## Pages, Navigation and User Interface

**Lab Time**: 60 minutes

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

**Lab Overview**: In this lab you will work with a SharePoint solution and a SharePoint application to add pages and create a user interface experience. Along the way you will learn how to add site pages which include web parts and views with custom rendering. You will also learn how to build out the user interface of a SharePoint app using the chrome control, app parts and UI custom actions.

### 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://pagesnavui.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:

[..]\PagesNavUi

* 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: Add Site Pages to a SharePoint Solution

In this exercise you will add some site pages to an existing solution and create links to these new pages.

1. Open an existing starter project in Visual Studio 2012:
   1. Launch **Visual Studio 2012** as administrator: **Start** 🡪 **All Programs** 🡪 **Microsoft Visual Studio 2012** 🡪 **Visual Studio 2012**
   2. Select **File 🡪 Open 🡪 Project/Solution**.
   3. In the **Open Project** dialog, select the following project provided in the files associated with this lab:

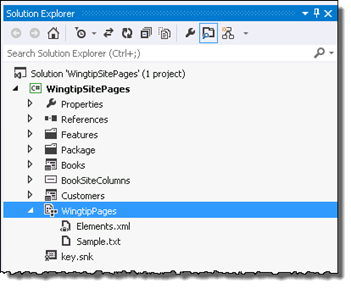
[..]\PagesNavUi\Exercises\Ex2\StarterProject\WingtipSitePages.sln

Note: You may get an error from Visual Studio about not having a site URL or debugging site specified. Ignore it for now.

1. Ensure the project is using the lab’s site collection for testing:
   1. Select the project **WingtipSitePages** in the **Solution Explorer** tool window.
   2. Look in the **Properties** tool window and ensure the **Site URL** property is set to <http://pagesnavui.wingtip.com>.
   3. Make sure the **Sandbox Solution** property of the **WingtipSitePages** project is set to a value of **False**.

Remember that the VM you are using to complete lab exercises does not support server-side code from a sandboxed solution. The **WingtipSitePages** project will fail to load if you attempt to debug it as a sandboxed solution.

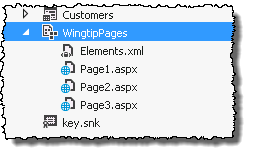
1. The **WingtipSitePages** project already contains SharePoint Project Items to create a few site columns and two lists. These two lists, **Books** and **Customers**, will be used throughout the remainder of this lab.
2. Create a new module to provision some site pages to the site:
   1. Right-click the project **WingtipSitePages** in the **Solution Explorer** tool window and select **Add 🡪 New Item**.
      1. In the **Add New Item** dialog, select the **Module** template from the **Visual C# Items 🡪 Office / SharePoint** category.
      2. **Name:** WingtipPages



1. Add the provided site pages to the **WingtipPages** module:
   1. Right-click the **WingtipPages** module in the **Solution Explorer** tool window and select **Add 🡪 Existing Item**.
   2. In the **Add Existing Item** dialog, select the three **\*.ASPX** files in the files provided with this exercise:

[..]\PagesNavUi\Exercises\Ex2\StarterFiles

* 1. Within the **WingtipPages** module, right click **Sample.txt** and select **Delete**.

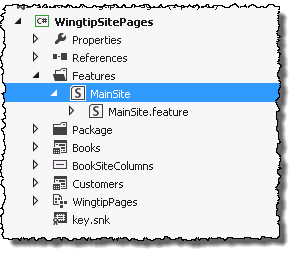


1. Verify the **WingtipPages** module is included in the list of items in the Feature:
   1. In the **Solution Explorer** tool window, expand the **WingtipSitePages** project to the **Features\MainSite** node.
   2. Right-click the **MainSite** Feature and select **View Designer**.
   3. Verify the **WingtipPages** module is listed on the right-hand column **Items in the Feature**. If it isn’t select it and use the arrows between the two columns to add it from the left column: **Items in the Solution**.

#### Add Navigation Elements to the Site to Surface the New Site Pages

At this point your project will simply create a few pages in the development site. Users would have to know the URL and manually type it into the browser to navigate to these files. To make this more user friendly, write some custom code to add some navigation elements to the site to point to these three new files.

1. In the **Solution Explorer** tool window, expand the **WingtipSitePages** project to the **Features\MainSite** node.
2. Right-click the **MainSite** Feature and select **Add Event Receiver**.



1. Within the **MainSite.EventReceiver.cs** code file, add the following line to the top of the file to add a reference to the SharePoint navigation namespace:

using Microsoft.SharePoint.Navigation;

1. Find the commented out method **FeatureActivated()** and replace it with the following code to create three new links in the top navigation link bar:

public override void FeatureActivated(SPFeatureReceiverProperties properties) {

SPSite siteCollection = properties.Feature.Parent as SPSite;

if (siteCollection != null) {

SPWeb site = siteCollection.RootWeb;

// create menu items on top link bar for custom site pages

SPNavigationNodeCollection topNav = site.Navigation.TopNavigationBar;

topNav.AddAsLast(new SPNavigationNode("Page 1", "WingtipPages/Page1.aspx"));

topNav.AddAsLast(new SPNavigationNode("Page 2", "WingtipPages/Page2.aspx"));

topNav.AddAsLast(new SPNavigationNode("Page 3", "WingtipPages/Page3.aspx"));

}

}

1. While the previous code will create three new links when the Feature is activated, these links are not automatically removed when you deactivate the Feature. In addition, the provisioned files are not automatically removed. To clean up everything done by the Feature activation, find the commented out method **FeatureDeactivating()** and replace it with the following code to delete the links and provisioned pages:

public override void FeatureDeactivating(SPFeatureReceiverProperties properties) {

SPSite siteCollection = properties.Feature.Parent as SPSite;

if (siteCollection != null) {

SPWeb site = siteCollection.RootWeb;

try {

// delete folder of site pages provisioned during activation

SPFolder sitePagesFolder = site.GetFolder("WingtipPages");

sitePagesFolder.Delete();

}

catch { }

SPNavigationNodeCollection topNav = site.Navigation.TopNavigationBar;

for (int i = topNav.Count - 1; i >= 0; i--) {

if (topNav[i].Url.Contains("WingtipPages")) {

// delete node

topNav[i].Delete();

}

}

}

}

#### Update the User Interface of the Provisioned Site Page Page1.aspx

Page1.aspx has references to CSS classes to create a more specialized user interface. In order to get this user interface you need to provision a new CSS file.

1. Add a new CSS file to the provisioned files to modify the presentation of the site page **Page1.aspx**:
   1. Right-click the **WingtipPages** module and select **Add 🡪 Existing Item**.
   2. In the **Add Existing** Item dialog, select the styles.css file from the files provided with this exercise:

[..]\PagesNavUi\Exercises\Ex2\StarterFiles

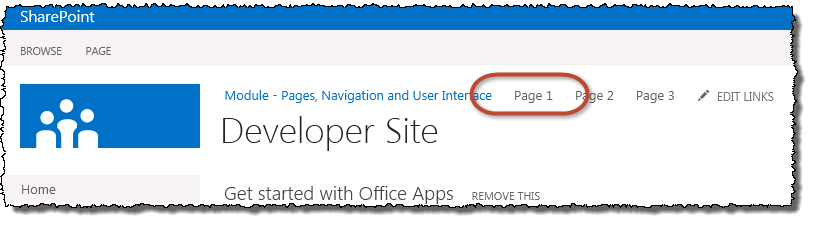
1. Add a reference to the CSS file in **Page1.aspx**:
   1. In the **Solution Explorer** tool window, find the **WingtipPages\Page1.aspx** file.
   2. Right-click **Page1.aspx** and select **Open**.
   3. Find the ASP.NET content placeholder **PlaceHolderAdditionalPageHead** and add the following markup to the content of the content placeholder:

<link href="styles.css" rel="stylesheet" type="text/css" />

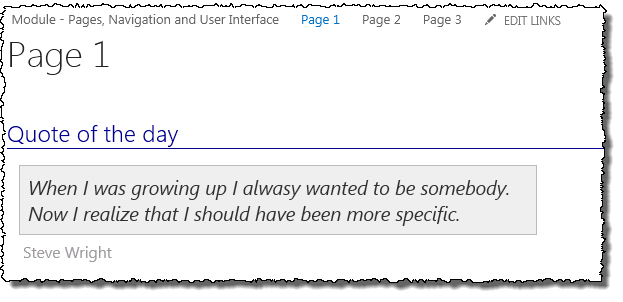
1. Save all changes: **File 🡪 Save All**.
2. Build and test your application by pressing **[F5]** or **Debug 🡪 Start Debugging**.

Note: If prompted with an **Attach Security Warning** dialog, click **Attach**.

1. Once the solution has been deployed, Internet Explorer will launch and navigate to the default page for the site.
2. In the top navigation, select **Page 1** to navigate to the **Page1.aspx** site page you provisioned.



1. Notice how **Page1.aspx** has unique styling compared to just plain old text:



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

#### Update the User Interface of the Provisioned Site Pages Page2.aspx & Page3.aspx

Page2.aspx is a Web Part Page but does not include any Web Parts. In this section you will add a new instance of the XsltListViewWebPart to display the Customer list. For Page3.aspx, which is also a Web Part Page, you will also add a XsltListViewWebPart to show the contents of the Books list.

1. Add a Web Part to **Page2.aspx**:
   1. In the **Solution Explorer** tool window, find the **WingtipPages\Page2.aspx** file.
   2. Right-click **Page2.aspx** and select **Open**.
   3. Find the ASP.NET content placeholder **PlaceHolderMain** and add the following markup to the content of the content placeholder:

<WebPartPages:WebPartZone ID="Main" Title="Main Web Part Zone"

FrameType="TitleBarOnly" runat="server" >

<ZoneTemplate>

<WebPartPages:XsltListViewWebPart

runat="server"

ID="CustomersWebPart"

Title="Customers"

ListUrl="Lists/Customers"

ChromeType="None"

InplaceSearchEnabled="false" >

</WebPartPages:XsltListViewWebPart>

</ZoneTemplate>

</WebPartPages:WebPartZone>

1. Add a Web Part to **Page3.aspx**:
   1. In the Solution Explorer tool window, find the **WingtipPages\Page3.aspx** file.
   2. Right-click **Page3.aspx** and select **Open**.
   3. Find the ASP.NET content placeholder **PlaceHolderMain** and add the following markup to the content of the content placeholder:

<WebPartPages:WebPartZone ID="Main" Title="Main Web Part Zone"

FrameType="TitleBarOnly" runat="server" >

<ZoneTemplate>

<WebPartPages:XsltListViewWebPart

runat="server"

ID="BooksWebPart"

Title="Books"

ListUrl="Lists/Books"

ChromeType="None"

InplaceSearchEnabled="false" >

</WebPartPages:XsltListViewWebPart>

</ZoneTemplate>

</WebPartPages:WebPartZone>

1. Save all changes: **File 🡪 Save All**.
2. Build and test your application by pressing **[F5]** or **Debug 🡪 Start Debugging**.
   1. If prompted with an **Attach Security Warning** dialog, click **Attach**.
3. When the site loads in Internet Explorer, select **Page 2** to navigate to the **Page2.aspx** site page you provisioned. Notice the **Customers** list is displayed in a typical format.
4. Now select **Page 3** and notice it displays the **Books** list in the same format.
5. Close the browser to stop the debugger and go back to Visual Studio.

In this exercise you added a few site pages to a sandbox solution and creating links in the navigation pointing to these pages.

### Exercise 3: Customize the Rendering of a SharePoint List Using JavaScript

In this exercise you will customize the view of a SharePoint list using the new SharePoint 2013 capability that enables you to use JavaScript to control the layout of the list contents.

1. Open an existing starter project in Visual Studio 2012:
   1. Launch **Visual Studio 2012** as administrator:
      1. **Start 🡪 All Programs 🡪 Microsoft Visual Studio 2012 🡪 Visual Studio 2012**
   2. Select **File 🡪 Open 🡪 Project/Solution**.
   3. In the **Open Project** dialog, select the following project provided in the files associated with this lab:

[..]\PagesNavUi\Exercises\Ex3\WingtipSitePages.sln

1. Ensure the project is using the lab’s site collection for testing:
   1. Right-click the project **WingtipSitePages** in the **Solution Explorer** tool window and select **Properties**.
   2. If the **Site URL** property does not show <http://pagesnavui.wingtip.com>, update it so it does.
2. Create a new module that will hold the JavaScript files used to customize the list rendering
   1. Right-click the project **WingtipSitePages** in the **Solution Explorer** tool window and select **Add 🡪 New Item**.
      1. In the **Add New Item** dialog, select the **Module** template from the **Visual C# Items 🡪 Office / SharePoint** category.
      2. **Name:** Scripts
   2. Within the **Scripts** module, right click **Sample.txt** and select **Delete**.
3. Add a new JavaScript file to the **Scripts** module:
   1. Right-click the **Scripts** module and select **Add 🡪 New Item**.
      1. In the **Add New Item** dialog, select the **JavaScript File** template from the **Visual C# Items 🡪 Web** category.
      2. **Name:** CustomRendering1.js
4. First, add a new JavaScript function that will initialize and override the context. To accomplish this add the following code to the **CustomRendering1.js** file:

(function () {

// create and intialize object to override context

var overrideCtx = {};

overrideCtx.BaseViewID = 2;

overrideCtx.ListTemplateType = 10000;

// initialize template

overrideCtx.Templates = {};

overrideCtx.Templates.Header = "<h2>My favorite classic books</h2>";

overrideCtx.Templates.Item = customItem;

// register with page template manager

SPClientTemplates.TemplateManager.RegisterTemplateOverrides(overrideCtx);

})();

1. The code you just added includes a reference to an object **customItem**. Add the following code to the end of the file to implement this object:

function customItem(ctx) {

// create item style

var itemStyle = "display:inline-block;width:200px;height:120px;border:black 1px solid;";

// create and return HTML for each item

return "<div style='" + itemStyle + "'>" +

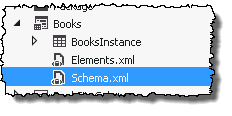
ctx.CurrentItem.Title + "<br/>by<br/>" + ctx.CurrentItem.BookAuthor +

"</div>";

}

#### Update the SharePoint List Books to use the new custom JavaScript rendering

1. In the **Solution Explorer** tool window, expand the **WingtipSitePages** project to the **Books\Schema.xml** node.



1. Right-click the **Schema.xml** file and select **Open**.
2. Create a new view for the Books list that uses the custom JavaScript rendering:
   1. Find the closing **</Views>** element which should be on or near line #79.
   2. Add the following markup just above the closing **</Views>** element:

<View BaseViewID="2"

Name="8d2719f3-c3c3-415b-989d-33840d8e2ddb"

DisplayName="Custom Client-side Rendering"

Type="HTML"

WebPartZoneID="Main"

SetupPath="pages\viewpage.aspx"

Url="CustomRendering.aspx"

DefaultView="TRUE">

<ViewFields>

<FieldRef Name="Title" />

<FieldRef Name="BookAuthor" />

</ViewFields>

<Query />

<Toolbar Type="Standard" />

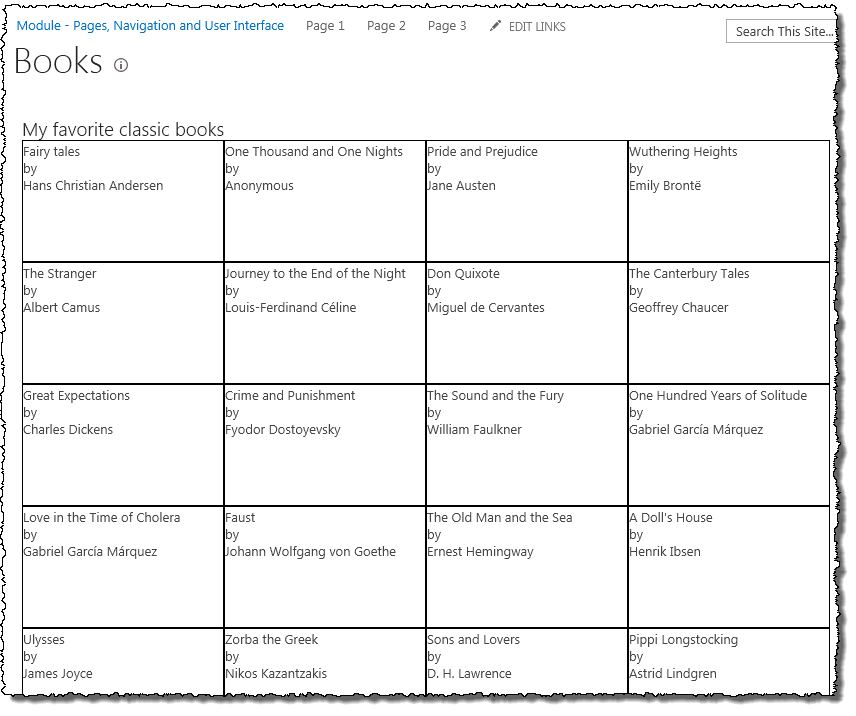
<XslLink>main.xsl</XslLink>

<JSLink Default="TRUE">~site/Scripts/CustomRendering1.js</JSLink>

</View>

Notice the view has the attribute **DefaultView** set to **TRUE** and there is a new element **JSLink** that is pointing to the JavaScript file you just created.

1. Save all changes: **File 🡪 Save All**.
2. Build and test your application by pressing **[F5]** or **Debug 🡪 Start Debugging**.
   1. If prompted with an **Attach Security Warning** dialog, click **Attach**.
3. Once the solution has been deployed, Internet Explorer will launch and navigate to the default page for the site.
4. In the quick launch navigation, select **Books** to navigate to the **Books** list.
5. Notice how the rendering looks nothing like a typical SharePoint list:



1. Close the browser to stop the debugger and go back to Visual Studio.
2. Now, let’s make it a bit more interesting with a different JavaScript rendering. Add a new JavaScript rendering file:
   1. Right-click the **Scripts** module and select **Add 🡪 New Item**.
      1. In the **Add New Item** dialog, select the **JavaScript File** template from the **Visual C# Items 🡪 Web** category.
      2. **Name:** CustomRendering2.js
3. Add the following JavaScript code to the **CustomRendering2.js** file:

(function () {

var overrideCtx = {};

overrideCtx.BaseViewID = 2;

overrideCtx.ListTemplateType = 10000;

overrideCtx.Templates = {};

overrideCtx.Templates.Header = "<h2>My favorite classic books</h2>";

overrideCtx.Templates.Item = customItem;

SPClientTemplates.TemplateManager.RegisterTemplateOverrides(overrideCtx);

})();

function customItem(ctx) {

var itemStyle = "display:inline-block;width:200px;height:120px;max-height:120px;margin:12px;" +

"border: 1px black solid;background-color:#ffe;padding-top:20px;" +

"color:black;text-align:center;font-size:12px;";

var itemHTML = "<div style='" + itemStyle + "'>" +

"<strong>" + ctx.CurrentItem.Title + "</strong>" +

"<br/><br/>by<br/><br/>" +

"<em>" + ctx.CurrentItem.BookAuthor + "</em>" +

"</div>";

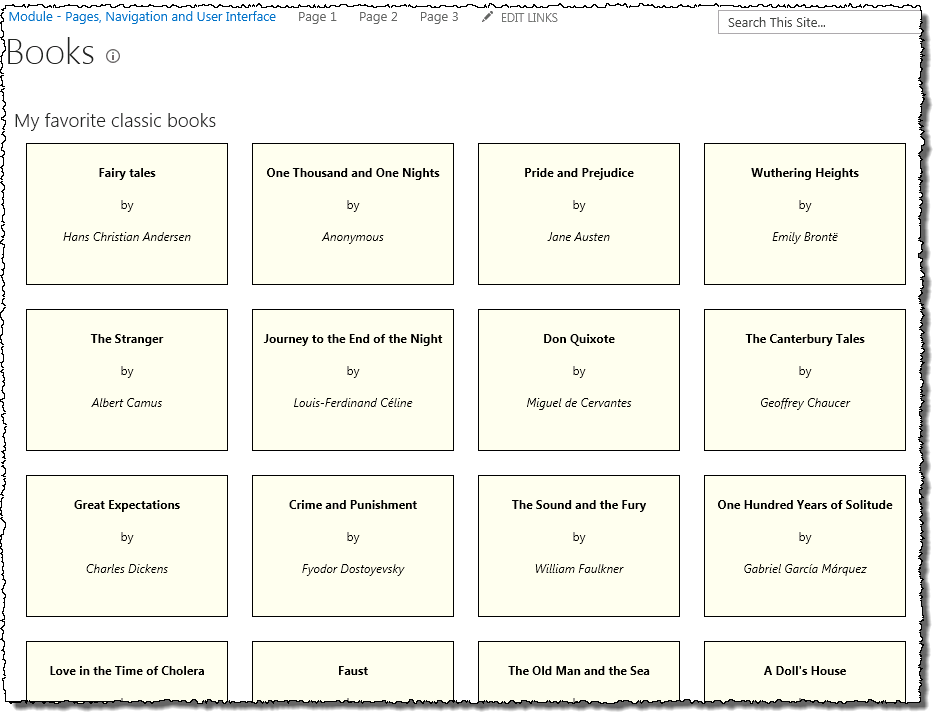
return itemHTML;

}

1. Update the view to use the new JavaScript file:
   1. Right-click the **Books\Schema.xml** file and select **Open**.
   2. Find the line towards the end of the file that looks like this:

<JSLink Default="TRUE">~site/Scripts/CustomRendering1.js</JSLink>

1. Update this line to point the **CustomRendering2.js** file by changing the **1** to a **2**.
2. Save all changes: **File 🡪 Save All**.
3. Build and test your application by pressing **[F5]** or **Debug 🡪 Start Debugging**.
   1. If prompted with an **Attach Security Warning** dialog, click **Attach**.
4. Once the solution has been deployed, Internet Explorer will launch and navigate to the default page for the site.
5. In the quick launch navigation, select **Books** to navigate to the **Books** list.
6. Notice how the rendering looks much more customized than before:



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

In this exercise you implemented a new view with a custom rendering using the new capability in SharePoint 2013 that enables developers to use JavaScript for customized renderings.

### Exercise 4: Creating a User Interface for a Cloud-Hosted App

In this lab you will create a provider-hosted app and work on extending the user interface for the app with the chrome control, and app part and UI custom actions.

1. Create a new project in Visual Studio 2012:
   1. Launch **Visual Studio 2012** as administrator: **Start** 🡪 **All Programs** 🡪 **Microsoft Visual Studio 2012** 🡪 **Visual Studio 2012**.
   2. 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: **RemotePages**
   4. In the **New App for SharePoint** wizard, use the following values to complete the wizard and click **Finish**.
      1. **What is the name of your App for SharePoint?** Remote Pages
      2. **What site do you want to use for debugging?** <http://pagesnavui.wingtip.com>
      3. **How do you want to host your app for SharePoint?** Provider-hosted
2. Change the new app’s authentication method to use internal authentication:
   1. Open the project **RemotePages** in the **Solution Explorer** tool window.
   2. Right-click the **AppManifest.xml** file and select **View Code**.
   3. 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. Next, configure the projects to disable the use of SSL so everything happens over HTTP rather than HTTPS:
   1. Select the project **RemotePagesWeb** 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**.
2. Add a new image to the project to use as the startup icon for the app:
   1. Using the **Solution Explorer** tool window, within the **RemotePages** project, right-click the **AppManifest.xml** file and select **Open**.
   2. In the **App Manifest** designer, click the **[…]** button next to the **Icon** field to select a new icon.
   3. Select the **CPTAppIcon96x96.png** image found in the extra files associated with this course and click **OK**:

[..]\ExtraStudentFiles\Images\CPT

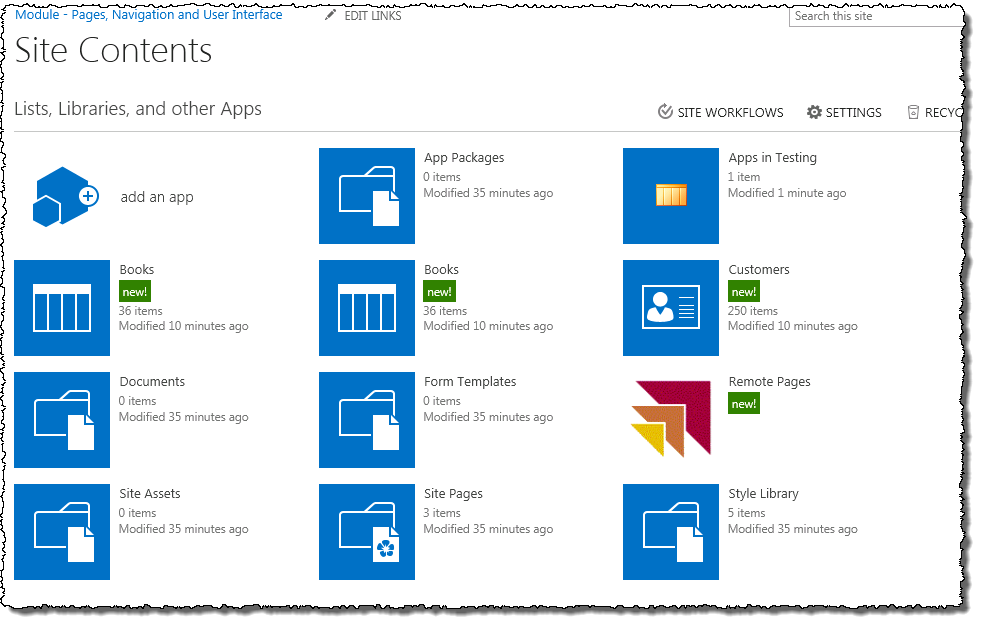
1. Delete the old icon:
   1. Using the **Solution Explorer** tool window, within the **RemotePages** project, right-click the **AppIcon.png** file and select **Delete**.
2. Add some content to the app’s homepage so we know we are hitting our new site:
   1. Using the **Solution Explorer** tool window, within the **RemotePagesWeb**, right-click the **Pages\Default.aspx** file and select **Open**.
   2. Add the following markup within the **<DIV>** already in the page:

<h1>Hello World</h1>

1. Remove all security code from the homepage’s code behind file for now:
   1. Using the **Solution Explorer** tool window, within the **RemotePagesWeb**, right-click the **Pages\Default.aspx** file and select **View Code**.
   2. Remove all the code within the **Page\_Load()** method.
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. 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://pagesnavui.wingtip.com> & click the **Site Contents** link the left-hand navigation. You should see your app listed in the **Lists, Libraries and other Apps** section of the page.



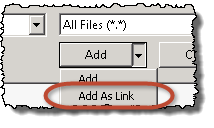
1. Close the browser to stop the debugger and go back to Visual Studio.

#### Add Some Content to the Application:

1. With a working app, now let’s start adding some content and branding to the app.
2. Add the SharePoint chrome control script files to the project:
   1. Right-click the **Scripts** folder in the **RemotePagesWeb** project and select **Add 🡪 Existing Item**.
   2. In the **Add Existing Item** dialog, browse to the following folder:

C:\Program Files\Common Files\Microsoft Shared\Web Server Extensions\15\TEMPLATE\LAYOUTS

* 1. Find the following two files and using **[CTRL]+CLICK**, select both of them:
     1. SP.UI.Controls.js
     2. SP.UI.Controls.debug.js
  2. To the right of the **Add** button in the dialog, there is a **down** arrow. Click the arrow and select **Add as Link**. This will add the files as references in the project, but will not create copies in the project.



1. Next, add a new 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 folder that contains files accompanying this exercise:

[..]\PagesNavUi\Exercises\Ex4

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

1. Now modify the master page that will add the chrome control to all pages in the remote site that use the master page:
   1. Using the Solution Explorer tool window, within the **RemotePagesWeb** project right-click the **Pages\RemoteWeb.master** file and select **Open**.
   2. Add the following markup to the **<head>** part of the page to reference the CSS file:

<link href="/Contents/app.css" type="text/css" rel="stylesheet" />

* 1. 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 **RemotePagesWeb** Project **Scripts** Folder (If this has changed here you **MUST** also change it in the script line below)

<script type=”text/javascript” src="/Scripts/jquery-1.7.1.min.js"></script>

* 1. Add the following JavaScript in a new **<script>** tag in the **<head>** part of the page, after the jQuery reference. This will dynamically load the required JavaScript file at runtime:
     1. You will find this code in the folder that contains files accompanying this exercise as **LoadSpUiControls.js**.

<script type="text/javascript">

var hostWebUrl;

var hostLayoutsUrl;

// Load the SharePoint resources.

$(document).ready(function () {

// Get the URI decoded app web URL.

hostWebUrl = decodeURIComponent(getQueryStringParameter("SPHostUrl"));

hostLayoutsUrl = hostWebUrl + "/\_layouts/15/";

// Load the js file and continue to the success handler.

$.getScript(hostLayoutsUrl + "SP.UI.Controls.js", renderChromeControl)

});

function getQueryStringParameter(paramToRetrieve) {

var params = document.URL.split("?")[1].split("&");

var strParams = "";

for (var i = 0; i < params.length; i = i + 1) {

var singleParam = params[i].split("=");

if (singleParam[0] == paramToRetrieve)

return singleParam[1];

}

}

</script>

* 1. Next, add a new JavaScript function to the end of the **<script>** block, just before the closing **</script>** tag, that will create and initialize the Chrome Control:
     1. You will find this code in the folder that contains files accompanying this exercise as **ChromeControl.js**.

function renderChromeControl() {

var options = {

siteUrl: hostWebUrl,

siteTitle: "Host Web",

appHelpPageUrl: "help.aspx?SPHostUrl=" + hostWebUrl,

appIconUrl: "/Contents/AppIcon.png",

appTitle: "Wingtip App",

settingsLinks: [

{ linkUrl: "start.aspx?SPHostUrl=" + hostWebUrl, displayName: "Home" },

{ linkUrl: "about.aspx?SPHostUrl=" + hostWebUrl, displayName: "About" },

{ linkUrl: "contact.aspx?SPHostUrl=" + hostWebUrl, displayName: "Contact" }

]

};

var nav = new SP.UI.Controls.Navigation("chrome\_ctrl\_container", options);

nav.setVisible(true);

}

1. Create a new folder for branding assets:
   1. Within the **Solution Explorer** tool window, right-click the project **RemotePagesWeb** and select **Add 🡪 New Folder**.
      1. Give the new folder the name of **Contents**.
2. Add a few branding files to the **Contents** folder:
   1. Right-click the **Contents** folder and select **Add 🡪 Existing Item**.
   2. In the **Add Existing Item** dialog, browse to the folder that contains files accompanying this exercise:

[..]\PagesNavUi\Exercises\Ex4

* 1. Select the following files and click **Add**:
     1. app.css
     2. background.jpg
     3. nochrome.css
     4. AppIcon.png

1. Add a few files to the Pages folder that will be used in the site:
   1. Create a new page: **start.aspx**:
      1. Right-click the **Pages** folder and select **Add 🡪 New Item**.
      2. In the **Add New Item** dialog, select the **Web Form** template within the **Visual C# \ Web** category.
         1. **Name:** Start.aspx
      3. 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">

<h1>Start</h1>

</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

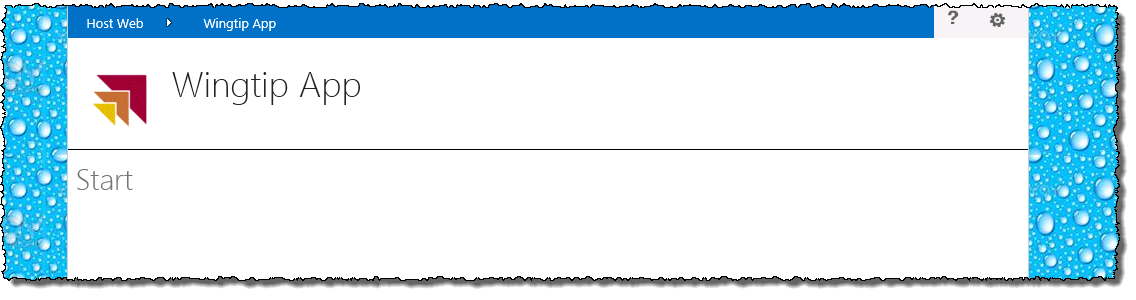
1. Ensure all four of the pages you just created reference the new master page you added to the Pages library previously. Do this by adding the following attribute to each files **@Page** directive:

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 **RemotePages** project, right-click the **AppManifest.xml** file and select **Open**.
   2. For the **Start Page**, select **RemotePagesWeb/Pages/Start.aspx**.
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. Once the solution has been deployed, Internet Explorer will launch and navigate to the **start.aspx** page of the remote web.
3. Notice how the pages have a similar look and feel at the top and throughout the styles using the app web’s branding:



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

#### Add a New Client Web Part to the Application

1. Now add a client Web Part to the app project.
   1. Using the **Solution Explorer** tool window, right-click the **RemotePages** 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
   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 **RemotePagesWeb** 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:

<script src="/Scripts/jquery-1.7.1.min.js"></script>

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

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

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

<ClientWebPart Name="AppPart"

Title="Wingtip App Part Title"

Description="My app part"

DefaultWidth="600"

DefaultHeight="360">

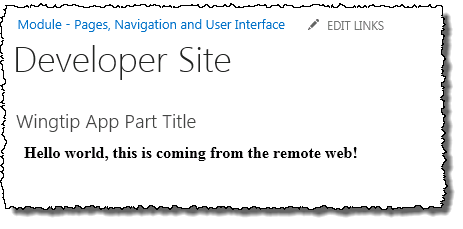
<Content Type="html" Src="~remoteAppUrl/Pages/AppPart.aspx?{StandardTokens}" />

</ClientWebPart>

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://pagesnavui.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 **Wingtip App Part Title** and click the **Add** button.
   4. Notice how the client Web Part is how showing content form the remote web site:



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

#### Add a Custom Action to the Project

1. Add a utility class to the **RemotePagesWeb** project:
   1. Using the **Solution Explorer** tool window, right-click **RemotePagesWeb** project and select **Add 🡪 Existing Item**.
   2. Select the file **HtmlTableWriter.cs** from the files in the folder that accompany this exercise and select **Add**:

[..]\PagesNavUi\Exercises\Ex4

1. Now add a page to serve as the target for a custom action
   1. Using the **Solution Explorer** tool window, within the **RemotePagesWeb** project, right-click the **Pages** folder and select **Add 🡪 New Item**.
      1. In the **Add New Item** dialog, select the **Web Form** template within the **Visual C# \ Web** category
         1. **Name:** CustomAction.aspx
   2. Using the Solution Explorer tool window, within the **RemotePagesWeb** project right-click the **Pages\CustomAction.aspx** file and select **Open**.
   3. Add the following markup to the ASP.NET page body:

<h2>UI Custom Action</h2>

<asp:PlaceHolder ID="pageContent" runat="server"></asp:PlaceHolder>

* 1. Using the **Solution Explorer** tool window, within the **RemotePagesWeb** project right-click the **Pages\CustomAction.aspx** file and select **View Code**.
  2. Add the following using statement to the file:

using RemoteAppPagesWeb;

* 1. Add the following code to the **Page\_Load()** method:

HtmlTableWriter table1 = new HtmlTableWriter();

foreach (var param in Request.QueryString.AllKeys) {

table1.AddRow("Request.QueryString['" + param + "']", Request.QueryString[param].ToString());

}

pageContent.Controls.Add(new LiteralControl("<h3>Query String Parameters</h3>"));

pageContent.Controls.Add(new LiteralControl(table1.ToString()));

HtmlTableWriter table2 = new HtmlTableWriter();

string urlSourcePage = Request.QueryString["Source"].ToString();

string linkSourcePage = "<a href='" + urlSourcePage + "'>" + urlSourcePage + "</a>";

table2.AddRow("[Source]", linkSourcePage);

string urlList = Request.QueryString["SPHostUrl"].ToString() + Request.QueryString["ListURLDir"].ToString();

string linkList = "<a href='" + urlList + "' target='\_blank'>" + urlList + "</a>";

table2.AddRow("[SPHostUrl] + [ListURLDir]", linkList);

string urlItem = Request.QueryString["SPHostUrl"].ToString() + Request.QueryString["ItemURL"].ToString();

string linkItem = "<a href='" + urlItem + "' target='\_blank'>" + urlItem + "</a>";

table2.AddRow("[SPHostUrl] + [ItemURL]", linkItem);

pageContent.Controls.Add(new LiteralControl("<h3>Links back to host web</h3>"));

pageContent.Controls.Add(new LiteralControl(table2.ToString()));

1. Next, add a new custom action to the app project that references the project:
   1. Using the **Solution Explorer** tool window, right-click the **RemotePages** project and select **Add 🡪 New Item**:
      1. Select the **Menu Item Custom Action** template
      2. **Name:** CustomAction
   2. In the **Create Custom Action for Menu Item** dialog, on the **Specify the properties to create custom action for menu item** page, use the following to complete the dialog and click **Next**:
      1. **Where do you want to expose the custom action?** Host Web
      2. **Where is the custom action scoped to?** ListTemplate
      3. **Which particular item is the custom action scoped to?** Custom List
   3. In the **Create Custom Action for Menu Item** dialog, on the **Specify the properties to create custom action for menu item** page, use the following to complete the dialog and click **Finish**:
      1. **What is the text on the menu item?** Custom Action
      2. **Where does the custom action navigate to?** ..\RemotePagesWeb\Pages\CustomAction.aspx
   4. Change the value of the **<UrlAction>** element to the following:

~remoteAppUrl/Pages/CustomAction.aspx

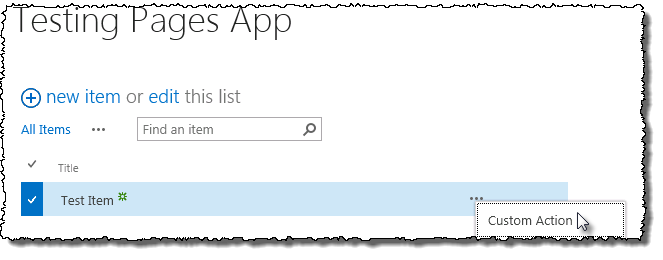
* 1. Add the following query string parameters to the end of the URL in the **<UrlAction>** element:

?Source={Source}&amp;ListURLDir={ListUrlDir}&amp;ListID={ListId}&amp;ItemURL={ItemUrl}&amp;ItemID={ItemId}&amp;{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://pagesnavui.wingtip.com> and click the **Site Contents** link in the left-hand navigation.
4. Create a new SharePoint list based on the custom list template:
   1. Click the **Add an app** link.
   2. On the **Site Contents > Your Apps**, click the **Custom List** template.
   3. Give the list of name of **Testing Pages App** and click **Create**.
5. Add an item to the new list:
   1. From the **Site Contents** page, click the **Testing Pages App**.
   2. Click the new item link to create a new item.
   3. Give the new item a **Title** of **Test Item** and click **Save**.
6. Click the edit control block (ECB) menu (the ‘**…**’) of item you just created and click **Custom Action**:



1. Notice the browser will redirect you to the other page.
2. Close the browser to stop the debugger and go back to Visual Studio.

#### Add a Custom Action to the Project that Opens in a Dialog

1. Add a new custom action to the app project that references the project:
   1. Using the **Solution Explorer** tool window, right-click the **RemotePages** project and select **Add 🡪 New Item**:
      1. Select the **Menu Item Custom Action** template
      2. **Name:** CustomAction
   2. In the **Create Custom Action for Menu Item** dialog, on the **Specify the properties to create custom action for menu item** page, use the following to complete the dialog and click **Next**:
      1. **Where do you want to expose the custom action?** Host Web
      2. **Where is the custom action scoped to?** ListTemplate
      3. **Which particular item is the custom action scoped to?** Custom List
   3. In the **Create Custom Action for Menu Item** dialog, on the **Specify the properties to create custom action for menu item** page, use the following to complete the dialog and click **Finish**:
      1. **What is the text on the menu item?** Custom Action Dialog
      2. **Where does the custom action navigate to?** ..\RemotePagesWeb\Pages\CustomAction.aspx
   4. Change the value of the **<UrlAction>** element to the following:

~remoteAppUrl/Pages/CustomAction.aspx

* 1. Add the following query string parameters to the end of the URL in the **<UrlAction>** element:

?Source={Source}&amp;ListURLDir={ListUrlDir}&amp;ListID={ListId}&amp;ItemURL={ItemUrl}&amp;ItemID={ItemId}&amp;{StandardTokens}

* 1. Add the following attributes to the **<CustomAction>** element:

HostWebDialog="TRUE"

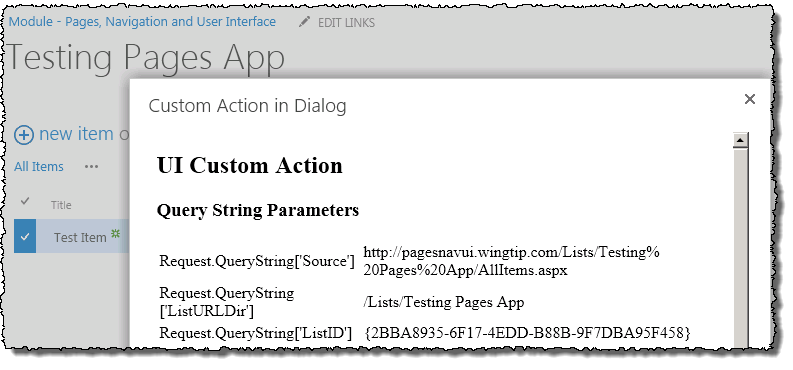
HostWebDialogWidth="600"

HostWebDialogHeight="400"

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://pagesnavui.wingtip.com> and click the **Site Contents** link in the left-hand navigation.
4. Click the **Testing Pages App** link in the left-hand navigation
5. Click the edit control block (ECB) menu (the ‘**…**’) of item you created previously and click **Custom Action Dialog**:
6. Notice the browser will open the page from the remote site in a dialog window:



1. Close the browser to stop the debugger.

In this exercise you created a few navigation components to see how things worked.