## Programming with the SharePoint 2013 REST API

**Lab Time**: 60 minutes

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

**Lab Overview**: In this lab you will learn how to call into the SharePoint 2013 REST API using JavaScript and jQuery. You will learn how to call into this API using the jQuery library and to process JSON-based results. You will also learn how to leverage the jsRender library to create HTML output using a template-based approach.

### 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://clientside.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:

[..]\DevClientSide

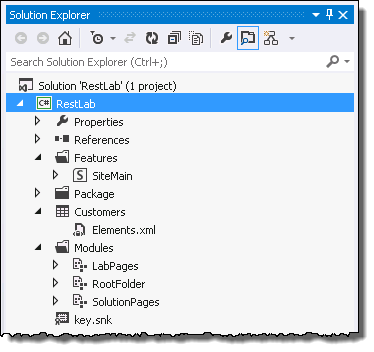
* 1. When the script completes, it will launch a new browser and navigate to the lab site collection.
  2. Close the PowerShell console window.

### Exercise 2: Calling the SharePoint 2013 REST API

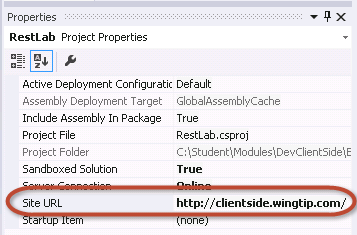
In this exercise, you will have a walkthrough which shows how to form URLs to make REST API calls using the browser.

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:

[..]\DevClientSide\Exercises\Ex2\RestLab.sln

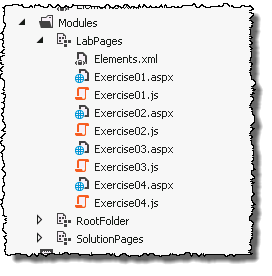


1. Ensure the project is using the lab’s site collection for testing:
   1. Select the project **RestLab** in the **Solution Explorer** tool window.
   2. Look in the **Properties** tool window. If the **Site URL** property does not show <http://clientside.wingtip.com>, update it so it does.



#### Tour of the Provided Visual Studio Solution

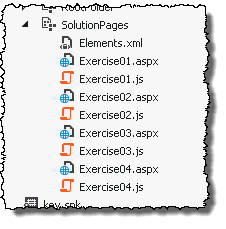
1. Note there is a project item to create a list named **Customers**. When you deploy the project later in this exercise, it creates a Contacts list with the title of **Customers** and adds a few default items that will be used in testing your code.
2. Using the **Solution Explorer** tool window, within the **RestLab** project, expand the **Modules** folder.
3. Inside the **Modules** folder, there are three existing modules named **LabPages**, **RootFolder** and **SolutionPage**.
   1. Expand the Module named **LabPages** and see what files are inside. There are four site pages and a JavaScript file associated with each of the site pages. Most of the work you will do in this lab will involve writing JavaScript code inside these JavaScript source files to retrieve data using the CSOM / REST API.



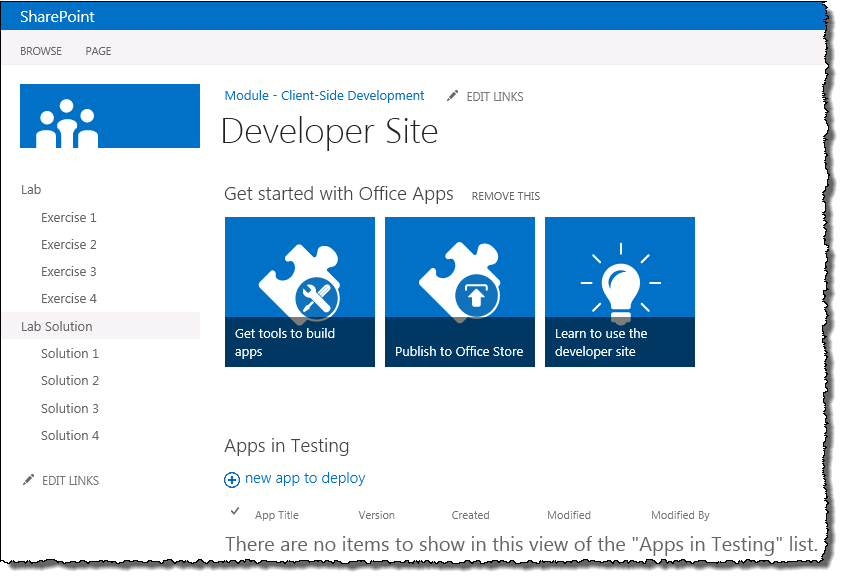
* 1. Expand the Module named **RootFolder** and see what files are inside. This modules contains JavaScript libraries for **jquery.js**, **jquery-ui.js**, **datajs.js** and **jsrender.js**. There is also a **css** folder with .css files and images which are part of the jQuery UI library. Note that the links to these library source files have already been added to the pages you will be working with. Therefore, the code you write in this lab can automatically use the jQuery and jQuery UI libraries without you having to worry about linking to their source files.



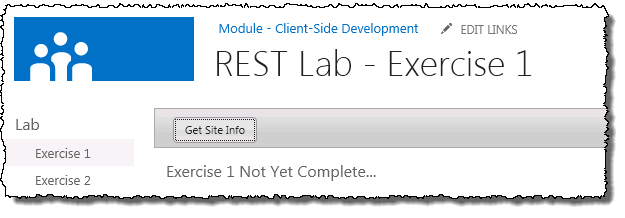
* 1. Expand the Module named **SolutionPages** and see what files are inside. There are four site pages and an associated JavaScript file for each site page. The four JavaScript files represent the solutions to the exercises for this lab.



1. The **RestLab** project contains a feature named **MainSite** which has an associated feature receiver. When this features activates, the code inside the feature receiver creates navigation links to the pages for the lab as well as the solution pages.
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 homepage of the site. Notice all the links in the Quick Launch navigation:



1. Using the Quick Launch navigation, select **Exercise 1** to navigate to the page **Exercise1.aspx**.
   1. There is a command button on the page. Click the **Get Site Info** button and notice that the code currently behind this button writes some text to an element on the page.



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

#### Use jQuery and the REST API to Display Site Properties

1. Using the **Solution Explorer** tool window, find the file **Modules \ LabPages \ Exercise1.aspx** file. Right-click the **Exercise01.aspx** file and select **Open**.
   1. Notice that the placeholder named **PlaceHolderAdditionalPageHead** already contains links to **jquery.js**, **jquery-ui.js** and the JavaScript file you will be editing in this exercise named **Exercise01.js**.

<SharePoint:ScriptLink Name="~sitecollection/js/jquery.js" Defer="false" runat="server" />

<SharePoint:ScriptLink Name="~sitecollection/js/jquery-ui.js" Defer="false" runat="server" />  
<script src="Exercise01.js" type="text/javascript" ></script>

* 1. Also note that **Exercise1.aspx** contains a placeholder named **PlaceHolderMain** which contains a **<input>** button with an **id** of **cmdGetSiteInfo** which acts as a command button. It also has a **<div>** with an **id** of **results**. When you write your code over the next few steps, you use this **<div>** display content on the page.

<asp:Content ContentPlaceHolderId="PlaceHolderMain" runat="server">

<div id="toolbar" class="ui-widget-header" >

<input id="cmdGetSiteInfo" type="button" value="Get Site Info" />

</div>

<div id="results" />

</asp:Content>

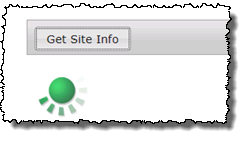
1. Update **Exercise1.aspx** to use jQuery to update the user interface:
   1. Using the **Solution Explorer** tool window, find the file **Modules \ LabPages \ Exercise01.js** file. Right-click the **Exercise01.js** file and select **Open**.
   2. Delete the contents of the **onGetSiteInfo()** function but leave the other code on the page.
   3. Add the following JavaScript code to the **onGetSiteInfo()** function. It will first clear the contents of the results **<div>** and then add the SharePoint animated gears icon.

// clear resultsArea and add spinning gears icon

$("#results").empty();

$("<img>", { "src": "/\_layouts/images/GEARS\_AN.GIF" }).appendTo("#results");

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 site.
3. Using the Quick Launch navigation, select **Exercise 1**.
4. Click the **Get Site Info** button. Notice it is now showing the working animated gears image:



1. Close the browser to stop the debugger and go back to Visual Studio.
2. Still in the **Exercise01.js** file, find the **onGetSiteInfo()** function.
3. Add the following code to the end of the **onGetSiteInfo()** function to create a reference to the REST endpoint for the current SharePoint site. Notice you are using the **\_spPageContextInfo** variable that SharePoint adds to every page.

var requestUri = \_spPageContextInfo.webAbsoluteUrl + "/\_api/web";

1. Next, execute an asynchronous request which formats the result in JSON format by calling the jQuery function named **$.ajax()**. This method will pass in a callback where your code will run when the response is received. Add the following code to the end of the **onGetSiteInfo()** function:

// execute AJAX request

var requestHeaders = { "accept": "application/json;odata=verbose" }

$.ajax({

url: requestUri,

headers: requestHeaders,

success: onDataReturned,

error: onError

});

1. Now implement the callback by adding a new function **onDataReturned()**. This will take the JSON formatted results and write them out to the page. Add the following code to the **Example01.js** file:

function onDataReturned(data) {

$("#results").empty();

var odataResults = data.d;

$("<h2>").html("Site Properties").appendTo("#results");

$("<p>").html("<b>Id</b>: " + odataResults.Id).appendTo("#results");

$("<p>").html("<b>Url</b>: " + odataResults.Url).appendTo("#results");

$("<p>").html("<b>Title</b>: " + odataResults.Title).appendTo("#results");

$("<p>").html("<b>MasterUrl</b>: " + odataResults.MasterUrl).appendTo("#results");

$("<p>").html("<b>Language</b>: " + odataResults.Language).appendTo("#results");

}

1. Implement the **onError()** function as follows to display the error message to the page. Add the following code to **Example01.js**:

function onError(err) {

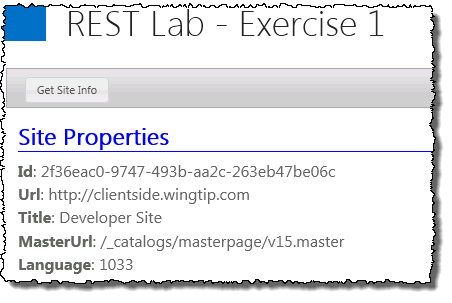
$("#results").text("ERROR: " + JSON.stringify(err));

}

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 site.
3. Using the Quick Launch navigation, select **Exercise 1**.
4. Click the **Get Site Info** button. Notice it is now showing the working animated gears image for a moment while it fetches the site details:



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

At this point you have successfully made your first REST call into the SharePoint 2013 CSOM using JavaScript and jQuery.

### Exercise 3: Executing OData Queries to Discover the Lists

In this exercise, you will call into the REST APIs as you did in the previous exercise. However, now you will deal with a JSON result set that has mutliple items. In particular, you must write the code to enumerate through the existing set of lists in the current site.

1. If you want to continue working with the solution from the previous example, you can skip this step. Otherwise you can follow these steps to open a clean project which effectively is where you left off if you had completed the previous exercise as outlined:
   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:

[..]\DevClientSide\Exercises\Ex3\RestLab.sln

1. Using the **Solution Explorer** tool window, find the file **Modules \ LabPages \ Exercise02.aspx** file. Right-click the **Exercise02.aspx** file and select **Open**.
   1. This file is setup the same way as the **Exercise1.aspx** page was setup.
2. Using the **Solution Explorer** tool window, find the file **Modules \ LabPages \ Exercise02.js** file. Right-click the **Exercise02.js** file and select **Open**.
3. Open the JavaScript source file named **Exercise02.js** and inspect the code inside.
   1. Delete the contents of the **onGetLists()** function but leave the other code on the page.
4. Add the following JavaScript code to the **onGetLists()** function. It will first clear the contents of the results **<div>** and then add the SharePoint animated gears icon.

// clear results area and add spinning gears icon

$("#results").empty();

$("<img>", { "src": "/\_layouts/images/GEARS\_AN.GIF" }).appendTo("#results");

1. Next, add the following code to the end of the **onGetLists()** function to create a reference to the REST endpoint for the current SharePoint site.

var requestUri = \_spPageContextInfo.webAbsoluteUrl + "/\_api/web/lists";

1. Next, execute an asynchronous request which formats the result in JSON format by calling the jQuery function named **getJSON()**. This method will pass in a callback where your code will run when the response is received. Add the following code to the end of the **onGetLists()** function:

// execute AJAX request

var requestHeaders = { "accept": "application/json;odata=verbose" }

$.ajax({

url: requestUri,

headers: requestHeaders,

success: onDataReturned,

error: onError

});

1. Now implement the callback by adding a new function **onDataReturned()**. This will take the JSON formatted results and write them out to the page. Add the following code to the **Example02.js** file:

function onDataReturned(data) {

$("#results").empty();

var odataResults = data.d.results;

$("<h2>").html("Lists in this site").appendTo("#results");

var ul = $("<ul>");

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

$("<li>").html(odataResults[i].Title).appendTo(ul);

}

ul.appendTo("#results");

}

1. Implement the **onError()** function as follows to display the error message to the page. Add the following code to **Example02.js**:

function onError(err) {

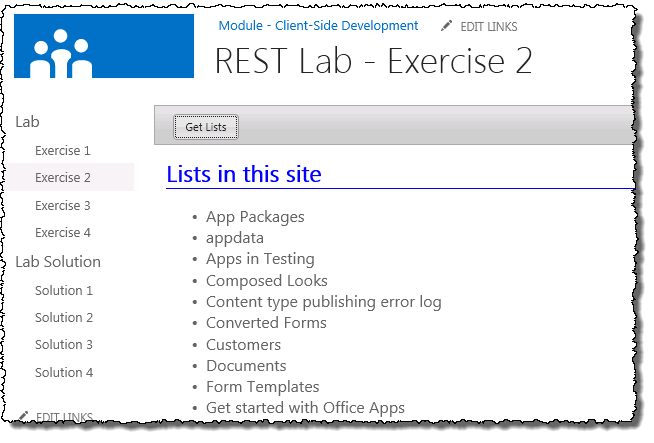
$("#results").text("ERROR: " + JSON.stringify(err));

}

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 site.
3. Using the Quick Launch navigation, select **Exercise 2**.
4. Click the **Get Lists** button. Notice it is now showing the working animated gears image for a moment while it fetches the lists:



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

In this exercise you queried the REST API to get a collection of all lists in the SharePoint site.

### Exercise 4: Using OData Filters and Templates

In this exercise, you will make calls to the REST APIs using the jQuery **getJSON** function as you did in the previous exercise. However, when calling into the REST API to get the list of the site, you will add a filter expression to filter out hidden system lists such as the MasterPage Gallery and the List Template Gallery. You will also begin working with a JavaScript utility library named jsRender which makes it much easier to parse a JSON resultset into a complex HTML element for display.

1. If you want to continue working with the solution from the previous example, you can skip this step. Otherwise you can follow these steps to open a clean project which effectively is where you left off if you had completed the previous exercise as outlined:
   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:

[..]\DevClientSide\Exercises\Ex4\RestLab.sln

1. Using the **Solution Explorer** tool window, find the file **Modules \ LabPages \ Exercise03.aspx** file. Right-click the **Exercise03.aspx** file and select **Open**.
   1. This file is setup the same way as the **Exercise1.aspx** page was setup.
   2. One important difference is that the page contains an extra **ScriptLink** control which links to the JavaScript source file for the jsRender library.

<SharePoint:ScriptLink Name="~sitecollection/js/jsrender.js" Defer="false" runat="server" />

1. Using the **Solution Explorer** tool window, find the file **Modules \ LabPages \ Exercise03.js** file. Right-click the **Exercise03.js** file and select **Open**.
2. Open the JavaScript source file named **Exercise03.js** and inspect the code inside.
   1. Delete the contents of the **onGetUserLists()** function but leave the other code on the page.
3. Add the following JavaScript code to the **onGetUserLists()** function. It will first clear the contents of the results **<div>** and then add the SharePoint animated gears icon.

// clear resultsArea and add spinning gears icon

$("#results").empty();

$("<img>", { "src": "/\_layouts/images/GEARS\_AN.GIF" }).appendTo("#results");

1. Next, add the following code to the end of the **onGetUserLists()** function to create a reference to the REST endpoint for the current SharePoint site. Notice this code will add an extra OData command to the service to filter out all hidden lists:

var requestUri = \_spPageContextInfo.webAbsoluteUrl + "/\_api/web/lists?$filter=Hidden eq false";

1. Next, execute an asynchronous request which formats the result in JSON format by calling the jQuery function named **getJSON()**. This method will pass in a callback where your code will run when the response is received. Add the following code to the end of the **onGetUserLists()** function:

// execute AJAX request

var requestHeaders = { "accept": "application/json;odata=verbose" }

$.ajax({

url: requestUri,

headers: requestHeaders,

success: onDataReturned,

error: onError

});

1. Now implement the callback by adding a new function **onDataReturned()**. This will take the JSON formatted results and write them out to the page. You are going to do things differently from the previous exercise because you are going to leverage the jsRender library to create a HTML ordered list from the JavaScript array held within a JSON result set. Complete the implementation of **onDataReturned()** using the following code which creates a jsRender template using the **$.template** method and generates formatted output from the array and the template using the **$.render** method. Add the following code to the **Example03.js** file:

function onDataReturned(data) {

$("#results").empty();

var odataResults = data.d.results;

$("<h2>").html("Lists in this site").appendTo("#results");

// create a template using jsRender library

renderingTemplate = "<li>{{=Title}}</li>";

$.template("tmplLists", renderingTemplate);

// generate output from array using template

var ol = $("<ol>");

ol.append($.render(odataResults, "tmplLists"));

$("#results").append(ol);

}

1. Implement the **onError()** function as follows to display the error message to the page. Add the following code to **Example03.js**:

function onError(err) {

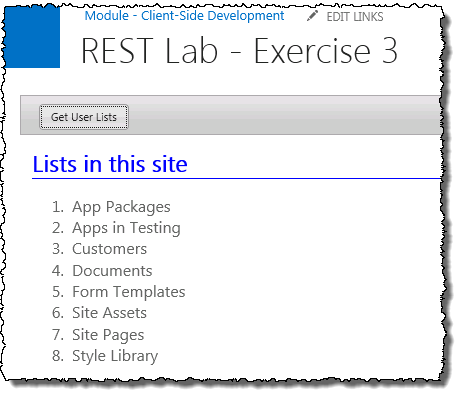
$("#results").text("ERROR: " + JSON.stringify(err));

}

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 site.
3. Using the Quick Launch navigation, select **Exercise 3**.
4. Click the **Get User Lists** button. Notice it is now showing the working animated gears image for a moment while it fetches the lists:



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

In this exercise you have learned how to add filtering expression in the query string of the REST URI as well as how to leverage the jsRender library to generate HTML output from a OData result set using templates.

### Exercise 5: Querying SharePoint List Items

In this exercise, you will use all the skills gained in the previous exercises to read and display the list items from a list. You will use the REST API to run a query against a specific list which returns a specific set of columns. You will use the jsRender library and a template to format the item content as a HTML list. Then you will get more involved by creating a second template that displays the same list content as an HTML table.

1. If you want to continue working with the solution from the previous example, you can skip this step. Otherwise you can follow these steps to open a clean project which effectively is where you left off if you had completed the previous exercise as outlined:
   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:

[..]\DevClientSide\Exercises\Ex5\RestLab.sln

1. Using the **Solution Explorer** tool window, find the file **Modules \ LabPages \ Exercise04.aspx** file. Right-click the **Exercise04.aspx** file and select **Open**.
   1. This file is setup the same way as the **Exercise1.aspx** page was setup.
   2. Notice though that this file has two buttons in it.
2. Using the **Solution Explorer** tool window, find the file **Modules \ LabPages \ Exercise04.js** file. Right-click the **Exercise04.js** file and select **Open**.

#### Implement the Get Customer List Button

1. Open the JavaScript source file named **Exercise04.js** and inspect the code inside.
   1. Delete the contents of the **onGetCustomerList()** function but leave the other code on the page.
2. Add the following JavaScript code to the **onGetCustomerList()** function. It will first clear the contents of the results **<div>** and then add the SharePoint animated gears icon.

// clear resultsArea and add spinning gears icon

$("#results").empty();

$("<img>", { "src": "/\_layouts/images/GEARS\_AN.GIF" }).appendTo("#results");

1. Next, add the following code to the end of the **onGetCustomerList()** function to create a reference to the REST endpoint for the current SharePoint site. Notice this code will use a new URL format to lookup values for a specific list and only retrieve specific properties from the list:

var requestUri = \_spPageContextInfo.webAbsoluteUrl + "/\_api/web/lists/GetByTitle('Customers')/Items?$select=Title,FirstName,WorkPhone";

1. Next, execute an asynchronous request which formats the result in JSON format by calling the jQuery function named **getJSON()**. This method will pass in a callback where your code will run when the response is received. Add the following code to the end of the **onGetCustomerList()** function:

// execute AJAX request

var requestHeaders = { "accept": "application/json;odata=verbose" }

$.ajax({

url: requestUri,

headers: requestHeaders,

success: onListDataReturned,

error: onError

});

1. Now implement the callback by adding a new function **onListDataReturned()**.Add the following code to the **Example04.js** file:
2. Implement the **onListDataReturned** function as follows to create an HTML list from the items in the customers list which displays customers in a *{Last Name, First Name}* format.

function onListDataReturned(data) {

$("#results").empty();

var odataResults = data.d.results;

$("<h2>").html("Customers").appendTo("#results");

// set rendering template

var renderingTemplate = "<li>{{=Title}}, {{=FirstName}}</li>";

$.template("tmplLists", renderingTemplate);

$("#results").append($("<ul>").append($.render(odataResults, "tmplLists")));

}

1. Implement the **onError()** function as follows to display the error message to the page. Add the following code to **Example04.js**:

function onError(err) {

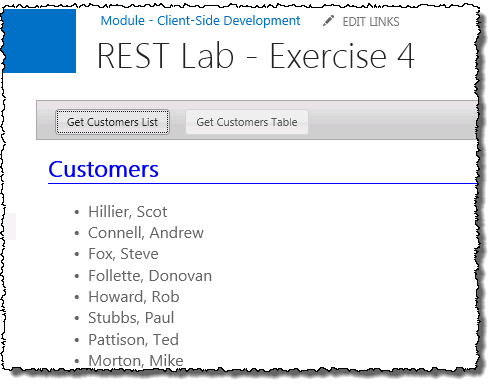
$("#results").text("ERROR: " + JSON.stringify(err));

}

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 site.
3. Using the Quick Launch navigation, select **Exercise 4**.
4. Click the **Get Customer List** button. Notice it is now showing the working animated gears image for a moment while it fetches the lists:



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

#### Implement the Get Customer Table Button

1. Open the JavaScript source file named **Exercise04.js** and inspect the code inside.
   1. Delete the contents of the **onGetCustomerTable()** function but leave the other code on the page.
2. Add the following JavaScript code to the **onGetCustomerTable()** function. It will first clear the contents of the results **<div>** and then add the SharePoint animated gears icon.

// clear resultsArea and add spinning gears icon

$("#results").empty();

$("<img>", { "src": "/\_layouts/images/GEARS\_AN.GIF" }).appendTo("#results");

1. Next, add the following code to the end of the **onGetCustomerTable()** function to create a reference to the REST endpoint for the current SharePoint site. Notice this code will use a new URL format to lookup values for a specific list and only retrieve specific properties from the list:

var requestUri = \_spPageContextInfo.webAbsoluteUrl + "/\_api/web/lists/GetByTitle('Customers')/Items?$select=Title,FirstName,WorkPhone";

1. Next, execute an asynchronous request which formats the result in JSON format by calling the jQuery function named **getJSON()**. This method will pass in a callback where your code will run when the response is received. Add the following code to the end of the **onGetCustomerTable()** function:

// execute AJAX request

var requestHeaders = { "accept": "application/json;odata=verbose" }

$.ajax({

url: requestUri,

headers: requestHeaders,

success: onTableDataReturned,

error: onError

});

1. Now implement the callback by adding a new function **onTableDataReturned()**.Add the following code to the **Example04.js** file:
2. Implement the **onTableDataReturned** function as follows to create an HTML table from the items in the customers list which displays customers in a *{Last Name, First Name}* format.

function onTableDataReturned(data) {

$("#results").empty();

var odataResults = data.d.results;

$("<h2>").html("Customers").appendTo("#results");

// set rendering template

var tableHeader = "<thead>" +

"<td>Last Name</td>" +

"<td>First Name</td>" +

"<td>Work Phone</td>" +

"</thead>";

var table = $("<table>").append($(tableHeader));

var renderingTemplate = "<tr>" +

"<td>{{=Title}}</td>" +

"<td>{{=FirstName}}</td>" +

"<td>{{=WorkPhone}}</td>" +

"</tr>";

$.template("tmplTable", renderingTemplate);

table.append($.render(odataResults, "tmplTable"));

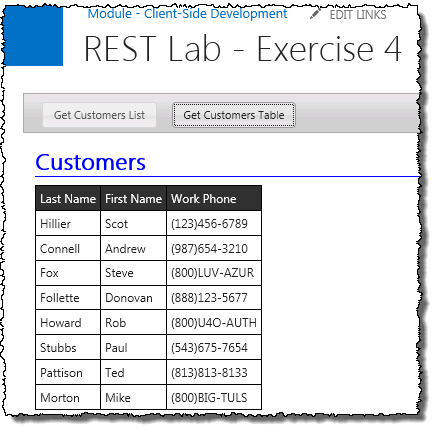
$("#results").append(table);

}

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 site.
3. Using the Quick Launch navigation, select **Exercise 4**.
4. Click the **Get Customer Table** button. Notice it is now showing the working animated gears image for a moment while it fetches the lists:



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

In this lab, you learned how to call into the SharePoint 2013 REST API using JavaScript and the jQuery. You learned how to call into this API using the getJSON function of the jQuery library and to process JSON-based results. You also learned how to leverage the jsRender library to create HTML output using a template-based approach.