## Developing a SharePoint App using Knockout.js

**Lab Time**: 60 minutes

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

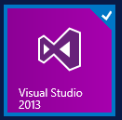
**Lab Overview**: In this lab you will create a complete app that performs CRUD operations on a Contacts list. This app will utilize the Model-View-ViewModel pattern to properly separate concerns between JavaScript libraries. You will also utilize the module pattern within JavaScript libraries to properly encapsulate code.

**Lab Setup**: Ensure you have a development site available for this lab

### Exercise 1: Reviewing the Starter Solution

In this exercise you will open the solution for the lab and examine the contents.

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



1. Open the starter solution in Visual Studio 2013:
   1. In Visual Studio select **File 🡪 Open 🡪 Project**.
   2. In the **Open Project** dialog:
      1. Browse to the **MVVM** folder and locate the file **MVVM.sln**.
      2. **Open** the solution.
   3. Examine the artifacts in the solution using the **Solution Explorer**.
      1. Expand the **Contacts** list node and open the **Elements.xml** file. Note the pre-populated contacts data.
      2. Expand the **Content** node and open the **App.css** file. Note the style definitions.
      3. Expand the **Images** node and note the available image files.
      4. Expand the **Pages** node and note the pre-defined pages for the solution. You will add code to these pages later.
      5. Expand the **Scripts** node and note the available libraries. You will add code to these libraries later.
2. Update the Site URL
   1. In the **Solution Explorer**, click the MVVM project node.
   2. Select **View 🡪 Properties Window** from the Main Menu.
   3. In the Properties Window, locate the **Site URL** property, and update it to refer to the site where you will deploy the completed app. (Note: if you created the <http://dev.wingtip.com> site earlier you may use this one or any other one of your choice).

### Exercise 2: Coding the Data Library

In this exercise, you will finish coding the wingtip.contacts.data.js library. This library contains all of the functions necessary to perform CRUD operations on the Contacts list. In the MVVM pattern, the data access functions are separated from the functions that display data in the web page. This “separation of concerns” allows for a greater reuse of the libraries.

1. Open the **wingtip.contacts.data.js** library for editing:
   1. In the Solution Explorer:
      1. Expand the **Scripts** node.
      2. Double-click the **wingtip.contacts.data.js** file.
   2. Examine the library and note its structure. The library declares a namespace **Wingtip.Contacts** and utilizes the JavaScript module pattern to encapsulate the CRUD operations.
2. Code the **create** function
   1. Locate the comment **//CODE GOES HERE** within the create function.
   2. Update the **create** function so that it looks like the code below:

var create = function (lname, fname, wphone) {

var deferred = $.ajax({

url: "../\_api/web/lists/getByTitle('Contacts')/items",

type: "POST",

contentType: "application/json;odata=verbose",

data: JSON.stringify(

{

'\_\_metadata': {

'type': 'SP.Data.ContactsListItem'

},

'Title': lname,

'FirstName': fname,

'WorkPhone': wphone

}),

headers: {

"accept": "application/json;odata=verbose",

"X-RequestDigest": $("#\_\_REQUESTDIGEST").val()

}

});

return deferred.promise();

},

Note in the above code how the jQuery ajax function returns a “deferred” object. The create function then returns the “promise” object of the deferred. The idea behind this pattern is that the asynchronous “success” and “failure” functions do not have to be coded directly in the data layer. Instead, the promise object allows the consuming code to manage success and failure separately. The use of promises aids significantly in enabling the MVVM pattern through separation of concerns. Later in the lab, you’ll see how the success and failure code is implemented by the ViewModel library.

1. Code the **readList** function
   1. Locate the comment **//CODE GOES HERE** within the readList function.
   2. Update the **readList** function so that it looks like the code below**:**

readList = function () {

var deferred = $.ajax(

{

url: "../\_api/web/lists/getByTitle('Contacts')/items/" +

"?$select=Id,FirstName,Title,WorkPhone" +

"&$orderby=Title,FirstName",

type: "GET",

headers: {

"accept": "application/json;odata=verbose",

}

}

);

return deferred.promise();

},

1. Code the **readItem** function
   1. Locate the comment **//CODE GOES HERE** within the readItem function.
   2. Update the **readItem** function so that it looks like the code below**:**

readItem = function (id) {

var deferred = $.ajax(

{

url: "../\_api/web/lists/getByTitle('Contacts')/getItemByStringId('" +

id + "')/?$select=Id,FirstName,Title,WorkPhone",

type: "GET",

headers: {

"accept": "application/json;odata=verbose",

}

}

);

return deferred.promise();

},

1. Code the **update** function
   1. Locate the comment **//CODE GOES HERE** within the update function.
   2. Update the **update** function so that it looks like the code below**:**

update = function (id, lname, fname, wphone) {

var deferred = $.ajax(

{

url: "../\_api/web/lists/getByTitle('Contacts')/getItemByStringId('" + id + "')",

type: "POST",

contentType: "application/json;odata=verbose",

data: JSON.stringify(

{

'\_\_metadata': {

'type': 'SP.Data.ContactsListItem'

},

'Title': lname,

'FirstName': fname,

'WorkPhone': wphone

}),

headers: {

"accept": "application/json;odata=verbose",

"X-RequestDigest": $("#\_\_REQUESTDIGEST").val(),

"IF-MATCH": "\*",

"X-Http-Method": "PATCH"

}

}

);

return deferred.promise();

},

Note in the above code how the “IF-MATCH” header contains an asterisk. This means that the update will succeed regardless of any changes that were made to the item since it was last retrieved. This is the simplest form of concurrency management in SharePoint. For more sophisticated management, the library could be modified to retrieve the item’s “e-Tag”, which could be compared during the update to ensure the target item has not been changed by some other process.

1. Code the **remove** function
   1. Locate the comment **//CODE GOES HERE** within the remove function.
   2. Update the **remove** function so that it looks like the code below**:**

remove = function (id) {

var deferred = $.ajax(

{

url: "../\_api/web/lists/getByTitle('Contacts')/getItemByStringId('" + id + "')",

type: "DELETE",

headers: {

"accept": "application/json;odata=verbose",

"X-RequestDigest": $("#\_\_REQUESTDIGEST").val(),

"IF-MATCH": "\*"

}

}

);

return deferred.promise();

};

1. Save and close the **wingtip.contacts.data.js** library.

### Exercise 3: Coding the Model Library

In this exercise, you will finish coding the wingtip.contacts.model.js library. This library provides a way to keep the data for a single list item. In the app, this library will be used to populate a collection of items based on the data in the Contacts list.

1. Open the **wingtip.contacts.model.js** library for editing:
   1. In the Solution Explorer:
      1. Expand the **Scripts** node.
      2. Double-click the **wingtip.contacts.model.js** file.
   2. Examine the library and note its structure. The library creates a simple object to maintain data from the Contacts list.
2. Code the library
   1. Locate the comment **//CODE GOES HERE** within the remove function.
   2. Update the **Wingtip.Contacts.Model** function so that it looks like the code below**:**

Wingtip.Contacts.Model = function (identifier, lastName, firstName, phoneNumber) {

//private members

var id = identifier,

lname = lastName,

fname = firstName,

phone = phoneNumber,

set\_id = function (v) { id = v; },

get\_id = function () { return id; },

set\_lname = function (v) { lname = v; },

get\_lname = function () { return lname; },

set\_fname = function (v) { fname = v; },

get\_fname = function () { return fname; },

set\_phone = function (v) { phone = v; },

get\_phone = function () { return phone; };

//public interface

return {

set\_id: set\_id,

get\_id: get\_id,

set\_lname: set\_lname,

get\_lname: get\_lname,

set\_fname: set\_fname,

get\_fname: get\_fname,

set\_phone: set\_phone,

get\_phone: get\_phone

};

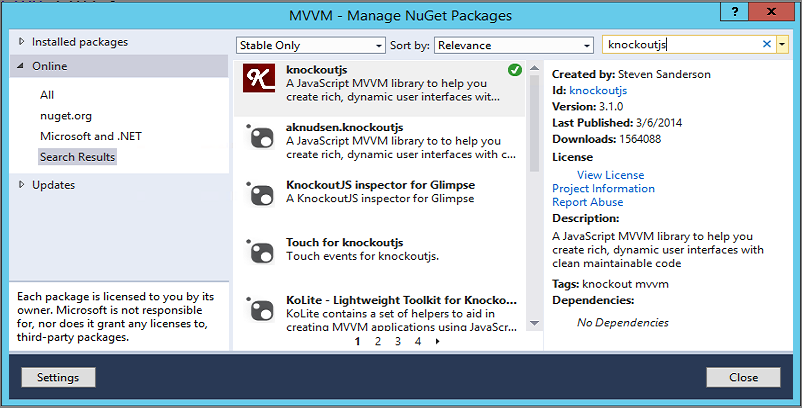
}

1. Save and close the **wingtip.contacts.model.js** library.

### Exercise 4: Coding the View Model

In this exercise, you will finish coding the wingtip.contacts.viewmodel.js library. The View Model is responsible for retrieving the data from the Contacts list, populating a collection, and binding it to the web page. This makes the View Model the heart of the functionality in the MVVM pattern. In order to support binding the data to the web pages, the app utilizes a popular third-party library called **Knockout**. Knockout supports data binding as well as automatic page updates when the bound data changes. You can get more information about knockout at http://knockoutjs.com/.

1. Add the NuGet package for knockoutjs:
   1. In the **Solution Explorer**:
      1. Right click on the MVVM project node and select **Manage NuGet Packages**.
      2. In the Manage NuGet Packages dialog, type **knockoutjs** in the **Search Online** box and press **Enter**
      3. Click the **Install** button associated with **knockoutjs**.
      4. Click the **Close** button.



1. Open the **wingtip.contacts.viewmodel.js** library for editing:
   1. In the Solution Explorer:
      1. Expand the **Scripts** node.
      2. Double-click the **wingtip.contacts.viewmodel.js** file.
   2. Examine the library and note its structure.
2. Code the **loadItem** function
   1. Locate the comment **//CODE GOES HERE** within the loadItem function.
   2. Update the **loadItem** function so that it looks like the code below:

loadItem = function (id) {

Wingtip.Contacts.Data.readItem(id).then(

//success

function (data) {

contacts.removeAll();

contacts.push(

new Wingtip.Contacts.Model(

data.d.Id,

data.d.Title,

data.d.FirstName,

data.d.WorkPhone));

//bind to web page with knockout

ko.applyBindings(Wingtip.Contacts.ViewModel, document.getElementById("formDiv"));

},

//failure

function (err) {

alert(JSON.stringify(err));

}

);

},

The above code has a lot going on. First, note the use of the “then” method following the call to the readItem function. Remember that the readItem function returns a promise object. The “then” method is a method of the promise object that takes a success and failure handler as arguments. This allows the consuming code to handle success and failure instead of the data layer. Second, note that the success function is filling an array of contacts. This special array is known as an “observable array”, which knockout can bind to the web pages of the app. The actual binding occurs when the applyBinding method of the ko object is called. The ko object is the core knockout object defined by the library you added earlier.

1. Code the **loadItems** function
   1. Locate the comment **//CODE GOES HERE** within the loadItems function.
   2. Update the **loadItems** function so that it looks like the code below**:**

loadItems = function () {

Wingtip.Contacts.Data.readList().then(

function (data) {

//When successful, fill observable array

var results = data.d.results;

contacts.removeAll();

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

contacts.push(

new Wingtip.Contacts.Model(

results[i].Id,

results[i].Title,

results[i].FirstName,

results[i].WorkPhone));

}

},

function (err) {

alert(JSON.stringify(err));

}

);

};

The above code is similar to the code found in the loadItem function, however, the above code is used to show views of the list containing many items. By contrast the loadItem method is used to display the details of a single item.

1. Save and close the **wingtip.contacts.viewmodel.js** library.

### Exercise 5: Coding the Default.aspx Page

In this exercise, you will finish coding the Default.aspx page. This page is used to display all of the Contacts in a table.

1. Open the **default.aspx** page for editing:
   1. In the Solution Explorer:
      1. Expand the **Pages** node.
      2. Double-click the **default.aspx** file.
   2. Examine the page and note its structure.
2. Validate/Update the script reference to knockout (as the current version is likely much greater than the version this course was created using) in the <asp:Content ContentPlaceHolderID=”PlaceHolderAdditionalPageHead”…> tag as shown below (i.e. update the version number to match the version of the knockoutjs script library you downloaded earlier (you can find the version number in the Solution Explorer, **MVVM** project, **Scripts** folder)):

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

<link rel="Stylesheet" type="text/css" href="../Content/App.css" />

<script type="text/javascript" src="../Scripts/jquery-1.8.2.min.js"></script>

<script type="text/javascript" src="../Scripts/knockout-2.3.0.js"></script>

<script type="text/javascript" src="/\_layouts/15/sp.runtime.js"></script>

<script type="text/javascript" src="/\_layouts/15/sp.js"></script>

<script type="text/javascript" src="/\_layouts/15/sp.ui.dialog.js"></script>

<script type="text/javascript" src="../Scripts/wingtip.contacts.data.js"></script>

<script type="text/javascript" src="../Scripts/wingtip.contacts.model.js"></script>

<script type="text/javascript" src="../Scripts/wingtip.contacts.viewmodel.js"></script>

<script type="text/javascript" src="../Scripts/DefaultPageEvents.js"></script>

</asp:Content>

1. Code the display table
   1. Locate the comment <!--Knockout binding code goes here -->.
   2. Update the contents of the **<div id=”resultsDiv”…>** tag with the following HTML and binding code to complete the **Default.aspx** page

<div id="resultsDiv" style="overflow: auto">

<table>

<caption>My Contacts</caption>

<thead>

<tr>

<th><span style="margin-right: 15px;"></span></th>

<th><span style="margin-right: 15px;"></span></th>

<th><span style="margin-right: 15px;">Last Name</span></th>

<th><span style="margin-right: 15px;">First Name</span></th>

<th><span style="margin-right: 15px;">Phone</span></th>

</tr>

</thead>

<tbody id="resultsTable" data-bind="foreach: get\_contacts()">

<tr>

<td nowrap="nowrap">

<img data-bind="attr: { id: 'edit' + get\_id() }" class="editButton" src="../images/edit.gif" />

</td>

<td nowrap="nowrap">

<img data-bind="attr: { id: 'delete' + get\_id() }" class="deleteButton" src="../images/delete.gif" />

</td>

<td nowrap="nowrap" data-bind="text: get\_lname()"></td>

<td nowrap="nowrap" data-bind="text: get\_fname()"></td>

<td nowrap="nowrap" data-bind="text: get\_phone()"></td>

</tr>

</tbody>

</table>

</div>

Note in the above code how the data-bind attributes are used to call methods on the View Model for data binding. This is the essence of what the knockout library does. It binds the page using a template so that you can use the same View Model to bind to different interfaces. To change the data from a table to a list, for example, all you would have to do is rewrite the HTML page template above to reflect a list instead of a table.

1. Save and close the **Default.aspx** file.

### Exercise 6: Coding the AddContact.aspx Page

In this exercise, you will finish coding the AddContact.aspx page. This page is used to display a modal dialog for adding a new contact.

1. Open the **addcontact.aspx** page for editing:
   1. In the Solution Explorer:
      1. Expand the **Pages** node.
      2. Double-click the **addcontact.aspx** file.
   2. Examine the page and note its structure.
2. Validate/Update the script reference to knockout (as the current version is likely much greater than the version this course was created using) in the <asp:Content ContentPlaceHolderID=”PlaceHolderAdditionalPageHead”…> tag as shown below (i.e. update the version number to match the version of the knockoutjs script library you downloaded earlier (you can find the version number in the Solution Explorer, **MVVM** project, **Scripts** folder)):

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

<link rel="Stylesheet" type="text/css" href="../Content/App.css" />

<script type="text/javascript" src="../Scripts/jquery-1.8.2.min.js"></script>

<script type="text/javascript" src="../Scripts/knockout-2.3.0.js"></script>

<script type="text/javascript" src="/\_layouts/15/sp.runtime.js"></script>

<script type="text/javascript" src="/\_layouts/15/sp.js"></script>

<script type="text/javascript" src="/\_layouts/15/sp.ui.dialog.js"></script>

<script type="text/javascript" src="../Scripts/wingtip.contacts.data.js"></script>

<script type="text/javascript" src="../Scripts/wingtip.contacts.model.js"></script>

<script type="text/javascript" src="../Scripts/wingtip.contacts.viewmodel.js"></script>

<script type="text/javascript" src="../Scripts/DefaultPageEvents.js"></script>

</asp:Content>

1. Code the display table
   1. Locate the comment <!—Form HTML goes here -->.
   2. Update the contents of the **<div id="formDiv"…>** tag with the following HTML and binding code to complete the **AddContact.aspx** page

<div id="formDiv" style="overflow: auto">

<form>

<table>

<tr>

<td>First Name</td>

<td>

<input id="firstName" type="text" autofocus required placeholder="John" />

<div class="error">First Name is required</div>

</td>

</tr>

<tr>

<td>Last Name</td>

<td>

<input id="lastName" type="text" required placeholder="Doe" />

<div class="error">Last Name is required</div>

</td>

</tr>

<tr>

<td>Phone Number</td>

<td>

<input id="workPhone" type="tel" required placeholder="555-555-5555" />

<div class="error">Please enter a valid phone number</div>

</td>

</tr>

<tr>

<td>

<div style="margin-top:5px;padding-top:5px;">

<input id="resetButton" type="reset" class="app-button" />

</div>

</td>

<td>

<div style="margin-top:5px;padding-top:5px;">

<button value="Submit" id="addButton" >Submit</button>

</div>

</td>

</tr>

</table>

</form>

</div>

1. Save and close the **AddContact.aspx** file.

### Exercise 7: Coding the EditContact.aspx Page

In this exercise, you will finish coding the EditContact.aspx page. This page is used to edit an existing contact.

1. Open the **editcontact.aspx** page for editing:
   1. In the Solution Explorer:
      1. Expand the **Pages** node.
      2. Double-click the **editcontact.aspx** file.
   2. Examine the page and note its structure.
2. Validate/Update the script reference to knockout (as the current version is likely much greater than the version this course was created using) in the <asp:Content ContentPlaceHolderID=”PlaceHolderAdditionalPageHead”…> tag as shown below (i.e. update the version number to match the version of the knockoutjs script library you downloaded earlier (you can find the version number in the Solution Explorer, **MVVM** project, **Scripts** folder)):

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

<link rel="Stylesheet" type="text/css" href="../Content/App.css" />

<script type="text/javascript" src="../Scripts/jquery-1.8.2.min.js"></script>

<script type="text/javascript" src="../Scripts/knockout-2.3.0.js"></script>

<script type="text/javascript" src="/\_layouts/15/sp.runtime.js"></script>

<script type="text/javascript" src="/\_layouts/15/sp.js"></script>

<script type="text/javascript" src="/\_layouts/15/sp.ui.dialog.js"></script>

<script type="text/javascript" src="../Scripts/wingtip.contacts.data.js"></script>

<script type="text/javascript" src="../Scripts/wingtip.contacts.model.js"></script>

<script type="text/javascript" src="../Scripts/wingtip.contacts.viewmodel.js"></script>

<script type="text/javascript" src="../Scripts/DefaultPageEvents.js"></script>

</asp:Content>

1. Code the display table
   1. Locate the comment <!—Form HTML goes here -->.
   2. Update the contents of the **<div id="formDiv"…>** tag with the following HTML and binding code to complete the **EditContact.aspx** page

<div id="formDiv" style="overflow: auto">

<form>

<input id="identifier" data-bind="attr: { value: get\_contact(0).get\_id() }" type="hidden" />

<table>

<tr>

<td>First Name</td>

<td>

<input id="firstName" data-bind="attr: { value: get\_contact(0).get\_fname() }" type="text"

autofocus required placeholder="John" />

<div class="error">First Name is required</div>

</td>

</tr>

<tr>

<td>Last Name</td>

<td>

<input id="lastName" data-bind="attr: { value: get\_contact(0).get\_lname() }" type="text"

required placeholder="Doe" />

<div class="error">Last Name is required</div>

</td>

</tr>

<tr>

<td>Phone Number</td>

<td>

<input id="workPhone" data-bind="attr: { value: get\_contact(0).get\_phone() }" type="tel"

required placeholder="555-555-5555" />

<div class="error">Please enter a valid phone number</div>

</td>

</tr>

<tr>

<td>

<div style="margin-top:5px;padding-top:5px;">

<button value="Submit" id="updateButton" >Submit</button>

</div>

</td>

</tr>

</table>

</form>

</div>

1. Save and close the **AddContact.aspx** file.

### Exercise 8: Testing the App

In this exercise, you will test the completed app.

1. Run the app:
   1. Press the **F5** key.
   2. When the app starts, verify you see a list of contacts.
2. Add a contact:
   1. Push the **New** button.
   2. Fill in the form with new data.
   3. Press the **Submit** button.
   4. Verify the new contact appears in the list.
3. Edit the contact
   1. Click the **edit icon** next to your new contact.
   2. Edit the contact information.
   3. Press the **Submit** button.
   4. Verify the modified contact information appears.
4. Delete the contact
   1. Click the **delete icon** next to your new contact.
   2. Verify the contact is deleted.

You have successfully created and tested your MVVM KnockoutJS Application!