

# Programming with the SharePoint REST API

**Lab Time:** 60 minutes

**Lab Folder:** C:\Student\Modules\04\_SharePointRestAPI\Lab

**Lab Overview:** In this lab you will get hands-on experience writing code in a SharePoint-hosted app which programs against the SharePoint REST API. Over the course of all the exercises in this lab, you will write all the code that's required to create a reusable JavaScript module named **Wingtip.Customers.DataAccess** which takes care of performing CRUD operations (i.e. create, read, update and delete) on items in a SharePoint list in the App Web.

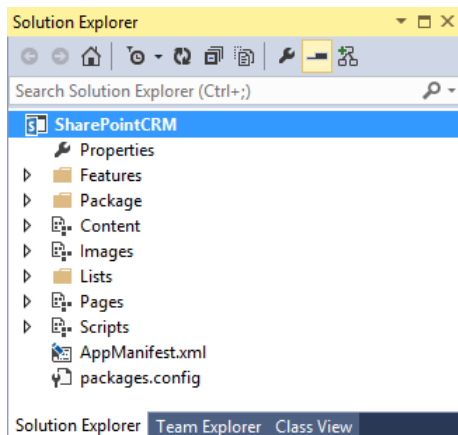
## Exercise 1: Getting the SharePointCRM Starter Project Up and Running

In this exercise you open an existing Visual Studio project for a SharePoint-Hosted App and make sure you can start it up and test in in the Visual Studio debugger.

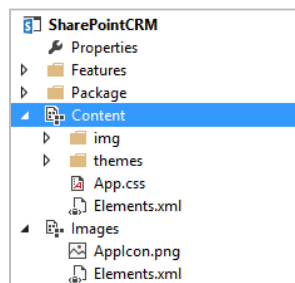
1. Launch **Visual Studio** as administrator:
2. Open the **SharePointCRM** starter project.
  - a) In the top menu of Visual Studio, select **File → Open → Project**.
  - b) When the **Open Project** dialog appears, select the Visual Studio solution file at the following path and click **OK**.

**C:\Student\Modules\04\_SharePointRestAPI\Lab\StarterProject\SharePointCRM\SharePointCRM.sln**

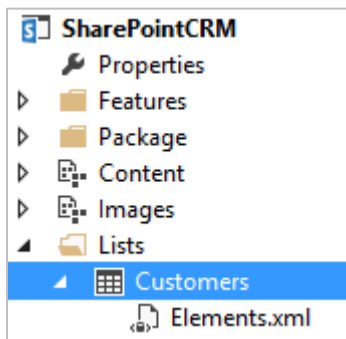
- c) When the **SharePointCRM** project opens, you should see it in the Solution Explorer.



3. Examine the structure of the **SharePointCRM** project.
  - a) There is a **Content** folder and an **Images** folder with CSS files and images. Note that you will not be required to modify these two folders in any way during this lab.



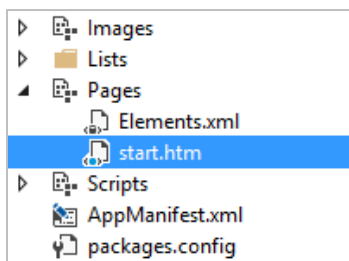
- b) If you expand the **Lists** folder, you can see that this app project includes a list named **Customers** that will be created in the App Web during the installation of the app. The **Customers** list is based on the standard **Contacts** list type.



- c) Open up the **Elements.xml** file associated with the **Customers** list so you can see the *ListInstance* element which creates the list and adds in a set of 16 sample customer items.

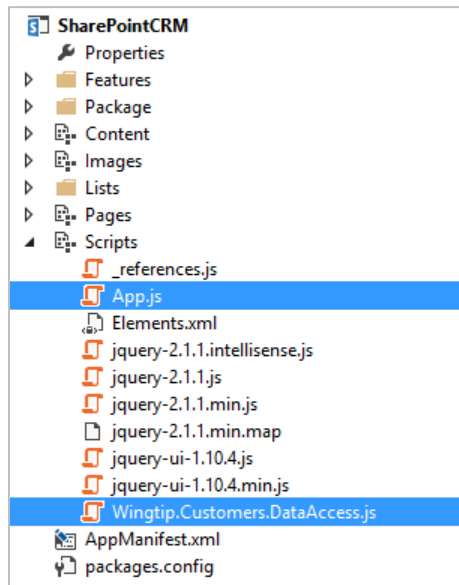
```
<?xml version="1.0" encoding="utf-8"?>
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
  <ListInstance Title="Customers"
    OnQuickLaunch="TRUE"
    TemplateType="105"
    FeatureId="00bfea71-7e6d-4186-9ba8-c047ac750105"
    Url="Lists/Customers" >
    <Data>
      <ROWS>
        <ROW>
          <Field Name="FirstName">Quincy</Field>
          <Field Name="Title">Nelson</Field>
          <Field Name="Company">Benthic Petroleum</Field>
          <Field Name="WorkPhone">1(340)608-7748</Field>
          <Field Name="HomePhone">1(340)517-3737</Field>
          <Field Name="Email">Quincy.Nelson@BenthicPetroleum.com</Field>
        </Row>
        <ROW>
          <Field Name="FirstName">Jude</Field>
          <Field Name="Title">Mason</Field>
          <Field Name="Company">Cyberdyne Systems</Field>
          <Field Name="WorkPhone">1(203)408-0466</Field>
          <Field Name="HomePhone">1(203)411-0071</Field>
          <Field Name="Email">Jude.Mason@CyberdyneSystems.com</Field>
        </Row>
        <!--16 new items in all -->
      </ROWS>
    </Data>
  </ListInstance>
</Elements>
```

- d) When you are done, close the **Elements.xml** file associated with the **Customers** list and be sure *not* to save any changes.
- e) Look inside the **Pages** folder and you will see a single page named **start.htm**.



- f) Open up **start.htm** and inspect the HTML inside. Note that you are not required to update **start.htm** at all during this lab.

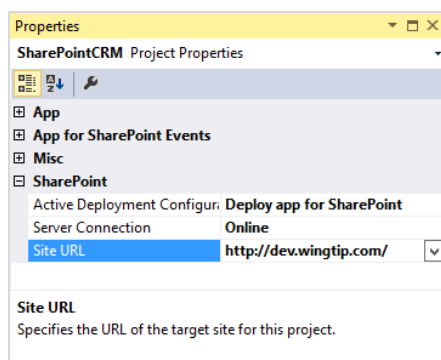
- g) Close **start.htm** and do not save any changes.
- h) Expand the **Scripts** folder and locate the two JavaScript files named **App.js** and **Wingtip.Customers.DataAccess.js**. These two sources files is where you will do all your working in the following exercises.



- i) Open **Wingtip.Customers.DataAccess.js** and take a quick look at what's inside without making any changes.
- j) Open **App.js** and take a quick look at what's inside without making any changes.

Up to this point, you have not been required to actually change anything in the project. You have just been looking at it to get familiar with the folders and files inside. Now as a final part to the first exercise, you will start up the **SharePointCRM** project in the Visual Studio debugger to see how it appears and behaves in its initial state.

- 4. Launch the **SharePointCRM** project in the Visual Studio debugger.
  - a) Select the **SharePointCRM** project in the Solution Explorer.
  - b) In the Project Properties window for the **SharePointCRM** project, set the **Site URL** to **http://dev.wingtip.com**.

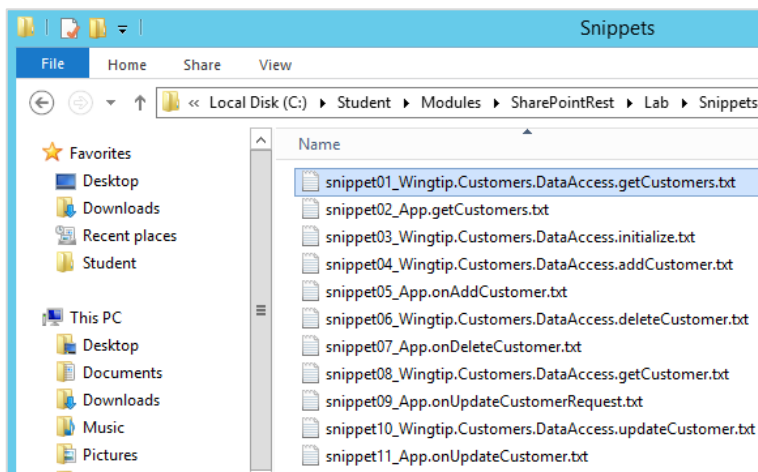


- c) Press the {F5} key to start a Visual Studio debugging session and wait for the app start page to appear. It should appear with a page that has a styled page layout but the main content area of the page is empty.



- d) Close the browser window and stop the Visual Studio debugging session.

The following exercises in this lab involve quite a bit of coding. Some of our students like to type in everything, even if that means typing in 100s of lines of code. Other students aren't as excited about typing in so many lines of code. These students often prefer to copy-and-paste code from snippets files. It's your choice as to whether you use the copy-and-paste text files in the **Snippets** folder for this lab at **C:\Student\Modules\SharePointRest\Lab\Snippets**.



## Exercise 2: Querying a SharePoint List using the SharePoint REST API

In this exercise you will implement the **getCustomers** function in **Wingtip.Customers.DataAccess.js** to execute a REST call which returns all the items inside the **Customers** list. You will then modify the **getCustomers** function in **App.js** to update the UI by displaying the returned ODATA results on the page.

1. Open **Wingtip.Customers.DataAccess.js** and locate the **getCustomers** function.
2. Implement **getCustomers** using the following code. Remember that you can also copy-and-paste the same code from the snippet file named **Snippet01\_Wingtip.Customers.DataAccess.getCustomers.txt**.

```
var getCustomers = function () {  
    var requestUri = "../_api/web/lists/getByTitle('Customers')/items" +  
        "?$select=ID,FirstName,Title,Company,WorkPhone,HomePhone,Email" +  
        "&$orderby=Title,FirstName";  
  
    // send call across network  
    return $.ajax({  
        url: requestUri,  
        headers: { "accept": "application/json;odata=verbose" }  
    });  
};
```

3. Implement **getCustomers** in **App.js** using the following code. Remember that you can also copy-and-paste the same code from the snippet file named **snippet02\_App.getCustomers.txt**.

```
function getCustomers() {  
    // clear results and add spinning gears icon  
    $("#content_box").empty();  
    $("<img>", { "src": "../Content/img/GEARS.gif" }).appendTo("#content_box");  
  
    // call view-model function which returns promise  
    var promise = Wingtip.Customers.DataAccess.getCustomers()  
  
    // use promise to implement what happens when OData result is ready  
    promise.then(onGetCustomersComplete, onError);  
}
```

There is no need to make any more changes before testing the app. However, before running the app for the first time you should take a look at the **onGetCustomersComplete** function in **App.js** so you can better understand what this function does and how it will take the ODATA results from your SharePoint REST API call and use these results to display **Customer** items on the page. Keep in mind you are just looking in the next step and that you will not make any changes to the **onGetCustomersComplete** function.

4. Inspect the implementation of the **onGetCustomersComplete** function in **App.js**.

- a) The **onGetCustomersComplete** accepts a single parameter named **data** which is used to access the ODATA results. The ODATA results are accessed using the syntax **data.d.results** and assigned to a variable named **customers**.

```
function onGetCustomersComplete(data) {  
    $("#content_box").empty();  
    var customers = data.d.results;  
    // more to follow  
}
```

- b) Next, there is some jQuery code which creates an HTML table and add a set of column headers.

```
var table = $("<table>", { ID: "customersTable" });  
table.append($("<thead>")  
    .append($("<td>").html("&nbsp;"))  
    .append($("<td>").html("&nbsp;"))  
    .append($("<td>").text("First Name"))  
    .append($("<td>").text("Last Name"))  
    .append($("<td>").text("Company"))  
    .append($("<td>").text("Work Phone"))  
    .append($("<td>").text("Home Phone"))  
    .append($("<td>").text("Email"))));
```

- c) Next, there is code to enumerate through each customer item in the ODATA result and to create the table cells required to create a complete table row. Note that the code for the first two columns are tricky. This code must create HTML Hyperlink elements which display custom images and call two JavaScript functions in **App.js** named **onUpdateCustomerRequest** and **onDeleteCustomer**.

```
for (var customer = 0; customer < customers.length; customer++) {  
    var linkEditUrl = "javascript: onUpdateCustomerRequest(" + customers[customer].Id + ")";  
    var linkEdit = $("<a>", { "href": linkEditUrl })  
        .append($("<img>", { "src": "../Content/img/EDITITEM.gif", "alt": "Edit" }));  
  
    var linkDeleteUrl = "javascript: onDeleteCustomer(" + customers[customer].Id + ")";  
    var linkDelete = $("<a>", { "href": linkDeleteUrl })  
        .append($("<img>", { "src": "../Content/img/DELITEM.gif", "alt": "Delete" }));  
  
    table.append($("<tr>")  
        .append($("<td>").append(linkEdit))  
        .append($("<td>").append(linkDelete))  
        .append($("<td>").text(customers[customer].FirstName))  
        .append($("<td>").text(customers[customer].Title))  
        .append($("<td>").text(customers[customer].Company))  
        .append($("<td>").text(customers[customer].WorkPhone))
```

```
.append($"<td>").text(customers[customer].HomePhone))
.append($"<td>").text(customers[customer].Email));
}
```

- d) If the SharePoint REST API call to retrieve the items in the customers fails, there is a callback which will call the **onError** function which will simply display the error information on the start page. While this isn't the type of code you would usually include in a production app, it's helpful to have a way to quickly visually what has gone wrong.

```
function onError(error) {
    $("#content_box").empty();
    $("#content_box").text("Error: " + JSON.stringify(error));
}
```

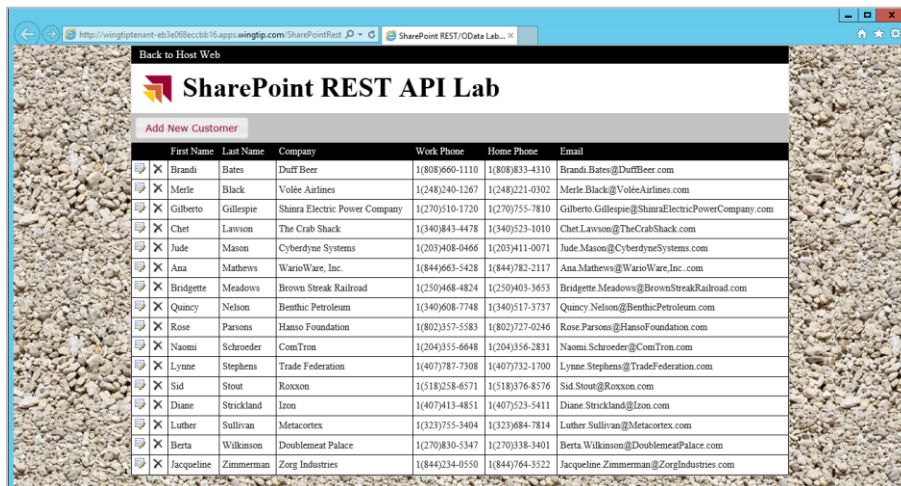
- e) Finally, there is line of code which appends the HTML table of customers to the app's start page.

```
// append table to div in DOM
$("#content_box").append(table);
```

- f) Now you should have a better understanding of how the ODATA results will be displayed on the start page.

5. Test your work by starting a new Visual Studio debugging session.

- Make sure you are in Visual Studio and that you have the **SharePointCRM** project open.
- Press the **{F5}** key to begin a debugging session and wait for the start page to appear.
- You should see the page update to display the table of customers.



The screenshot shows a web browser displaying the 'SharePoint REST API Lab' application. At the top, there is a 'Back to Host Web' link and a title 'SharePoint REST API Lab'. Below the title is a button labeled 'Add New Customer'. The main content area displays a table with customer information. The table has columns for First Name, Last Name, Company, Work Phone, Home Phone, and Email. Each row represents a customer and includes a small 'X' icon in the first column.

First Name	Last Name	Company	Work Phone	Home Phone	Email
Brandi	Bates	Duff Beer	1(800)660-1110	1(800)833-4310	Brandi.Bates@DuffBeer.com
Merle	Black	Volée Airlines	1(248)240-1267	1(248)321-0302	Merle.Black@VoléeAirlines.com
Gilberto	Gillespie	Shura Electric Power Company	1(270)510-1720	1(270)755-7810	Gilberto.Gillespie@ShuraElectricPowerCompany.com
Chet	Lawson	The Crab Shack	1(340)843-4478	1(340)523-1010	Chet.Lawson@TheCrabShack.com
Jude	Mason	Cyberdyne Systems	1(203)408-0466	1(203)411-0071	Jude.Mason@CyberdyneSystems.com
Ana	Mathews	WarioWare, Inc.	1(844)663-5428	1(844)782-2117	Ana.Mathews@WarioWare,Inc.com
Bridgette	Meadows	Brown Streak Railroad	1(250)468-4824	1(250)403-3653	Bridgette.Meadows@BrownStreakRailroad.com
Quincy	Nelson	Benthic Petroleum	1(340)608-7748	1(340)517-3737	Quincy.Nelson@BenthicPetroleum.com
Rose	Parsons	Hanso Foundation	1(802)357-5583	1(802)727-0246	Rose.Parsons@HansoFoundation.com
Naomi	Schroeder	ComTron	1(204)355-6648	1(204)356-2831	Naomi.Schroeder@ComTron.com
Lynne	Stephens	Trade Federation	1(407)787-7308	1(407)732-1700	Lynne.Stephens@TradeFederation.com
Sid	Stout	Roxxon	1(518)258-6571	1(518)376-8576	Sid.Stout@Roxxon.com
Diane	Strickland	Izon	1(407)413-4851	1(407)523-5411	Diane.Strickland@Izon.com
Luther	Sullivan	Metacortex	1(323)755-3404	1(323)684-7814	Luther.Sullivan@Metacortex.com
Berta	Wilkinson	Doublemeat Palace	1(270)830-5347	1(270)338-3401	Berta.Wilkinson@DoublemeatPalace.com
Jacqueline	Zimmerman	Zorg Industries	1(844)234-0550	1(844)764-3522	Jacqueline.Zimmerman@ZorgIndustries.com

6. Close the browser and stop your Visual Studio debugging session.

Now you have succeeded in executing a SharePoint REST API call for a read-only query. In the next exercise you will execute SharePoint REST API calls that will modify data. This means that you must begin to work with the Request Digest.

### Exercise 3: Adding and Deleting Items with the SharePoint REST API

You will begin this exercise by writing the code to retrieve and cache a Request Digest value. The a Request Digest value is required whenever you are making SharePoint REST API calls which modify anything in the SharePoint host including list items inside the Customers list. After you have written code to manage the Request Digest value, you will then add code to implement INSERT and DELETE behavior in the SharePointCRM app.

- Open **Wingtip.Customers.DataAccess.js** and locate the **initialize** function. You should observe that there is a module-level variable named **requestDigest** that is defined just before the **initialize** function. The **requestDigest** variable will be used to store and cache a Request Digest value that is retrieved from the SharePoint host.

```
var requestDigest;
```

```
var initialize = function () {  
    // TODO: snippet03_Wingtip.Customers.DataAccess.initialize.txt  
}
```

2. Implement the **initialize** function using the following code. Remember that you can also copy-and-paste the same code from the snippet file named **snippet03\_Wingtip.Customers.DataAccess.initialize.txt**.

```
var initialize = function () {  
    var deferred = $.ajax({  
        url: "../_api/contextinfo",  
        type: "POST",  
        headers: { "accept": "application/json;odata=verbose" }  
    })  
  
    deferred.then(function (data) {  
        requestDigest = data.d.GetContextWebInformation.FormDigestValue  
    });  
}
```

3. Implement **addCustomer** using the following code. Remember that you can also copy-and-paste the same code from the snippet file named **snippet04\_Wingtip.Customers.DataAccess.addCustomer.txt**.

```
var addCustomer = function (FirstName, LastName, Company, WorkPhone, HomePhone, Email) {  
    var requestUri = "../_api/web/lists/getByTitle('Customers')/items";  
  
    var requestHeaders = {  
        "accept": "application/json;odata=verbose",  
        "X-RequestDigest": requestDigest  
    }  
  
    var customerData = {  
        __metadata: { "type": "SP.Data.CustomersListItem" },  
        Title: LastName,  
        FirstName: FirstName,  
        Company: Company,  
        WorkPhone: WorkPhone,  
        HomePhone: HomePhone,  
        Email: Email  
    };  
  
    var requestBody = JSON.stringify(customerData);  
  
    return $.ajax({  
        url: requestUri,  
        type: "POST",  
        contentType: "application/json;odata=verbose",  
        headers: requestHeaders,  
        data: requestBody,  
    });  
};
```

4. Return to **App.js** and locate the **onAddCustomer** function.
5. Implement **onAddCustomer** using the following code. Remember that you can also copy-and-paste the same code from the snippet file named **snippet05\_App.onAddCustomer.txt**.

```
function onAddCustomer() {  
    // get input data from add customer dialog  
    var LastName = $("#lastName").val();  
    var FirstName = $("#firstName").val();  
    var Company = $("#company").val();  
    var WorkPhone = $("#workPhone").val();  
    var HomePhone = $("#homePhone").val();  
    var Email = $("#email").val();
```

```
// add new customer
var promise =
    wingtip.Customers.DataAccess.addCustomer(FirstName, LastName, Company, WorkPhone, HomePhone, Email);

// refresh UI after adding new customer
promise.then(onSuccess, onError);
}
```

6. Note that when the SharePoint REST API call to add the customer succeeds, the code in the **onAddCustomer** function assigns the **onSuccess** function as the callback function. You do not need to modify the **onSuccess** function, but you should understand what it does. The **onSuccess** function calls the **getCustomers** function in **App.js** to send a new query to the SharePoint host and refresh the page with the latest set of items in the Customers list.

```
function onSuccess(data, request) {
    getCustomers();
}
```

7. Test your work by starting a new Visual Studio debugging session.
- Make sure you are in Visual Studio and that you have the **SharePointCRM** project open.
  - Press the **{F5}** key to begin a debugging session and wait for the start page to appear.
  - You should see the page update to display the table of customers.
  - Click the **Add New Customer** button to display the **Add Customer** dialog.

The screenshot shows the 'SharePoint REST API Lab' interface. At the top, there is a 'Back to Host Web' link. Below it is the 'SharePoint REST API Lab' title. A button labeled 'Add New Customer' is visible. Below the button is a table with columns: First Name, Last Name, Company, Work Phone, Home Phone, and Email. The table contains several rows of customer data. An 'Add Customer' dialog box is open, showing input fields for First Name, Last Name, Company, Work Phone, Home Phone, and Email. The dialog has 'Add' and 'Cancel' buttons at the bottom.

- e) Fill out the **Add Customer** dialog with sample customer data as shown in the following screenshot. If you use enter a **Last Name** that starts with "A", the customer will be sorted to the top of the table. Click **Add** to save the new customer.

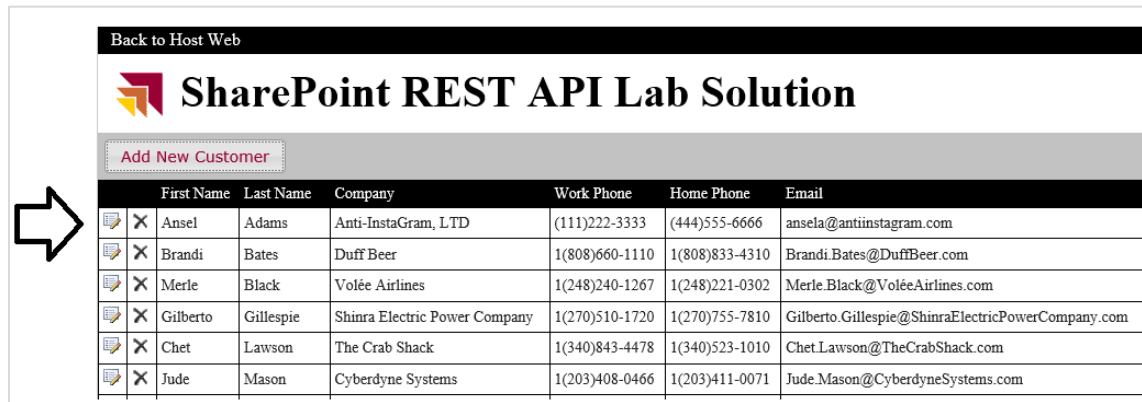
The screenshot shows the 'Add Customer' dialog box with the following data entered:

- First Name: Ansel
- Last Name: Adams
- Company: Anti-InstaGram, LTD
- Work Phone: (111)222-3333
- Home Phone: (444)555-6666
- Email: ansela@antiinstagram.com

The dialog has 'Add' and 'Cancel' buttons at the bottom.

- f) Once the start page has refreshed, you should see that new customer in the table on the start page.





	First Name	Last Name	Company	Work Phone	Home Phone	Email
	Ansel	Adams	Anti-InstaGram, LTD	(111)222-3333	(444)555-6666	ansela@antiinstagram.com
	Brandi	Bates	Duff Beer	1(808)660-1110	1(808)833-4310	Brandi.Bates@DuffBeer.com
	Merle	Black	Volée Airlines	1(248)240-1267	1(248)221-0302	Merle.Black@VoléeAirlines.com
	Gilberto	Gillespie	Shinra Electric Power Company	1(270)510-1720	1(270)755-7810	Gilberto.Gillespie@ShinraElectricPowerCompany.com
	Chet	Lawson	The Crab Shack	1(340)843-4478	1(340)523-1010	Chet.Lawson@TheCrabShack.com
	Jude	Mason	Cyberdyne Systems	1(203)408-0466	1(203)411-0071	Jude.Mason@CyberdyneSystems.com

g) Close the browser and stop the Visual Studio debugging session.

Next, you will add the code require to support DELETE behavior.


8. Return to Visual Studio.
9. Open the source file named **Wingtip.Customers.DataAccess.js** if it is not already open.
10. Look inside **Wingtip.Customers.DataAccess.js** and find the **deleteCustomer** function.
11. Implement **deleteCustomer** using the following code. Remember that you can also copy-and-paste the same code from the snippet file named **snippet06\_Wingtip.Customers.DataAccess.deleteCustomer.txt**.

```
var deleteCustomer = function (Id) {
    var requestUri = "../_api/web/lists/getByTitle('Customers')/items(" + Id + ")";
    var requestHeaders = {
        "accept": "application/json;odata=verbose",
        "X-RequestDigest": requestDigest,
        "If-Match": "*"
    }
    return $.ajax({
        url: requestUri,
        type: "DELETE",
        headers: requestHeaders,
    });
};
```

12. Move over to **App.js** and locate the **onDeleteCustomer** function.
13. Implement **onDeleteCustomer** using the following code. Remember that you can also copy-and-paste the same code from the snippet file named **snippet07\_App.onDeleteCustomer.txt**.

```
function onDeleteCustomer(customerId) {
    var promise = Wingtip.Customers.DataAccess.deleteCustomer(customerId);
    promise.then(onSuccess, onError);
}
```

14. Test your work by starting a new Visual Studio debugging session.
  - a) Make sure you are in Visual Studio and that you have the **SharePointCRM** project open.
  - b) Press the **{F5}** key to begin a debugging session and wait for the start page to appear.
  - c) You should see the start page load and display the table of customers.
  - d) Click the delete icon in the row of the customer named **Chet Lawson**.



		First Name	Last Name	Company
	X	Merle	Black	Volée Airlines
	X	Gilberto	Gillespie	Shinra Electric Power Company
	X	Chet	Lawson	The Crab Shack
	X	Jude	Mason	Cyberdyne Systems
	X	Ana	Mathews	WarioWare, Inc.

- e) Verify that the customer named **Chet Lawson** has been deleted from the table.

	X	Merle	Black	Volée Airlines
	X	Gilberto	Gillespie	Shinra Electric Power Company
	X	Jude	Mason	Cyberdyne Systems
	X	Ana	Mathews	WarioWare, Inc.
	X	Bridgette	Meadows	Brown Streak Railroad

- f) Close the browser window and stop the Visual Studio debugging session.

Finally, you are ready to begin the last exercise where you will add the extra code required to update existing customers.

## Exercise 4: Updating Existing Items with the SharePoint REST API

In this exercise you will implement UPDATE behavior so that the SharePointCRM app allows users to update existing customer items from the Customer list.

1. Return to Visual Studio.
2. Look inside **Wingtip.Customers.DataAccess.js** and locate the **getCustomer** function.
3. Implement **getCustomer** using the following code. Remember that you can also copy-and-paste the same code from the snippet file named **snippet08\_Wingtip.Customers.DataAccess.getCustomer.txt**.

```
var getCustomer = function (Id) {  
    // create the REST URI to target an item in the Customers list ID  
    var requestUri = "../_api/web/lists/getByTitle('Customers')/items(" + Id + ")";  
  
    return $.ajax({  
        url: requestUri,  
        contentType: "application/json;odata=verbose",  
        headers: { "accept": "application/json;odata=verbose" }  
    });  
}
```

4. Move over to **App.js** and locate the **onUpdateCustomerRequest** function.
5. Implement **onUpdateCustomerRequest** using the following code. Remember that you can also copy-and-paste the same code from the snippet file named **snippet09\_App.onUpdateCustomerRequest.txt**.

```
function onUpdateCustomerRequest(customerId) {  
    var promise = Wingtip.Customers.DataAccess.getCustomer(customerId);  
    promise.then(onUpdateCustomerDialog, onError);  
}
```

Note that the **onUpdateCustomerRequest** function use the **onUpdateCustomerDialog** function as the success callback function. There is no need for you to modify the **onUpdateCustomerDialog** function as it is already fully functional. However, you might want to look at the implementation to better understand how this function fits in to the flow of updating an existing customer.

6. Inspect the implementation of the **onUpdateCustomerDialog** function. As you can see, this function populates the **Edit Customer** dialog with the data returned for the current customer and then it displays the **Edit Customer** dialog to the user. It also provides the flow to call the **onUpdateCustomer** function in **App.js** to save the customer item.

```
function onUpdateCustomerDialog(data) {  
    // update customer dialog with current customer data  
    $("#firstName").val(data.d.FirstName);  
    $("#lastName").val(data.d.Title);  
    $("#company").val(data.d.Company);  
    $("#workPhone").val(data.d.WorkPhone);  
    $("#homePhone").val(data.d.HomePhone);  
    $("#email").val(data.d.Email);  
  
    // store item metadata values into hidden controls  
    $("#customer_id").val(data.d.ID);  
    $("#etag").val(data.d.__metadata.etag);  
  
    var customer_dialog = $("#customer_dialog");  
  
    customer_dialog.dialog({  
        autoOpen: true,  
        title: "Edit Customer",  
        width: 640,  
        buttons: {  
            "Update": function () {  
                onUpdateCustomer();  
                $(this).dialog("close");  
            },  
            "Cancel": function () {  
                $(this).dialog("close");  
            },  
        },  
    });  
}
```

7. Look inside **Wingtip.Customers.DataAccess.js** and locate the **updateCustomer** function.
8. Implement **updateCustomer** using the following code. Remember that you can also copy-and-paste the same code from the snippet file named **snippet10\_Wingtip.Customers.DataAccess.updateCustomer.txt**.

```
var updateCustomer = function (Id, FirstName, LastName, Company, WorkPhone, HomePhone, Email, ETag) {  
    var requestUri = "../_api/web/lists/getByTitle('Customers')/items(" + Id + ")";  
  
    var requestHeaders = {  
        "accept": "application/json;odata=verbose",  
        "X-HTTP-Method": "MERGE",  
        "X-RequestDigest": requestDigest,  
        "If-Match": ETag  
    }  
  
    var customerData = {  
        __metadata: { "type": "SP.Data.CustomersListItem" },  
        Title: LastName,  
        FirstName: FirstName,  
        Company: Company,  
        WorkPhone: WorkPhone,  
        HomePhone: HomePhone,  
        Email: Email  
    };  
  
    var requestBody = JSON.stringify(customerData);  
  
    return $.ajax({  
        url: requestUri,  
        type: "POST",  
        contentType: "application/json;odata=verbose",  
        headers: requestHeaders,  
    });  
}
```


```
data: requestBody,
});
};
```

9. Move back over to **App.js** and locate the **onUpdateCustomerRequest** function.
10. Implement the **onUpdateCustomer** function using the following code. Remember that you can also copy-and-paste the same code from the snippet file named **snippet11\_App.onUpdateCustomerRequest.txt**.

```
function onUpdateCustomer() {
    // scrape input values from dialog
    var Id = $("#customer_id").val();
    var FirstName = $("#firstName").val();
    var LastName = $("#lastName").val();
    var Company = $("#company").val();
    var WorkPhone = $("#workPhone").val();
    var HomePhone = $("#homePhone").val();
    var Email = $("#email").val();
    var ETag = $("#etag").val();

    // update customer
    var promise = Wingtip.Customers.DataAccess.updateCustomer(Id, FirstName, LastName,
                                                                Company, WorkPhone, HomePhone, Email, ETag);
    promise.then(onSuccess, onError);
}
```

11. Test your work by starting a new Visual Studio debugging session.
  - a) Make sure you are in Visual Studio and that you have the **SharePointCRM** project open.
  - b) Press the **{F5}** key to begin a debugging session and wait for the start page to appear.
  - c) You should see the start page load and display the table of customers.
  - d) Try and update a customer by clicking the update icon in the row of the customer named **Ana Mathews**.



	First Name	Last Name	Company	Work Phone
	Merle	Black	Volée Airlines	1(248)240-1267
	Gilberto	Gillespie	Shinra Electric Power Company	1(270)510-1720
	Jude	Mason	Cyberdyne Systems	1(203)408-0466
	Ana	Mathews	WarioWare, Inc.	1(844)663-5428
	Bridgette	Meadows	Brown Streak Railroad	1(250)468-4824
	Quincy	Nelson	Benthic Petroleum	1(340)608-7748
	Rose	Parsons	Hanso Foundation	1(802)357-5583






- e) The **Edit Customer** dialog should appear with the column values for Ana Mathews. Change Ana's Last Name from **Mathews** to **Conda** and click the **Update** button to save your changes.

	First Name	Last Name	Company	Work Phone	Home Phone	Email
	Merle	Black	Volée Airlines	1(248)240-1267	1(248)221-0302	Merle.Black@VoléeAirlines.com
	Gilberto	Gill	Shinra Electric Power Company			ShinraElectricPowerCompany.com
	Jude	Mason	Cyberdyne Systems			CyberdyneSystems.com
	Ana	Mat	WarioWare, Inc.			WarioWareInc.com
	Bridgette	Meadows	Brown Streak Railroad			BrownStreakRailroad.com
	Quincy	Nelson	Benthic Petroleum			BenthicPetroleum.com
	Rose	Parsons	Hanso Foundation			HansoFoundation.com
	Naomi	Schultz	WarioWare, Inc.			WarioWareInc.com
	Lynne	Stephens	WarioWare, Inc.			WarioWareInc.com
	Sid	Stout	WarioWare, Inc.			WarioWareInc.com
	Diane	Strickland	WarioWare, Inc.			WarioWareInc.com
	Luther	Sullivan	WarioWare, Inc.			WarioWareInc.com

**Edit Customer**

First Name:   
Last Name:   
Company:   
Work Phone:   
Home Phone:   
Email:

- f) Verify that Ana's last name has been changed from **Mathews** to **Conda**.

		First Name	Last Name	Company	Work Phone	Home Phone
	X	Merle	Black	Volar Airlines	1(248)240-1267	1(248)221-0302
	X	Ana	Conda	Volvo Inc.	1(844)663-5428	1(844)782-2117
	X	Gilberto	Gillespie	Shinn Electric Power Company	1(270)510-1720	1(270)755-7810
	X	Jude	Mason	Cyberdyne Systems	1(203)408-0466	1(203)411-0071
	X	Bridgette	Meadows	Brown Streak Railroad	1(250)468-4824	1(250)403-3653

g) Close the browser window and stop the Visual Studio debugging session.

You have now completed all the exercises in this lab.