## SharePoint Development Practices & Techniques

**Lab Time**: 45 minutes

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

**Lab Overview**: SharePoint 2013 has a strong emphasis on adding client-side development to the SharePoint developer’s toolbox. JavaScript development is very different from managed code development and not all SharePoint developers are as strong with JavaScript coding. In this lab you will use some of the popular JavaScript libraries that developers use to make their lives easier: jQuery, DataJS and KnockoutJS.

### 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://devpractices.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:

[..]\DevPracticesTechniques

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

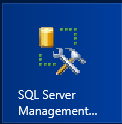
#### Check to see if you need to setup the Sample Resource Web Services

1. Ensure you are logged into the **WingtipServer** server as **WINGTIP\Administrator**.
2. Check to see if the sample services have already been deployed:
   1. Open Internet Explorer and try to navigate to the following OData services:
      1. <http://cptresources.wingtip.com:81/services/AdventureWorks2012Person.svc>
      2. <http://cptresources.wingtip.com:81/services/AdventureWorks2012Product.svc>
      3. <http://cptresources.wingtip.com:81/services/Calculator.svc>
   2. If the links above work and do not return 404 errors, you can skip to the next exercise **(Exercise 2)** because the sample site & services have already been deployed.
3. If the **Calculator.svc** service works but the two **AdventureWorks2012\*.svc** services **do not work**, this indicates the services were deployed, but two of the services are not able to connect to the local SQL Server sample **AdventureWorks2012** database. The possible issues include:
   1. A connection problem with the IIS website.
   2. The connection strings in the **cptservices** application’s **web.config**.
   3. The sample **AdventureWorks2012** database is not installed in the local SQL Server.
   4. The identity of the hosting app pool does not have rights to the **AdventureWorks 2012** database.
4. If all three of the web services return 404 errors, that means you need to setup these web services and that you should continue on to step 6 to work through the steps to properly configure these web services.

You must resolve any issues and ensure these web services are working before proceeding to Exercise 2.

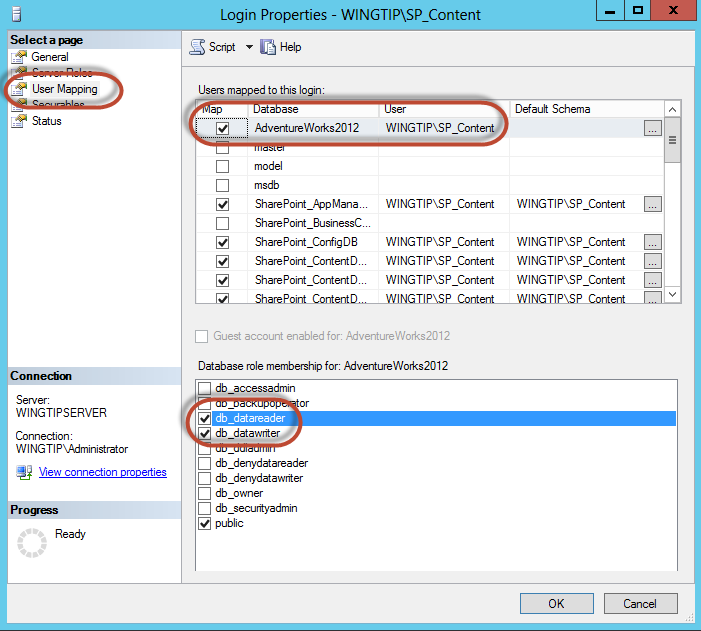
#### Update the App Pool Identity Rights in the SQL Server Adventure Works Database

1. Grant the identity of the hosting web site’s application pool access to the AdventureWorks2012 sample database:
   1. Open **SQL Server Management Studio**:
      1. **Windows Keyboard Key 🡪 SQL Server Management Studio**



**(Note:** Alternatively after pressing the **Windows Keyboard Key** you can simply start typing the name of the program you are looking for (e.g. SQL); this will filter the results to those that match the letters typed on the keyboard)

* 1. In the **Connect to Server** dialog, enter the following and click **Connect**:
     1. **Server Type:** Database Engine
     2. **Server name:** WINGTIPSERVER
     3. **Authentication:** Windows Authentication
  2. In the **Object Explorer** tool window, expand the tree to show the contents of **WINGTIPSERVER 🡪 Security 🡪 Logins**.
  3. Right-click **WINGTIP\SP\_Content** and select **Properties**.
  4. In the **Select a page** pane on the left-hand side of the **Login Properties – WINGTIP\SP\_Content** dialog, select **User Mapping**.
  5. In the top portion of the dialog, in the **Users mapped to this login**, check the box next to **AdventureWorks2012**.
  6. In the bottom portion of this dialog, in the **Database role membership for AdventureWorks2012**, check the boxes next to:
     1. **db\_datareader**
     2. **db\_datawriter**



1. Next we need to add the **NT AUTHORITY\IUSR** account the SQL Database to account for an issue with the Single Server image environment. (*Note: you would not do this on a production multi-server farm*.)
   1. In the SQL Server Management Studio Object Explorer tool window, expand the tree to show the contents of WINGTIPSERVER 🡪 Security 🡪 Logins.
   2. Right-click **Logins** and select **New Logins**.
   3. In the **Login - New** dialog box:
      1. **Login name:** NT AUTHORITY\IUSR
      2. On the left-hand side of the Login– New dialog, select **User Mapping**.
   4. In the top portion of the dialog, in the **Users mapped to this login**, check the box next to **AdventureWorks2012**.
   5. In the bottom portion of this dialog, in the **Database role membership for AdventureWorks2012**, check the boxes next to:
      1. db\_datareader
      2. db\_datawriter
   6. Click **OK** and **close** SQL Server Management Studio.

#### Create Sample Resource Website & Deploy Sample Services Project

1. Create a new IIS web site to host the sample CPT OData web services:
   1. Run a PowerShell script, found in the **ExtraStudentFiles** folder in the student folder:
      1. Right-click **CreateCptResourcesWebSite.ps1** and select **Run with PowerShell**. This file can be found in the files associated with this course:

[..]\ExtraStudentFiles\Scripts

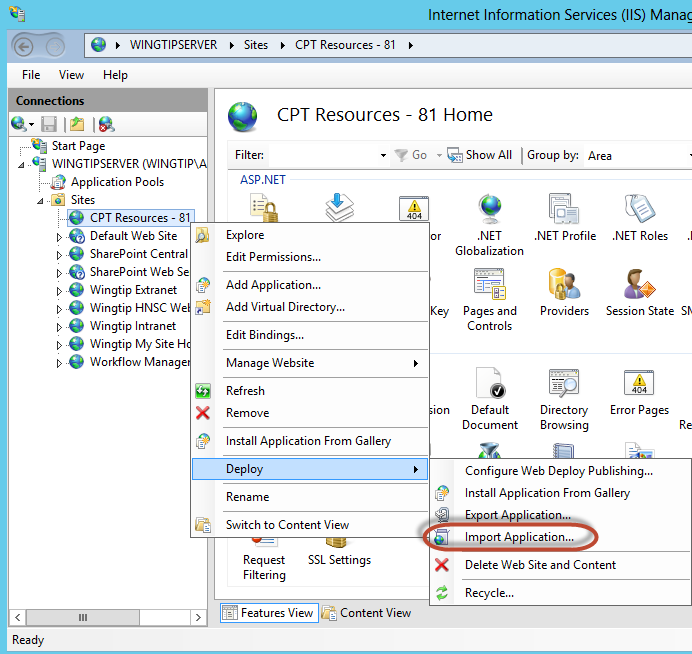
This script will create a new website in IIS named **CPT Resources – 81** with a binding of **http://cptresources.wingtip.com:81**.

* 1. Close the PowerShell console window.

1. Install the CPT Services sample OData Web services project:
   1. Open **Internet Information Services (IIS) Manager**:
      1. **Windows Keyboard Key 🡪 Internet Information Services (IIS) Manager**



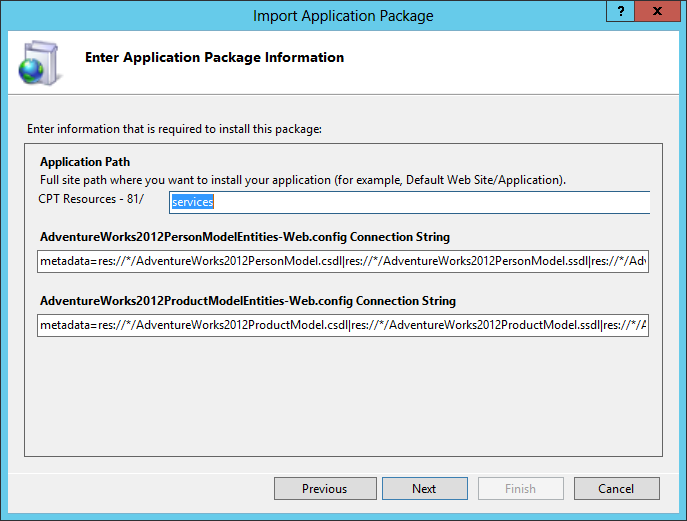
* 1. In the left-hand **Connections** pane, expand the tree to the following location:
     1. **WINGTIPSERVER 🡪 Sites 🡪 CPT Resources – 81**
  2. Right-click **CPT Resources – 81** and select **Deploy 🡪 Import Application**.



* 1. In the **Import Application Package** dialog’s **Select the Package** page, browse to the file **CptServices\_WebDeployPackage.zip** found in the files associated with this course:

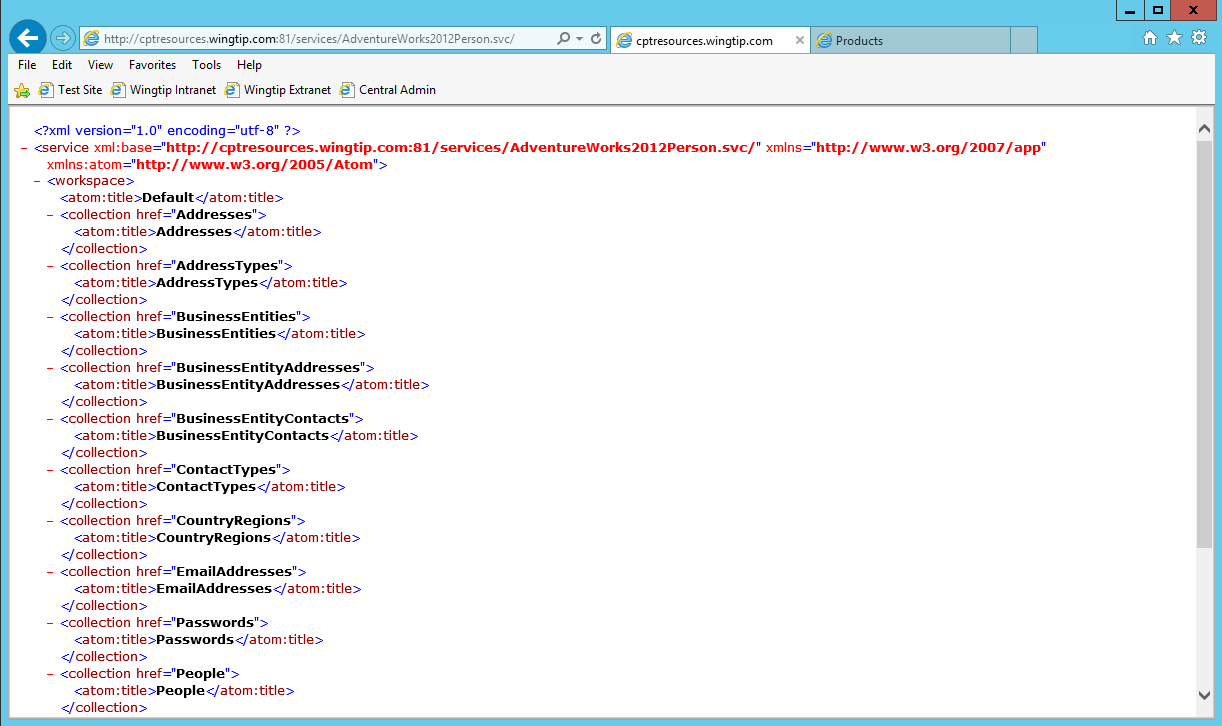
[..]\ExtraStudentFiles\Resources

* 1. Click **Open** and then click **Next**.
  2. On the **Select the Contents of the Package** page, click **Next**.
  3. On the **Enter Application Package Information** page, make sure the **Application Path** is set to **services** and click **Next.**



* 1. After the package is deployed, click **Finish** on the **Installation Progress and Summary** page to close the dialog.

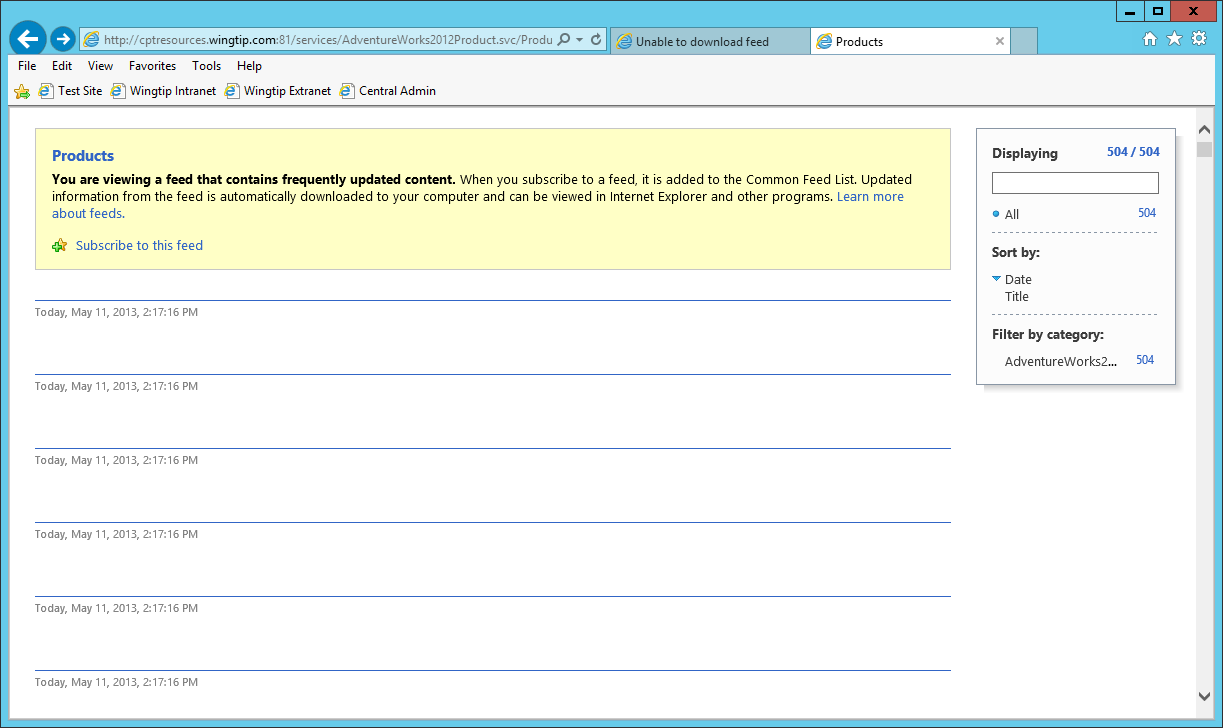
1. Verify the sample OData services are working:
   1. Open Internet Explorer and try to navigate to the following OData services:
      1. <http://cptresources.wingtip.com:81/services/AdventureWorks2012Person.svc>
      2. <http://cptresources.wingtip.com:81/services/AdventureWorks2012Product.svc>
      3. <http://cptresources.wingtip.com:81/services/Calculator.svc>
   2. You should see XML returned by all services as the following figure shows:



* 1. Open Internet Explorer and try to navigate to the following OData services:
     1. <http://cptresources.wingtip.com:81/services/AdventureWorks2012Person.svc/People> (see note below)
     2. <http://cptresources.wingtip.com:81/services/AdventureWorks2012Product.svc/Products>

With the Person.svc/People you should see a message that Internet Explorer cannot display this feed (The size of this feed exceeds the download limit)

* 1. With the **Products.svc/Products** you should see feed information on Products similar to that shown in the image below:

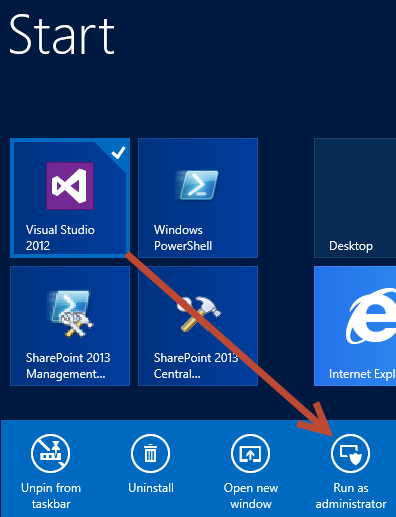


You are now ready to perform the following lab exercises.

### Exercise 2: Working with the jQuery and datajs JavaScript Libraries

In this exercise you will create a simple example that uses the datajs library to query data from the OData services you created in the previous exercise. You will also use jQuery to access HTML objects on the page.

1. Create a new project in Visual Studio 2012:
   1. Launch **Visual Studio 2012** as administrator:
      1. **Windows Keyboard Key 🡪 Right click** on the **Visual Studio 2012** tile and select **Run as administrator**



* 1. In Visual Studio select **File 🡪 New 🡪 Project**.
  2. In the **New Project** dialog:
     1. Find the **ASP.NET Empty Web Application** template under the **Templates \ Visual C# \ Web** section.
     2. **Name**: DataJsExample
     3. **Location:** [..]\DevPracticesTechniques\Exercises\Ex2  
        (Where [..] represents the location of the student files (e.g. c:\student\..)
     4. **Uncheck** the **Create directory for solution** checkbox
     5. Click **OK**

1. When the project loads, add the required JavaScript libraries to it:
   1. Right-click the project **DataJsExample** in the **Solution Explorer** tool window and select **Add 🡪 Existing Item**.
   2. In the **Add Existing Item** dialog add the jQuery library from the files provided for this course:

[..]\ExtraStudentFiles\Resources\Libraries\jquery\jquery-1.8.1.min.js

* 1. Repeat the above steps to add the datajs library to the project:

[..]\ExtraStudentFiles\Resources\Libraries\datajs\datajs-1.0.3.min.js

1. Add a new web page to the project:
   1. Right-click the project **DataJsExample** in the **Solution Explorer** tool window and select **Add 🡪 New Item:**
      1. In the **Add New Item** dialog, select the **HTML Page** template under the **Visual C# \ Web** section.
      2. **Name:** default.html
      3. Click the **Add** button
2. Add JavaScript library references to the page:
   1. Add the two lines to the page to the **<head>** portion of the **default.html** page:

<script src="jquery-1.8.1.min.js"></script>

<script src="datajs-1.0.3.min.js"></script>

1. Add visual components to the page. Within the **<body>** of the page, add the following HTML markup to design the page:

<h1>Adventure Works Product Catalog Search Tool</h1>

<p>Enter words to search for in the catalog, separated by spaces.</p>

<input id="searchBox" type="text" />

<button id="searchButton">Search</button>

<div>

<div style="float: left; width: 200px; padding: 10px;">

<h2>Query Results</h2>

<ul id="queryResultsList">

<li>item</li>

<li>item</li>

<li>item</li>

</ul>

</div>

</div>

1. Now, using jQuery, attach a new event to the search button once the page has loaded. Add the following code immediately after the code you just added: (i.e. after the closing </div> tag)

<script type="text/javascript">

var serviceUri = "http://cptresources.wingtip.com:81/services/AdventureWorks2012Product.svc/";

// using jQuery, once the jQuery library has loaded & the page is ready...

$(document).ready(function () {

// attach event to search button

$("#searchButton").click(function () {

//NEXT-STEP

});

});

</script>

1. The first step is to take the search phrase that was entered and split it into multiple words. Replace the **//NEXT-STEP** JavaScript comment with the following code:

// break search into array of words

var searchQuery = $("#searchBox").val();

var regEx = /\b(\w+)\b/g;

var searchWords = searchQuery.match(regEx);

// if something was entered, execute query

if (searchWords) {

//NEXT-STEP

}

1. Next create the query you will issue to the OData service. This involves taking the words entered in the search box and appending them to the URI that will be sent to the service as well as specifying the order the data should be returned and what fields to pull back. Replace the **//NEXT-STEP** JavaScript comment with the following code:

// get words to add to query

var filterWords = "";

$.each(searchWords, function (key, word) {

if (key) filterWords += "%20and%20";

// double up any single quotes

word = word.replace(/'/g, "''");

// url escape word

word = encodeURIComponent(word);

filterWords += "substringof('" + word + "',Name)";

});

// create base portion of the URI

var queryUri = serviceUri + "Products?" +

"$select=ProductID,ProductNumber,Name,ListPrice&" +

"$top=10&" +

"$orderby=Name%20asc&" +

"$filter=" + filterWords;

//NEXT-STEP

1. With the query built, now issue the query to the service to get the response. Replace the **//NEXT-STEP** JavaScript comment with the following code:

// configure datajs to allow cross domain callbacks using JSONP

OData.defaultHttpClient.enableJsonpCallback = true;

// run query

OData.read(queryUri, function (queryResponse) {

// clear out returned items

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

// for each result, add to list

$.each(queryResponse.results, function (o) {

//NEXT-STEP

});

}, function (error) { alert("error occured:" + JSON.stringify(error)); });

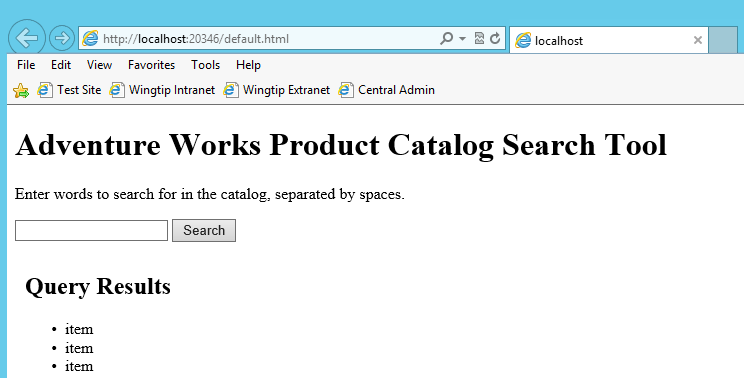
1. With the response in hand, the next step is to update the user interface with the items returned. Replace the **//NEXT-STEP** JavaScript comment with the following code:

$("#queryResultsList").append("<li>" + this.Name + "</li>");

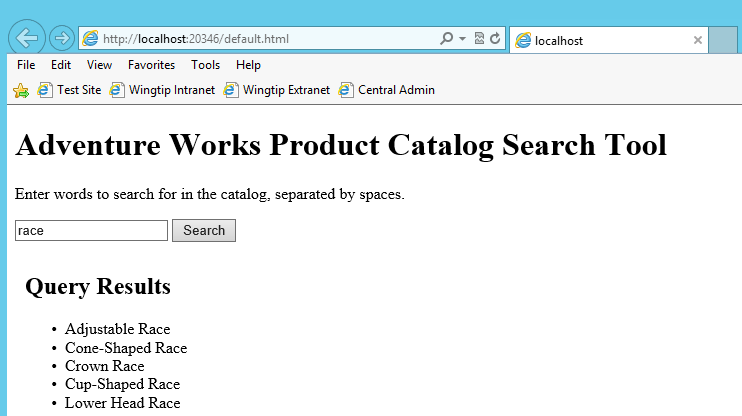
1. Save all changes: **File 🡪 Save All**.

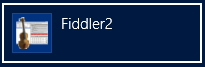
#### Build and Test the Project

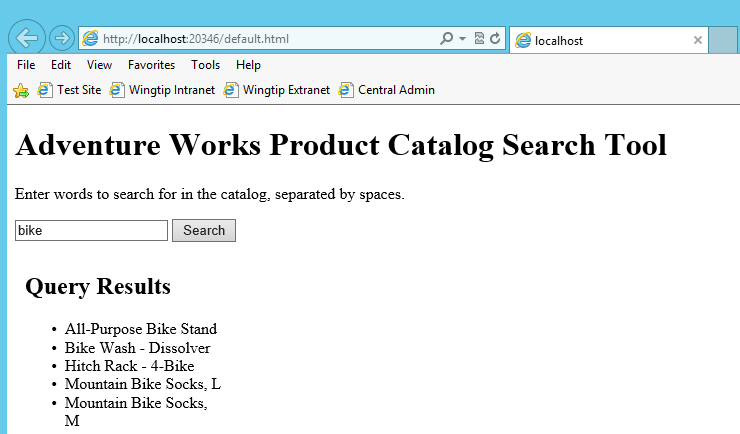
1. Build and test your application by pressing **[F5]** or **Debug 🡪 Start Debugging**.
2. Once the solution has been deployed, Internet Explorer will launch and navigate to the default page for the site. The default view will show the default dummy items in the list and an empty search box:



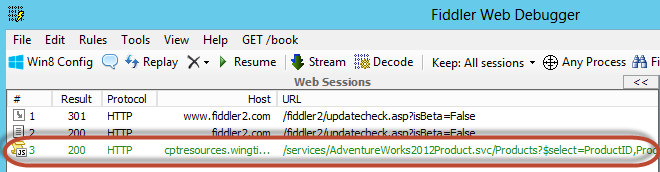
1. Execute the search by entering **race** in the search box and clicking the **search** button.
2. The results should be processed and you should now see results showing in the unordered list:



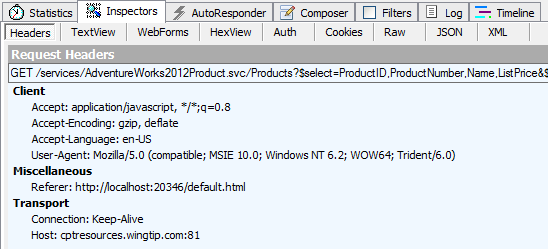
1. Inspect the actual response coming from the service using Fiddler.
   1. Launch **Fiddler**: **Windows Key 🡪** Type **Fiddler2**
   2. Go back to the browser and rerun a search using a different term of **bike**:



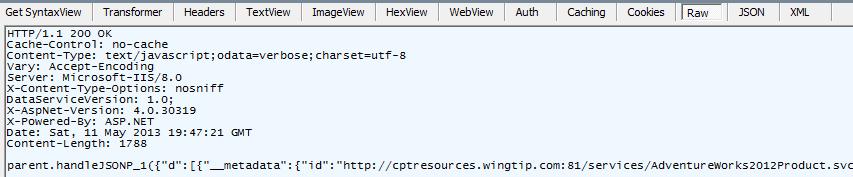
* 1. Switch back to Fiddler and you should see one request in the Web Sessions pane:



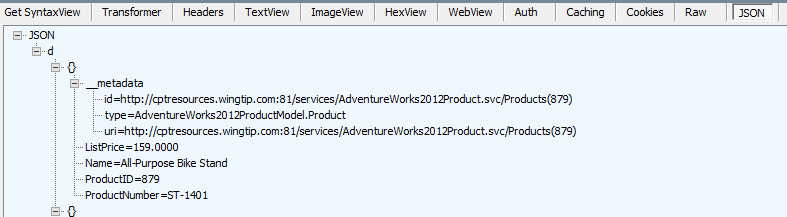
* 1. If you select the session, the right-hand side of Fiddler will show you the request details (on the top) and response details (on the bottom).
  2. For the request portion (the top part of Fiddler), select the **Inspectors** tab and the **Headers** button to see details of the request:



* 1. For the response portion (the bottom part of Fiddler), select first the **Raw** button to see the raw response. You may see a **yellow box** at the top of the Response pane prompting you to click it to decode (i.e. **Click here to transform**) the response. Go ahead and click this yellow box to decode the response. Notice how the results are in the JSON format:



* 1. Now click the JSON button to see the formatted results in a much more readable format:



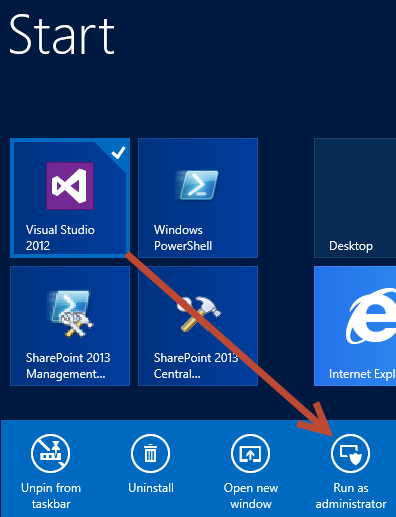
* 1. Close the browser to stop debugging.

In this exercise you used the popular jQuery & datajs JavaScript libraries to issue a cross domain call and update the user interface with the results.

### Exercise 3: Working with the KnockoutJS JavaScript Library

In this exercise you will build off the previous sample to create a more robust client-side solution using the MVVM JavaScript library KnockoutJS.

1. Open an existing starter project in Visual Studio 2012:
   1. Launch **Visual Studio 2012** as administrator:
      1. **Windows Keyboard Key** 🡪 **Right click** on the **Visual Studio 2012** tile and select **Run as administrator**



* 1. Select **File 🡪 Open 🡪 Project/Solution**.
  2. In the **Open Project** dialog, select the following project provided in the files associated with this lab:

[..]\DevPracticesTechniques\Exercises\Ex3\KnockoutJsExample.sln

1. This project is similar to the one you created in the previous exercise. The first step is to add the KnockoutJS JavaScript MVVM library:
   1. Right-click the project **KnockoutJsExample** in the **Solution Explorer** tool window and select **Add 🡪 Existing Item**.
   2. In the **Add Existing Item** dialog add the jQuery library from the files provided for this course:

[..]\ExtraStudentFiles\Resources\Libraries\knockoutjs\knockout-2.1.0.js

1. Add JavaScript library references to the page:
   1. Add the following line to the **<head>** portion of the **default.html** page:

<script src="knockout-2.1.0.js"></script>

KnockoutJS is a JavaScript implementation of the model-view view-model (MVVM) programming pattern. In this pattern you have a clear separation of the model (the data objects) from the view (web page) and glued together using a view model. It greatly simplifies updating the user interface in that you write the code to update the model & the view model automatically updates the view for you.

You could achieve the same thing just using jQuery, but it would mean you have a lot of code to write to update the user interface. Using KnockoutJS, you’ll see you have very little code to write to get the user interface updated.

1. The first step is to create a new Product object that will hold the data. Within the **<script>** tag in the page, just before the JavaScript line that creates the **serviceUri** variable, add the following JavaScript code to create a Product object:

var AWProduct = function (id, name, productNumber, listPrice) {

this.productId = id;

this.productName = name;

this.productNumber = productNumber;

this.listPrice = listPrice;

}

1. Now, create the view model that will be bound to the user interface and update the web page when changes occur. Add the following JavaScript code immediately after the code you just added in the last step:

function AdventureWorksProductsViewModel() {

this.products = ko.observableArray();

this.selectedProduct = ko.observable();

}

1. This is your first real introduction to KnockoutJS. What you did was create two properties in the view model. For **Products**, you flagged it as an **observableArray** which is just like an observable collection in managed code, commonly used in XAML based languages like WPF and Silverlight. For the **SelectedProduct** property you made it a simple **observable**. These observable properties are monitored by the KnockoutJS library. When they see things change they will update the user interface accordingly.
2. With the data structure and view model setup there are three steps left:
   1. **Update HTML controls with data binding notations** – this involves adding special attributes to HTML controls.
   2. **Update the view model with data** – in this scenario this means you need to modify the callback datajs is using to implement some logic after it receives a response from the OData service.
   3. **Create an instance of the view model and bind it to the view** – this is the last step and glues the view model to the controls.
3. Let’s first update the HTML controls.
   1. Look at the top of the page and find the unordered list that looks like this:

<ul id="queryResultsList"></ul>

* 1. Replace this unordered list with a dropdown list as follows:

<select data-bind="options: productViewModel.products,

optionsCaption:'Select product...',

optionsText: 'productName',

value:selectedProduct"></select>

* 1. This drop down list has a new attribute, **data-bind**, which will be used by KnockoutJS. It effectively tells it to do the following:
     1. Bind the dropdown list **options** to the products collection in the view model.
     2. For the very first option in the list, make it a caption, not a real value. This caption should give the user instructions to select something.
     3. For each option, set the text value equal to the **productName** property on the product.
     4. For the selected **value**, bind that to the view model’s **selectedProduct** property.

1. To make things a little interesting make this a master-detail page. Add the following HTML markup to the page. It will bind the controls within the new **<DIV>** to the **seletedProduct** in the view model and the properties for the actual product. This HTML fragment should be a sibling, not a child, to the **<DIV>** that contains the Query Results and look like the picture following the code fragment:

<div data-bind="with: productViewModel.selectedProduct"

style="float: left; width: 300px; padding: 10px; margin-left: 25px;">

<h2>Selected Item Detail:</h2>

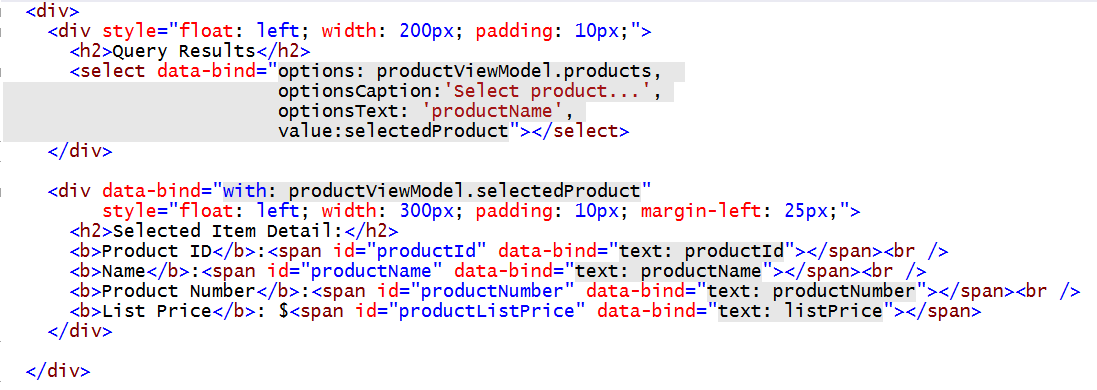
<b>Product ID</b>:<span id="productId" data-bind="text: productId"></span><br />

<b>Name</b>:<span id="productName" data-bind="text: productName"></span><br />

<b>Product Number</b>:<span id="productNumber" data-bind="text: productNumber"></span><br />

<b>List Price</b>: $<span id="productListPrice" data-bind="text: listPrice"></span>

</div>



1. In order to make the binding work, you need to create an instance of the view model and make KnockoutJS aware of all the bindings. Do this by adding the following two lines to the very bottom of the **<script>** block already on the page:

var productViewModel = new AdventureWorksProductsViewModel();

ko.applyBindings(productViewModel);

1. The last step in coding this up is to update the JavaScript JSONP callback to update the **products** collection in the view model with the results.
   1. First, update the code to clear our any previous results. Find the following line:

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

* 1. Replace this line with the following code:

productViewModel.products.removeAll();

* 1. Next, update the code to update the **products** collection in the view model rather than the HTML page. Find the following line:

$("#queryResultsList").append("<li>" + this.Name + "</li>");

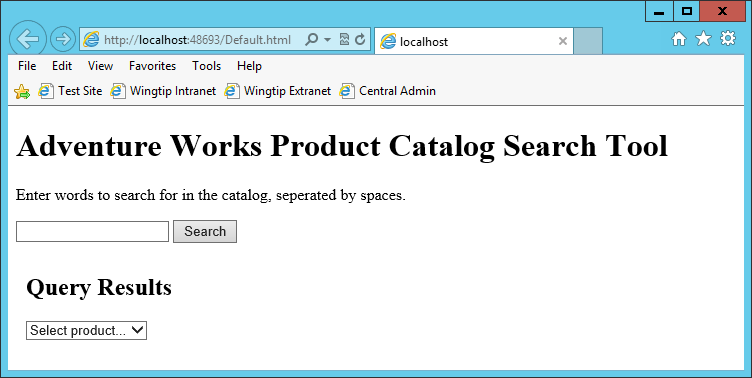
* 1. Replace this line with the following code:

productViewModel.products.push(new AWProduct(this.ProductID, this.Name, this.ProductNumber, this.ListPrice));

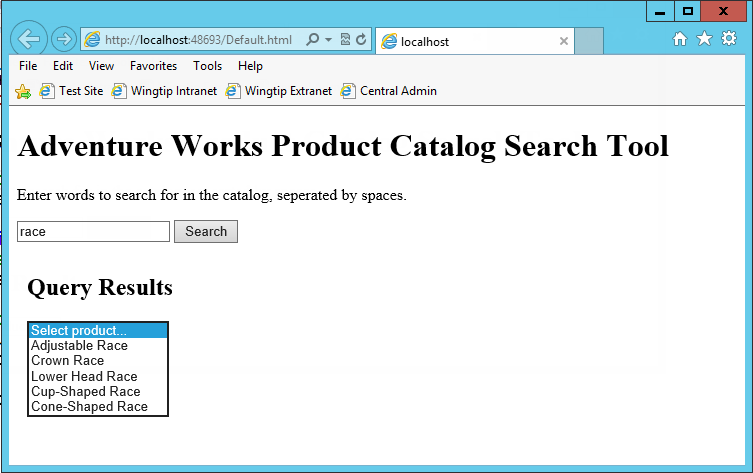
1. Save all changes: **File 🡪 Save All**.

#### Build and Test the Project

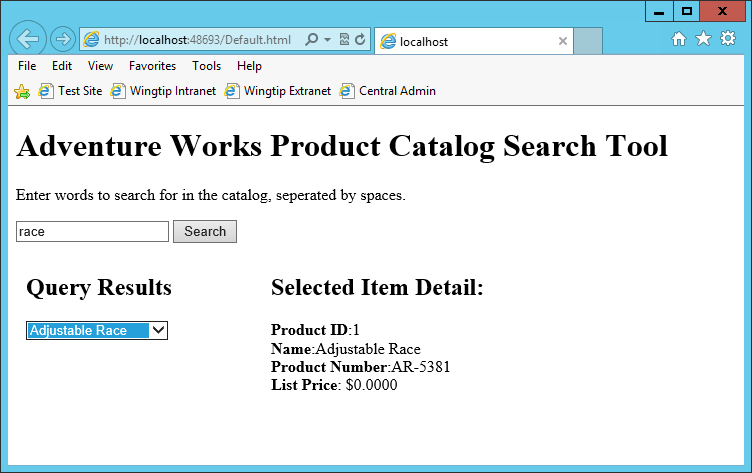
1. Build and test your application by pressing **[F5]** or **Debug 🡪 Start Debugging**.
2. Once the solution has been deployed, Internet Explorer will launch and navigate to the default page for the site. The default view will show the default dummy items in the list and an empty search box:



1. Execute the search by entering **race** in the search box and clicking the **search** button.
2. The results should be processed and you should now see results showing in the dropdown list:



1. Select one of the items to see the detail view appear:



1. Now try changing your query to **bike** and selecting a few items.
2. Close the browser to stop debugging.

In this exercise you used the popular KnockoutJS JavaScript library to issue a cross domain call and update the user interface with the results.