate **<h1>** tag:
     1. About.aspx
     2. Contact.aspx
     3. Help.aspx

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**)

1. 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"

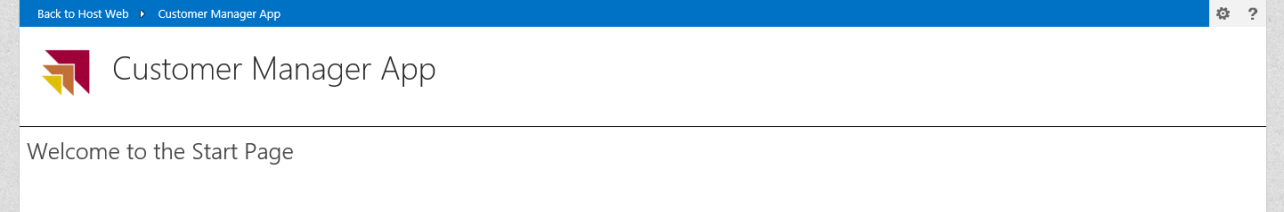
1. Now, change the start page of the app to use one of the new pages you just created:
   1. Using the **Solution Explorer** tool window, within the **CustomerManager** project, right-click the **AppManifest.xml** file and select **Open**.
   2. On the **General** tab of the app manifest designer, locate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



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

#### Build and Test the Project

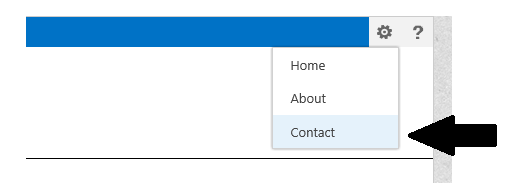
1. Build and test your application by pressing **[F5]** or **Debug 🡪 Start Debugging**.
2. Once the solution has been deployed, Internet Explorer will launch and navigate to the **start.aspx** page of the remote web.



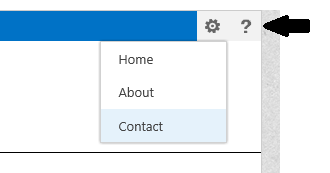
1. Now it is time for experiment by moving from page to page to ensure that all pages in this app have a similar look and feel:
   1. Locate the Chrome Controls navigation menu which is displays as a gear icon located in the top right corner of the page just to the left of the Help icon.



* 1. 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.



* 1. Click on the Help link and make sure you can navigate to the Help page as well.

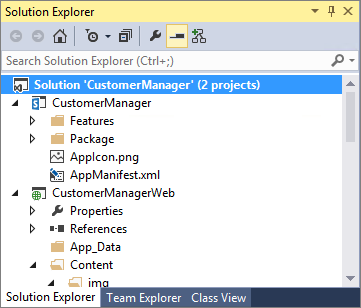


1. Close the browser to stop the debugger and go back to Visual Studio.
2. Leave Visual Studio and the current project open because you will continue working on it in the next exercise.

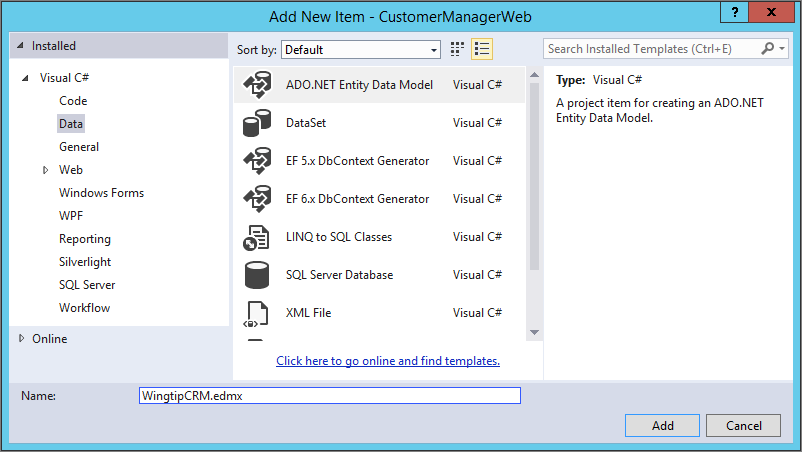
### Exercise 3: Interacting with Data in a Provider Hosted App

In this exercise you will modify your Provider Hosted App to display and interact with data housed in a SQL Server Database.   
**Note**: This lab assumes you have already completed the previous lab on REST where you created the SQL Server database named **WingtipCRM**. If you have not already completed the REST lab, you will not have the database required to continue with this lab. If this is the case, you should stop in this lab and go back to the REST lab and complete **Exercise 1: Create and Populate the Wingtip CRM Database in SQL Server**. Once you have completed this exercise and created the **WingtipCRM** database, you can continue with this lab.

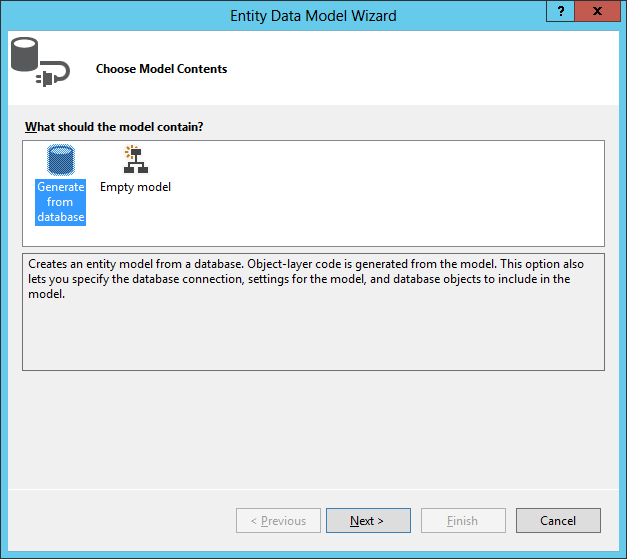
1. At this point, you should still have the **CustomerManager** solution for your provider-hosted app opened in Visual Studio.



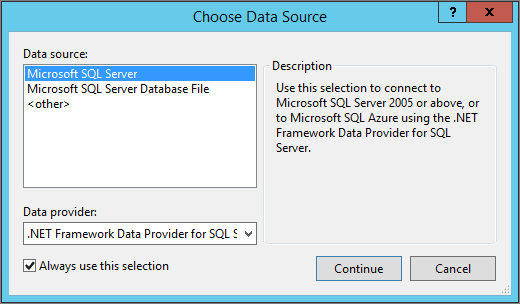
1. Add an ADO.NET Entity Model for the **WingtipCRM** Database.
   1. In the **CustomerManagerWeb** project node, right-click on the **Models** folder and click **Add >> New Item**.
   2. In the left side of the **Add New Item** dialog, select **Installed >>** **Visual C# >> Data**.
   3. Select the project item template named **ADO.NET Entity Data Model**.
   4. Enter a **Name** of **WingtipCRM.edmx**.
   5. Click the **Add** button on the bottom right to begin the process of creating the new project item.



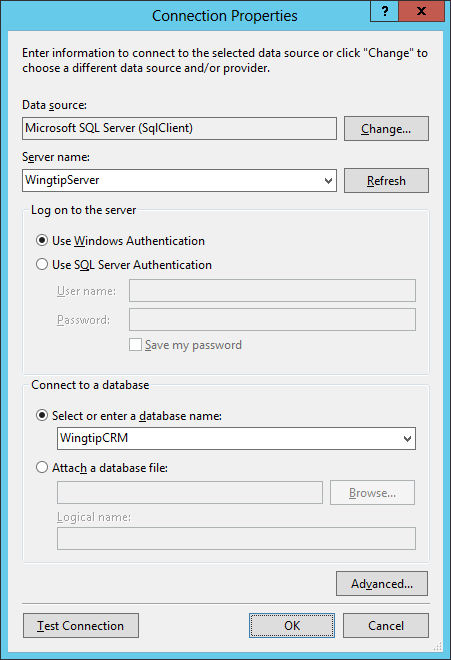
* 1. When you are prompted by **Choose Model Contents** page of the **Entity Data Model Wizard** dialog, select the option **Generate from database** and click **Next**.



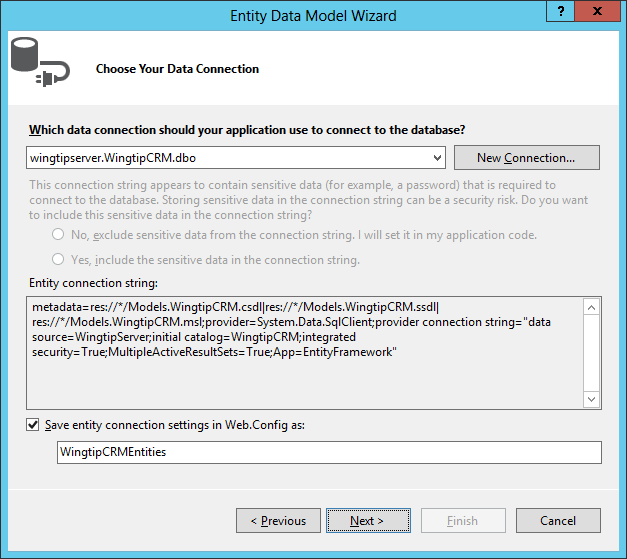
* 1. On the **Choose Your Data Connection** page of the **Entity Data Model Wizard**, you are prompted to select or create a new connection. Click the **New Connection…** button to display the **Connection Properties** dialog.
  2. If you are prompted by the **Choose Data Source** dialog, select **Microsoft SQL Server** and then click **Continue**.



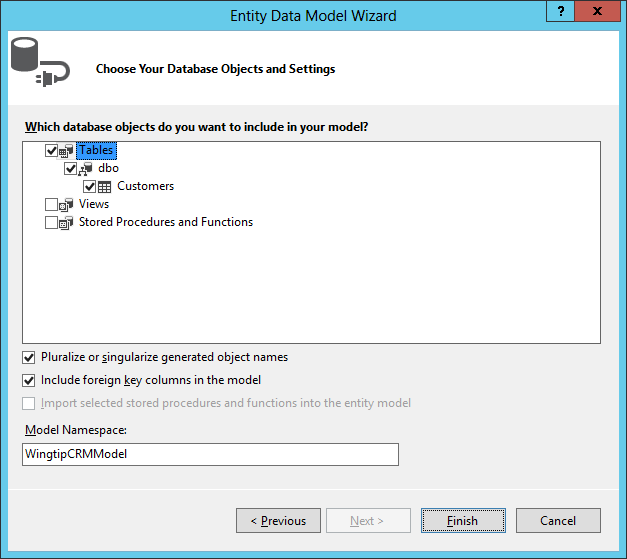
* 1. Next, you should see the **Connection Properties** dialog. Follow these steps to fill in the required information.
     1. Enter a **Server name** of **WingtipServer**.
     2. Enter a database name of **WingtipCRM**.
     3. Once the **Connection Properties** dialog looks like the following screenshot, click the **OK** button to save the connection information and return to the **Choose Your Data Connection** page of the **Entity Data Model Wizard**.



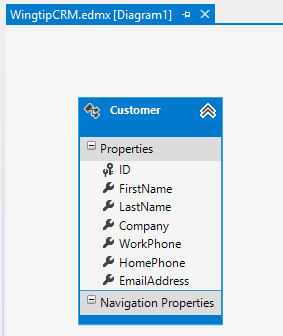
* 1. On the **Choose Your Data Connection** page of the **Entity Data Model Wizard,** leave the bottom of the page with the default settings as shown in the following screenshot so that the connection information is stored in **web.config** file under the name of **WingtipCRMEntities**. Click **Next** to continue to the **Choose Your Database Objects and Settings** page.



* 1. On the **Choose Your Version** for the Entity Framework page select **Entity Framework 5.0** and click **Next**
  2. On the **Choose Your Database Objects and Settings** page, expand the **Tables** node and select the **Customers** table. Leave the other settings on the page with their default settings as shown in the following screenshot. Click the Finish button to complete the task of create the entity data model.

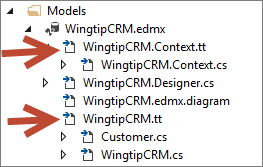


* 1. If Visual Studio prompts you regarding a Security Warning or whether it is OK to update the **web.config** file, select **OK** or **Yes** as appropriate.
  2. Once the Entity Data Model Wizard has finished its work, it will then display a Visual Studio designer that provides a visual representation of the new entity data model which contains a single entity named **Customer**.



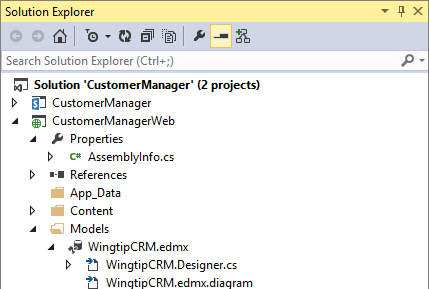
* 1. After you have examined the **Customer** entity and its properties in the visual designer, close **WingtipCRM.edmx**.

1. There are a few issues with the Entity Framework and the EntityDataSource control in Visual Studio 2013. Entity Framework 6.0 does not work at all with the EntityDataSource Control currently and Entity Framework 5.0 needs to be “tweaked” to allow it to work with the EntityDataSource control:
   1. In **Solution Explorer** expand the **Models** folder and **delete** the two files with **.tt** extensions as shown in the image below.

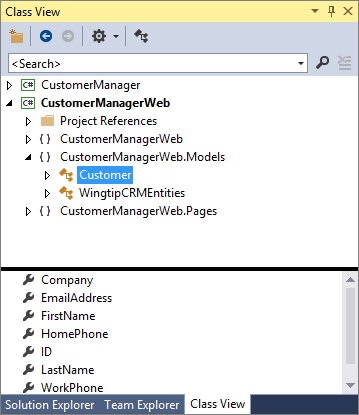


* 1. Click back into the Visual Studio Designer area for the **WingtipCRM.edmx [Diagram]**. Underneath the **Properties** window for **WingtipCRMModel** (on the right hand side of the screen) set the Code Generation Strategy From T4 to **Legacy ObjectContext**
  2. Now using the Visual Studio **BUILD** menu select **Rebuild Solution**.
  3. We have just completed configuring the Entity Framework to work with the EntityDataSource Control.

1. Use the Solution Explorer to examine the new files that the Entity Data Model Wizard added to the **WingtipCRM** project.
   1. You should be able to see several files have been added the project to support the new entity data model.



1. Use Class View to see what C# classes have been created.
   1. Switch from the Solution Explorer over to Class View.
   2. Expand the node with the **CustomerManagerWeb.Models** namespace.
   3. Verify that there is an entity model class named **Customer** which is associated with the **Customers** table in SQL Server.
   4. Verify that there is second class named **WingtipCRMEntities**.



1. Run the project in the Visual Studio debugger once to build out all the code generated by the Entity Framework. This step is required so that later steps in this lab work correctly.
   1. Press the **F5** key to begin a debugging session.
   2. The Customer Manager app should install and then you should be directed to the app's start page.
   3. Close the browser.
   4. Return to Visual Studio and ensure that the debugging session has ended.

Next, you will build a user interface in the start page using an ASP.NET **GridView** control and an **EntityDataSource** control.

1. In the **Solution Explorer** expand **CustomerManagerWeb** project and expand the **Pages** folder.
   1. Double click on the **start.aspx** page to open this page.
   2. Update the contents of the <h2> tag to **Customer List**.
   3. On the Source view place a <div></div> tag after the </h2> closing tag on a new line.

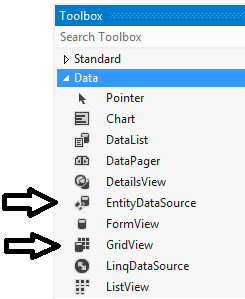
<asp:Content ContentPlaceHolderID="PlaceholderMain" runat="server">

<h2>Customer List</h2>

<div></div>

</asp:Content>

* 1. In the **Data** section of the Visual Studio toolbox, locate the **GridView** control and the **EntityDataSource** control.



* 1. Drag and drop a GridView control and a EntityDataSource control into the div tag you created.

<asp:Content ID="Content1" ContentPlaceHolderID="PlaceholderMain" runat="server">

<h2>Customer List</h2>

<div>

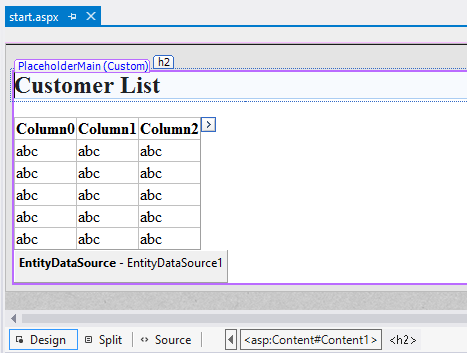
<asp:GridView ID="GridView1" runat="server"></asp:GridView>

<asp:EntityDataSource ID="EntityDataSource1" runat="server"></asp:EntityDataSource>

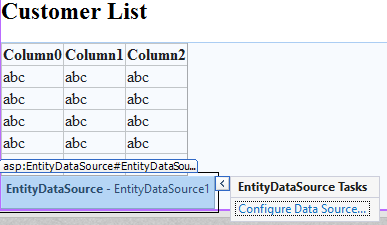
</div>

</asp:Content>

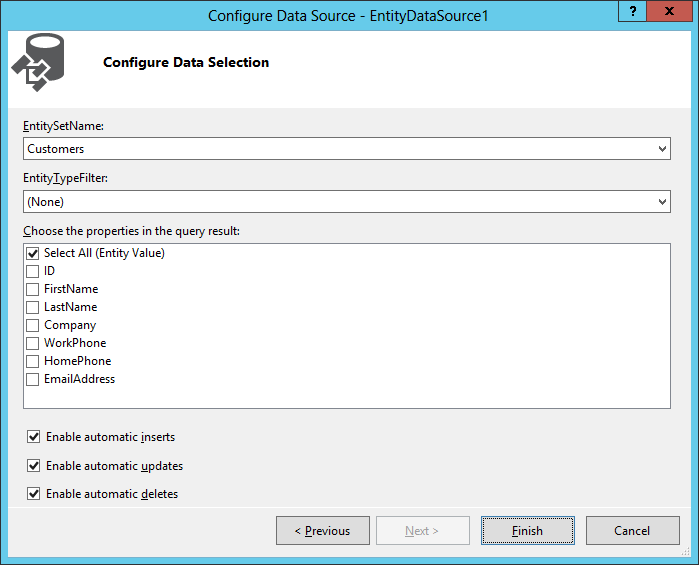
* 1. Switch the page **start.aspx** from **Code View** to **Design View**. You should see the **GridView** control and the **EntityDataSource** control render on the page in **Design View** as shown in the following screenshot.



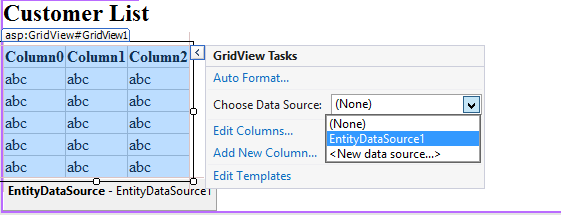
1. Configure the **Entity Data Source:**
   1. Select the **EntityDataSource1** on the page and click the button with the arrow icon on the right side of this control and select **Configure Data Source** as shown below:



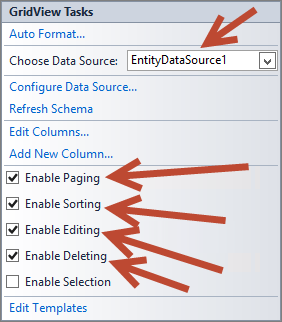
* 1. On the **Configure Data Source – EntityDataSource1 d**ialog, select the Named Connection of **WingtipCRMEntities** and click **Next**.
  2. Select **Customers** for the **Entity Set Name**.
  3. Place a check in **Enable automatic inserts, Enable automatic updates, Enable automatic deletes** (as shown below and click **Finish.**



1. Configure the DataView control to use this EntityDataSource:
   1. Select the GridView and click on the button with the arrow icon as shown below:



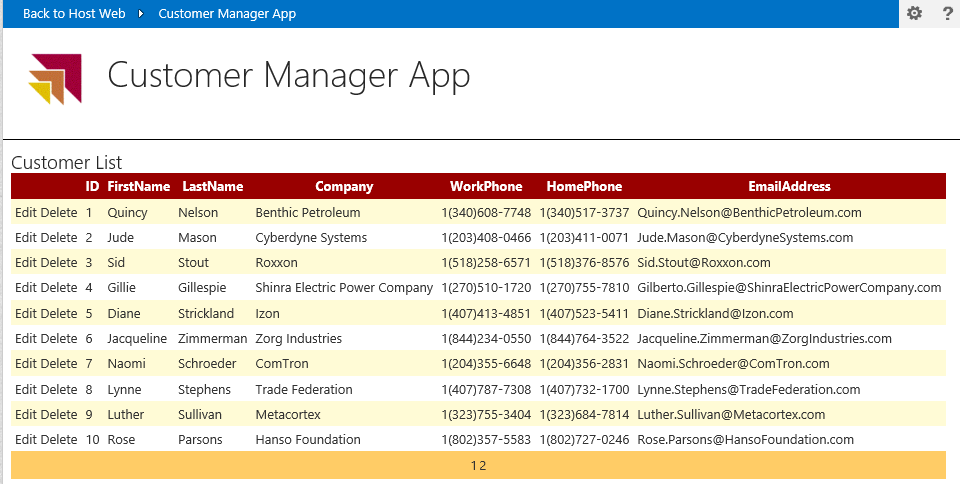
* 1. Now select the **EntityDataSource1** in the **Choose Data Source** dropdown.
  2. Place a check in the **Enable Paging, Enable Sorting, Enable Editing, Enable Deleting** check boxes.



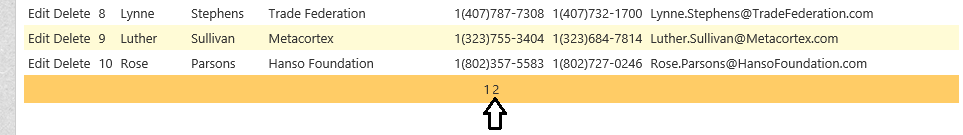
* 1. Click on the **Auto Format…** menu option and select **Colorful** from the list of options and click **OK**

#### Build and Test the Project

1. Build and test your application by pressing **[F5]** or **Debug 🡪 Start Debugging**.
   1. Once the solution has been deployed, Internet Explorer will launch and navigate to the Start page. You should see 10 customers displayed on the page.



* 1. Locate the paging menu at the bottom of the GridView control and click on the link for page 2.



* 1. Experiment with moving back and forth between pages using the paging menu.
  2. Now try to update a customer record. Click **Edit** for ID 1 Quincy Nelson. Change **Last Name** to **Adams**
  3. Now Click **Update.**
  4. **Delete** ID 2 (Jude Mason)
  5. **Close** the browser to stop debugging

#### Rounding out the project… or Making this a completely CRUDy project…

1. The only thing missing from a completely CRUDy (Create, Read, Update, Delete) application is the ability to add new records (or Create). Let’s see how hard adding this Create option is...
2. Back on your Start.aspx page switch to source view and add another **<div></div>** tag after the closing </div> tag you added earlier

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

<h2>Customer List</h2>

<div>

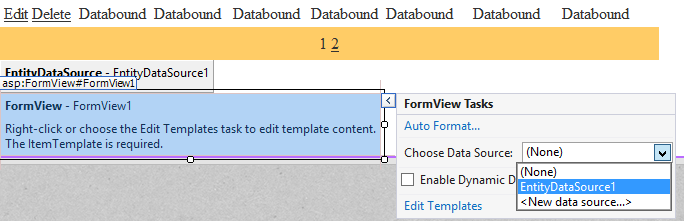
………

</div>

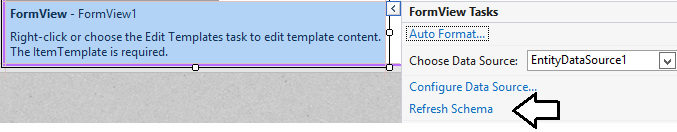
<div></div>

</asp:Content>

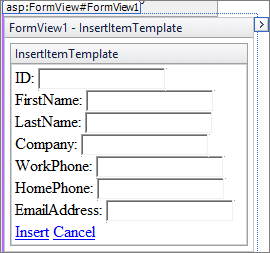
1. Add a **FormView** webpart in between your second <div></div> tag.
2. Switch back to **Design** **View** and on the **FormView** click on the Right Arrow and then set Choose Data Source to **EntityDataSource1** as shown below.



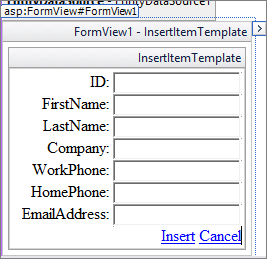
* 1. Now click **Refresh** **Schema** to populate the Form View.



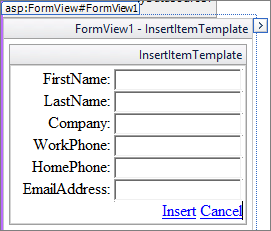
* 1. Now click on **Edit Templates** in the same pop-up window and then select **Insert Item Template** from the **Display** drop down window. This will display the form that will be used to add new customer records.



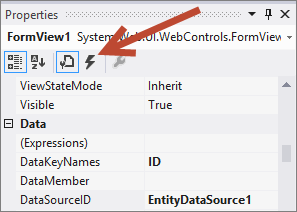
* 1. Click anywhere inside of the **InsertItem** Template and on the Visual Studio Menu bar select FORMAT 🡪 Justify 🡪 Right.



* + 1. **Delete** the **ID** row completely from the **InsertItem Template** (We do not need this as it is auto populated by SQL Server).



* + 1. Now expand the **Right** **arrow** on the **Form View** and select **End Template Editing**
  1. Next Select the **FormView** by clicking on it and in the **Properties** window in Visual Studio:
     1. Underneath the **Behavior** section set the **DefaultMode** property to **Insert**
  2. **Save** your changes to persist the updates to the FormView
  3. Select the FormView control again and in the Properties window in Visual Studio show the Events for this control by clicking the lightning bolt icon as shown below:



* 1. Now select and double click on the **IntemInserted** event to autowireup an event handler. In the **FormView1\_ItemInserted** event handler, add the following line of code:

protected void FormView1\_ItemInserted(object sender, FormViewInsertedEventArgs e) {

GridView1.DataBind();

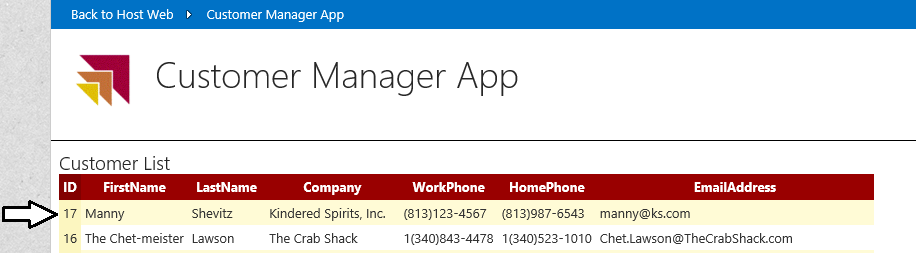
}

#### 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 Start page.
3. Go to the bottom of the start page where the FormView control is located. Enter customer information into the FormView control for yourself or use sample customer data like the sample customer data shown in the following screenshot. AFter entering in sample data for a new customer, click **Insert**



1. In the GridView click on the **ID** column header two times to sort ID in descending order. Verify that your sample customer data appears at the end of the list (e.g. ID 17)



We have just created a completely CRUDy application writing only a single line of C# Code!

### Exercise 4: Customizing the Application with a new Client Web Part

In this exercise you will embed the CRUDy application parts in a new Client Web Part that can be reused.

1. Make sure you continue with the project you created in the previous exercise.
2. Now add a client Web Part to the app project.
   1. Using the **Solution Explorer** tool window, right-click the **CustomerManager** project and select **Add 🡪 New Item**.
   2. In the **Add New Item** dialog, select the **Client Web Part (Host Web)** in the **Visual C# Items \ Office / SharePoint** category.
      1. **Name:** AppPart
      2. Click **Add**
   3. In the **Create Client Web Part** dialog, select the option **Create a new client web part page,** enter **AppPart** for the name and click **Finish**.
   4. Using the **Solution Explorer** tool window, within the **CustomerManagerWeb** project, within the **Pages** folder right-click **AppPart.aspx** and select **Open**.
   5. Add a reference to the jQuery library to the **<head>** part of the page:  
      (**NOTE**: You **MUST** check the current version number of the jQuery file located In the Solution Explorer CustomerManagerWeb Scripts Folder (If this has changed here you **MUST** also change it in the script line below)

<script type=”text/javascript” src="/Scripts/jquery-2.1.0.js"></script>

* 1. Add the following markup to the body of the page

<form id=”form1” runat=”server”>

<b>Hello world, this is coming from the remote web!</b>

</form>

* 1. Modify the contents of the **AppPart\Elements.xml** file in the **CustomerManager** project to include the following:

<ClientWebPart Name="AppPart"

Title="Customers App Part"

Description="Displays CRUD based Customer Information"

DefaultWidth="1000"

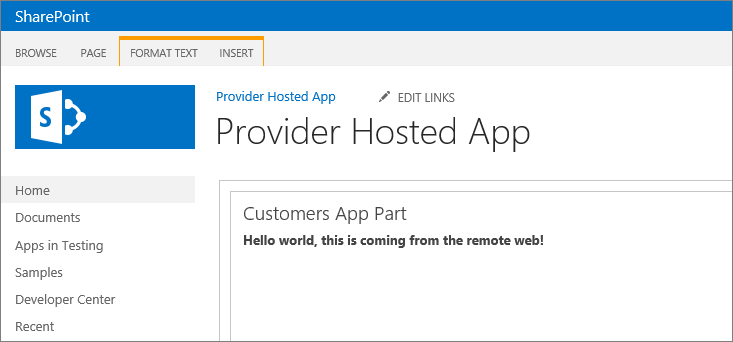
DefaultHeight="600">

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

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

#### Build and Test the Project

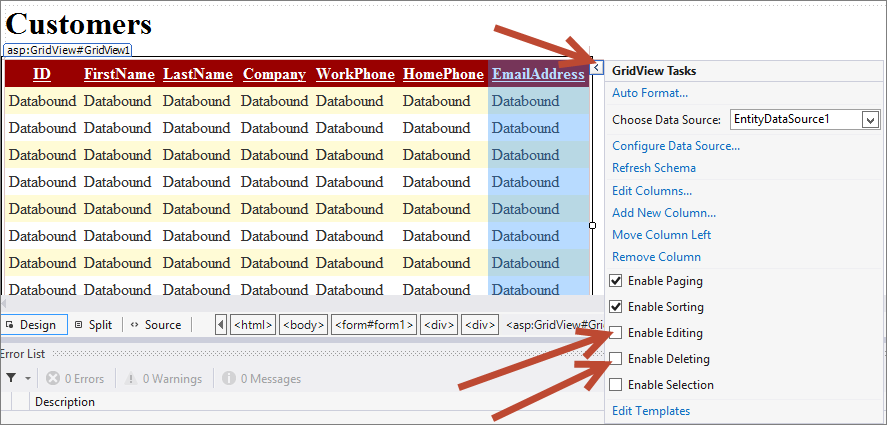
1. Build and test your application by pressing **[F5]** or **Debug 🡪 Start Debugging**.
2. Once the solution has been deployed, Internet Explorer will launch and navigate to the homepage of the remote web.
3. Using the browser navigate to http://Dev.wingtip.com
4. Add the Client Web Part to the page:
   1. Using the **Site Actions** “gear” icon in the top-right corner, select **Edit Page**.
   2. Using the ribbon, select the **Page** tab, then click the **Insert** tab and click the **App Part** button.
   3. Select the **Customers App Part** and click the **Add** button.
   4. Notice how the client Web Part is how showing content form the remote web site:



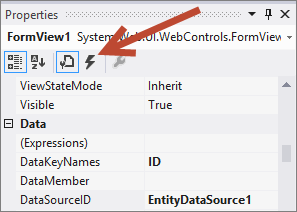
1. **Save** the page and then **close** the browser to stop the debugger and go back to Visual Studio.

#### Making the AppPart CRUDy (i.e. adding our Data Access capability into the AppPart)

1. **Back in the Application Open the start.aspx page in the Pages folder**
2. **Select everything in between the <asp:Content…> </asp:Content> tag (but not this tag itself) and copy this.**
3. **Open the AppPart.aspx page in the Pages folder, Select the <b>Hello world, this is coming from the remote web!</b> and paste this copied content in place of the selected text in the <form id=”form1” runat=”server”></form> tag.**
4. **Switch the AppPart.aspx page to Design view**
5. **Select the GridView, expand the Right arrow and reselect the Enable Editing and Enable Deleting Choices  
   (Note: these two settings do not always transfer over when copying a GridView between locations)**



1. **Now the only thing left to re-connect is that event handler we created on the other form:**
   1. **Save** your changes to persist the updates to the AppPart.aspx page
   2. Select the FormView control and in the Properties window in Visual Studio show the Events for this control by clicking the lightning bolt icon as shown below:



* 1. Now select and double click on the **IntemInserted** event to autowireup an event handler. In the **FormView1\_ItemInserted** event handler add the following line of code:

protected void FormView1\_ItemInserted(object sender, FormViewInsertedEventArgs e) {

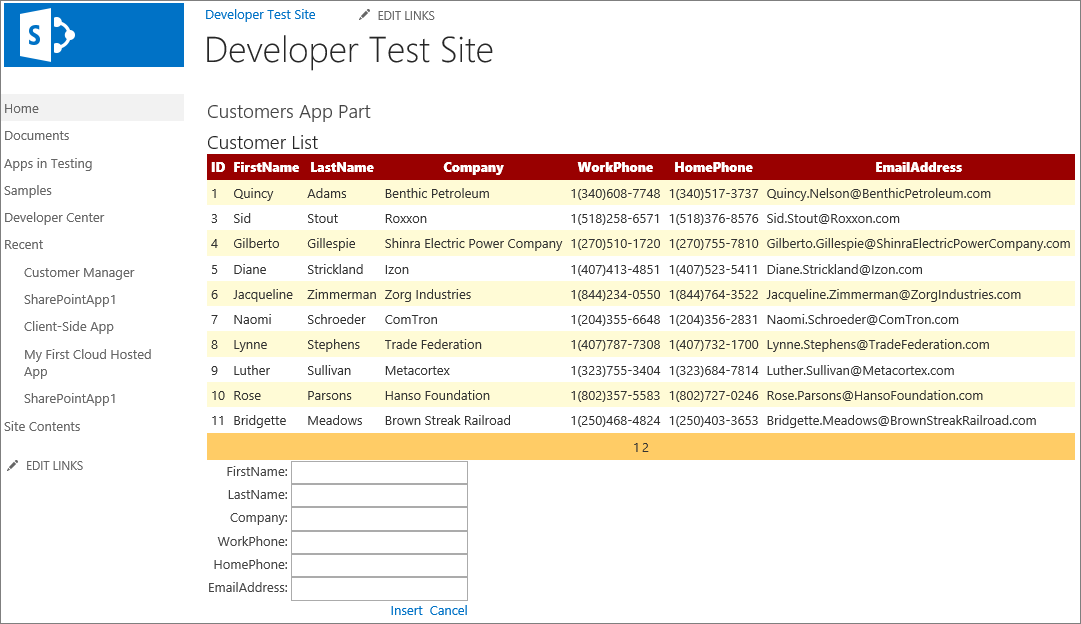
GridView1.DataBind();

}

* 1. **Now we are ready to test the project.**

#### 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 homepage of the remote web.
3. Using the browser navigate to http://Dev.wingtip.com
   1. We should see the updated Client Web Part on the page (as we added an earlier version and then saved the page after editing).
   2. Notice how the client Web Part is how showing content form the remote web site:



1. Feel free to experiment with testing (Editing, Deleting, and Inserting data as time permits)… Explore this new CRUDy App Part…
2. When satisfied that this still works as expected. Close the browser to stop the debugger and go back to Visual Studio.