

Creating a SharePoint-hosted Add-in using AngularJS

Lab Time: 45 minutes

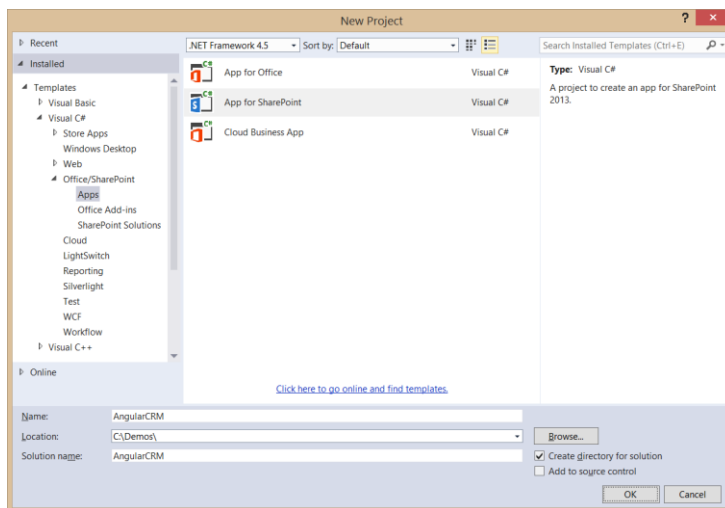
Lab Folder: C:\Student\Modules\05_Angular\Lab

Lab Overview: In this lab you will get hands-on experience working with Bootstrap and AngularJS by developing a SharePoint-hosted App which plays the role of a customer relationship management system (CRM).

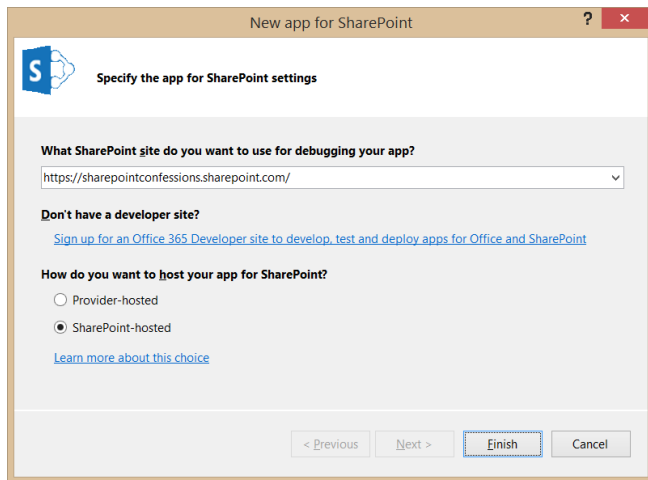
Exercise 1: Creating a SharePoint-hosted App using Bootstrap and AngularJS

In this exercise you will create a SharePoint-hosted app and configure it to use Bootstrap and the AngularJS framework.

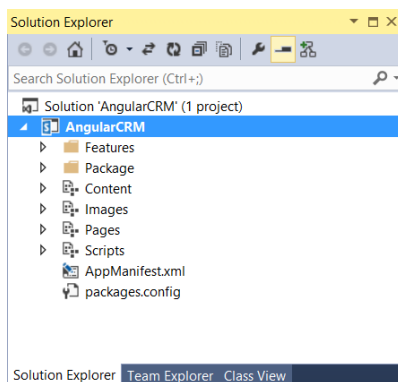
1. Launch Visual Studio as administrator.
2. Create a new project in Visual Studio by selecting the menu command **File > New > Project**.
3. In the **New Project** dialog
 - a) Select the **App for SharePoint** project template under the **Templates > Visual C# > Office / SharePoint > Apps** section.
 - b) Enter a name of **AngularCRM**
 - c) Enter a location of **C:\Student\Modules\05_Angular\Lab**.
 - d) Click the **OK** button.



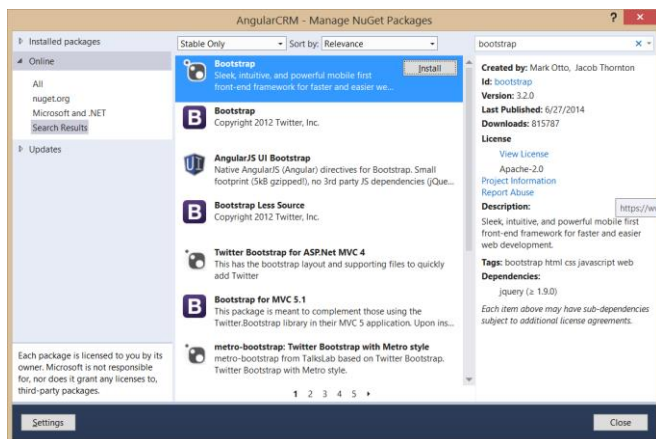
4. In the **New app for SharePoint wizard**, enter the URL of <https://dev.wingtip.com> and select **SharePoint-hosted** for the app hosting model. When done, complete the wizard by clicking the **Finish** button.



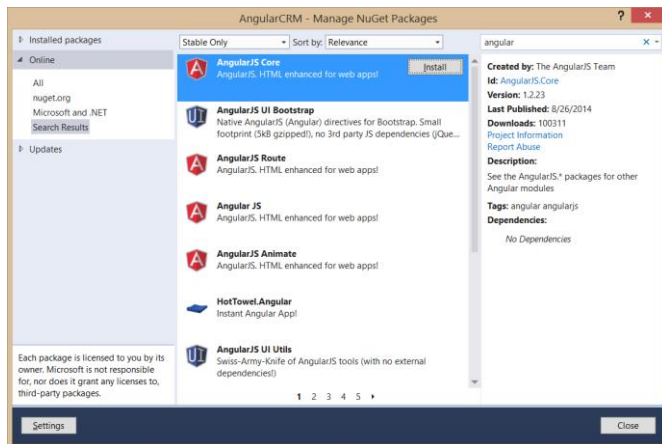
5. Examine the default project setup for the new SharePoint-Hosted app. As you can see, it is like a traditional SharePoint solution-based project because you have a **Features** and **Packages** node. Note that there are project folders named **Content**, **Images** & **Pages** are actually SharePoint Project Modules that will provision their contents to the respective folders in the app web when the app is installed.



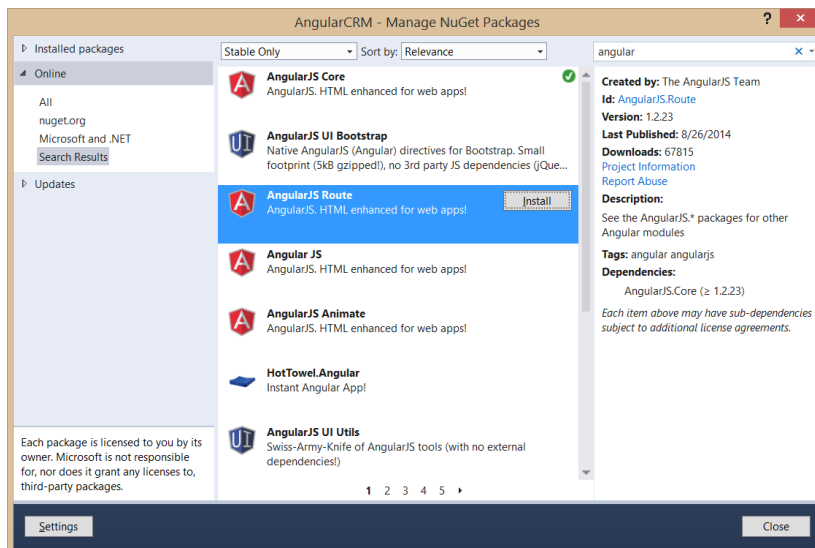
6. Add the NuGet Packages for Bootstrap and AngularJS.
- Right-click on the **AngularCRM** project in the solution Explorer and select **Manage NuGet Packages**.
 - Using the **Manage NuGet Packages** dialog, install the NuGet package for **Bootstrap**.



- c) Next, install the NuGet package for **AngularJS Core**.

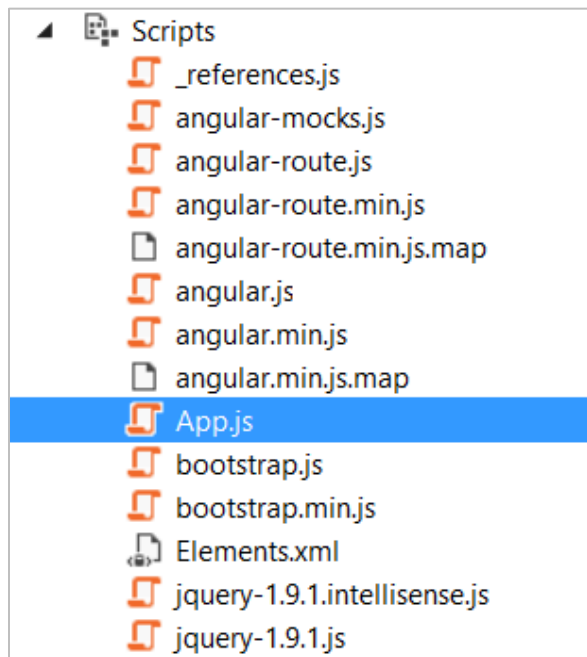


d) Finally, install the NuGet package for AngularJS Route.

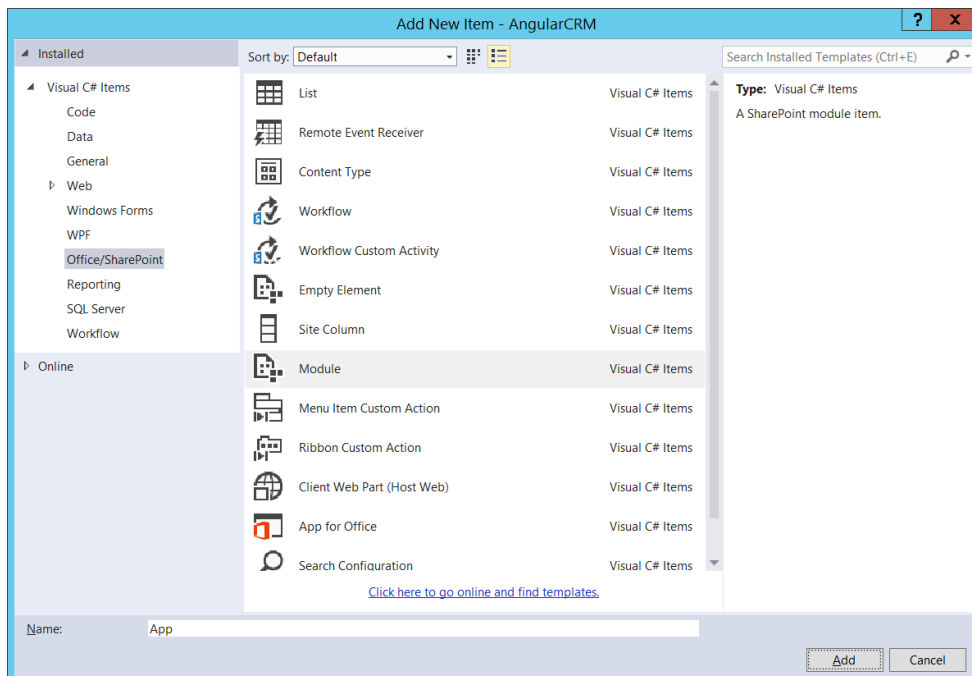


e) Close the **Manage NuGet Packages** dialog.

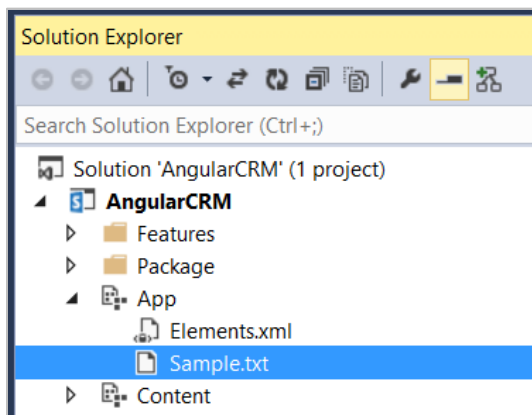
7. Your app will not be using anything from the **Pages** folder. Therefore, you should delete the **Pages** folder from your project by right-clicking on it in the Solution Explorer and selecting the **Delete** command.
8. Delete the **App.js** file from the **Scripts** folder.
 - a) Open the Scripts folder and inspect what's inside.
 - b) Note that there is a **App.js** file that you will not be using in this project. Delete the **App.js** file.



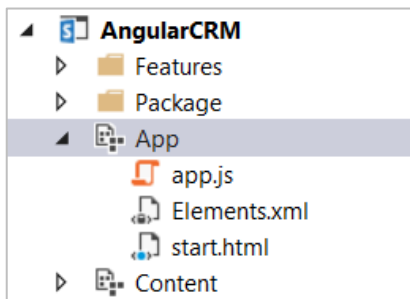
9. Add a new SharePoint module named **App**.
- In the solution Explorer, right-click on the top-level node of the **AngularCRM** project and click **Add > New Item**.
 - In the **Add New Item** dialog, select the **Module** template and enter the name **App** as shown in the following screenshot. Click the **Add** button when you are done.



- Once the **App** module has been created in the project, you should notice that it already contains a file named **Elements.xml** and a second file **Sample.txt**. Delete the file named **Sample.txt**.



10. Add a new HTML file named **start.html** into the folder for the **App** module.
11. Add a new JavaScript file named **App.js** into the folder for the **App** module.



12. Open **start.html** in an editor windows and modify the head section to match the following code listing.

```
<head>
  <meta charset="utf-8" />
  <meta http-equiv="X-UA-Compatible" content="IE=10" />
  <title>Angular CRM</title>
</head>
```

13. Inside the **head** section, add two links to the CSS files in the Content folder named **bootstrap.css** and **App.css**.

```
<head>
  <meta charset="utf-8" />
  <meta http-equiv="X-UA-Compatible" content="IE=10" />
  <title>Angular CRM</title>

  <link href="../../Content/bootstrap.css" rel="stylesheet" />
  <link href="../../Content/App.css" rel="stylesheet" />
</head>
```

14. Add in **script** links to the JavaScript files in the scripts folder for jQuery, as well as **bootstrap.js**, **angular.js** and **angular.route.js**. Also, add a link to the **App.js** file which is located in the **App** folder.

```
<head>
  <meta charset="utf-8" />
  <meta http-equiv="X-UA-Compatible" content="IE=10" />
  <title>Angular CRM</title>

  <link href="../../Content/bootstrap.css" rel="stylesheet" />
  <link href="../../Content/App.css" rel="stylesheet" />

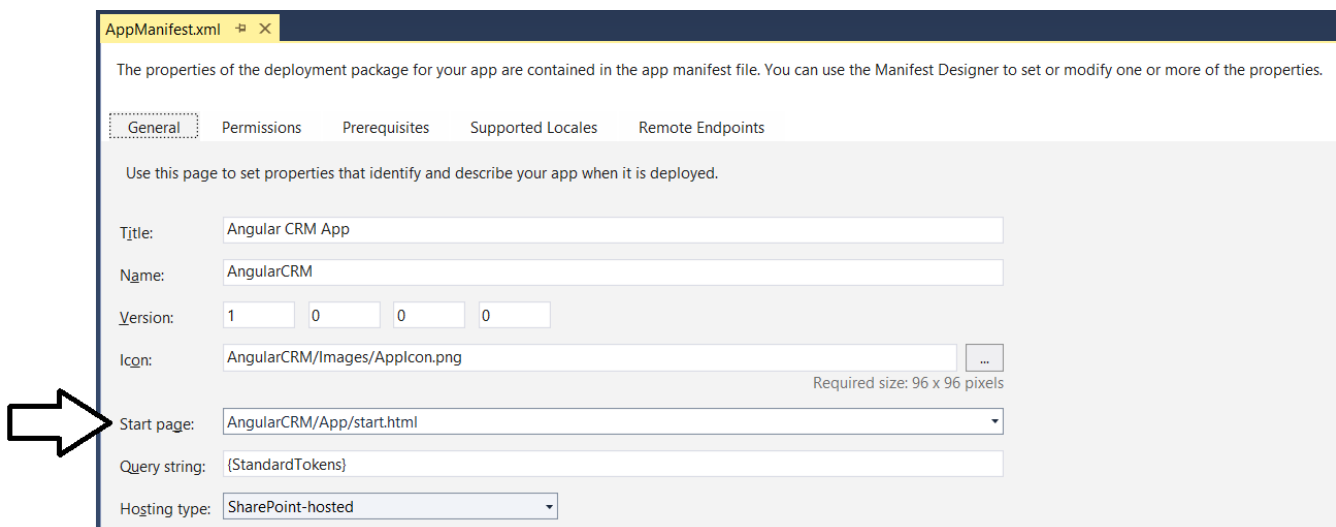
  <script src="../../Scripts/jquery-1.9.1.js"></script>
  <script src="../../Scripts/bootstrap.js"></script>
  <script src="../../Scripts/angular.js"></script>
```

```
<script src="../../Scripts/angular-route.js"></script>
<script src="App.js"></script>
</head>
```

15. Update the **body** element by adding the following HTML layout which uses predefined Bootstrap styles such as **container**, **container-fluid** and **navbar**. If you'd rather note type all this code by hand, you can copy it from the **start.html.body.txt** files in the **StarterFiles** folder for this lab.

```
<body>
  <div class="container">
    <div class="navbar navbar-default" role="navigation">
      <div class="container-fluid">
        <div class="navbar-header">
          <a class="navbar-brand" href="#/">Angular CRM</a>
        </div>
        <div class="navbar-collapse collapse">
          <ul class="nav navbar-nav">
            <li><a href="#/">Home</a></li>
            <li><a href="#/">Add Customer</a></li>
            <li><a href="#/">About</a></li>
          </ul>
          <ul class="nav navbar-nav navbar-right">
            <li><a id="lnkHostWeb">Back to Host Web</a></li>
          </ul>
        </div>
      </div>
    </div>
  </div>
  <div class="container">
    <div id="content-box" />
  </div>
</body>
```

16. Save and close **start.html**.
17. Open up **AppManifest.xml** in the designer and update the app's **Start page** setting to point to **start.html**. Also, update the **Title** to something more readable such as **Angular CRM App**.



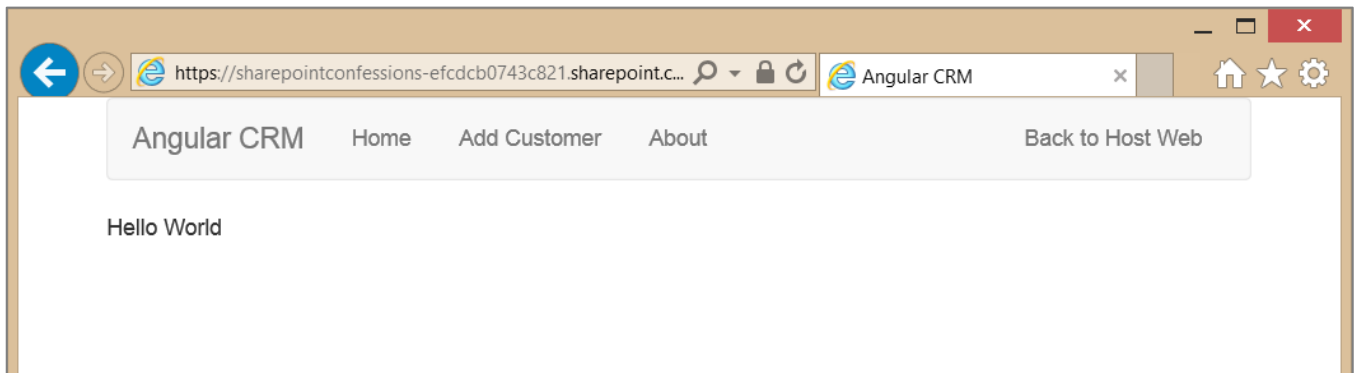
The screenshot shows the 'AppManifest.xml' Manifest Designer. The 'General' tab is selected. The 'Start page' dropdown menu is highlighted with a large white arrow. The 'Start page' is currently set to 'AngularCRM/App/start.html'. Other fields include 'Title' (Angular CRM App), 'Name' (AngularCRM), 'Version' (1.0.0.0), 'Icon' (AngularCRM/Images/AppIcon.png), 'Query string' (StandardTokens), and 'Hosting type' (SharePoint-hosted).

18. Save and close **AppManifest.xml**.
19. Open **App.js** in an editor window and update its contents to match the following code listing.

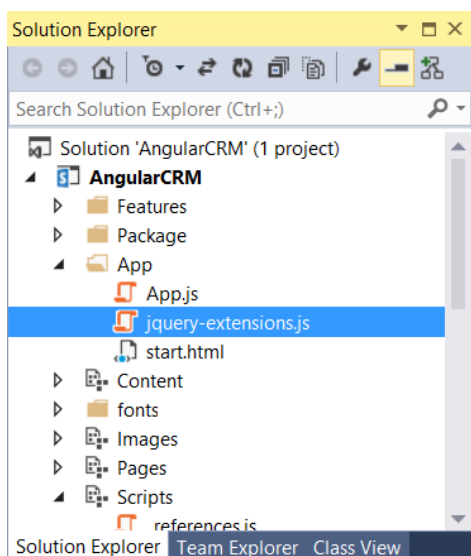
```
'use strict';
angular.element(document).ready( function () {
```

```
$("#content-box").text("Hello world");  
});
```

20. It's now time to test the app. Press **{F5}** to begin a new debugging sessions. Once the app has been installed, Visual Studio will start up Internet Explorer and redirect you to the app's start page. You should see the app's navbar and the "Hello World" test message as shown in the following screenshot.



21. Close the browser window to terminate the debugging session and return to Visual Studio.
22. Your app will need to read query string parameters. Therefore, you will create a simple jQuery extension to add a few helper functions. Begin by creating a new JavaScript file into the App folder named **jquery-extensions.js**.



23. Add the following code to **jquery-extensions.js** to extend the jQuery library with two helper methods named **getQueryStringValue** and **getQueryStringValues**. If you'd rather not type all this code by hand, you can copy it from the **jquery-extensions.js.txt** file in the **StarterFiles** folder for this lab.

```
$.extend({  
  getQueryStringValues: function () {  
    var vars = [], hash;  
    var hashes = window.location.href.slice(window.location.href.indexOf('?') + 1).split('&');  
    for (var i = 0; i < hashes.length; i++) {  
      hash = hashes[i].split('=');  
      vars.push(hash[0]);  
      vars[hash[0]] = hash[1];  
    }  
    return vars;  
  },  
});
```

```
getQueryStringValue: function (name) {  
    return decodeURIComponent($.getQueryStringValues()[name]);  
}  
  
});
```

24. Open **start.html** and add a new script link for named **jquery-extensions.js** right after the script link for the jQuery library.

```
<head>  
  <meta charset="utf-8" />  
  <meta http-equiv="X-UA-Compatible" content="IE=10" />  
  <title>Angular CRM</title>  
  <link href="../../Content/bootstrap.css" rel="stylesheet" />  
  <link href="../../Content/App.css" rel="stylesheet" />  
  <script src="../../Scripts/jquery-1.9.1.js"></script>  
  
  <script src="jquery-extensions.js"></script>  
  
  <script src="../../Scripts/bootstrap.js"></script>  
  <script src="../../Scripts/angular.js"></script>  
  <script src="../../Scripts/angular-route.js"></script>  
  <script src="App.js"></script>  
</head>
```

25. Save and close **start.html**.

26. Open **App.js** and update its contents to match the following code listing.

```
'use strict';  
  
angular.element(document).ready( function () {  
    var hostweb = $.getQueryStringValue("SPHostUrl");  
    $("#lnkHostweb").attr("href", hostweb);  
});
```

27. It's time again to test the app. Press **{F5}** to begin a new debugging sessions. Once the app has been installed, Visual Studio will start up Internet Explorer and redirect you to the app's start page. At this point, you should be able to click the **Back to Host Web** link on the right-hand side of the navbar and successfully navigate back to the host web which you are using for your testing.
28. Close the Internet explorer to terminate the debugging session and return to Visual studio.
29. The final steps in this exercise will involve adding the **ng-app** directive to the **body** element of **start.html** and adding JavaScript code to properly initialize the app while the Angular framework is loading. Begin by opening **start.html** in an editor window.
30. Locate the opening tag of the body element and add the **ng-app** directive with an app name of **AngularCRM**.

```
<body ng-app="AngularCRM">
```

31. Save your changes to **start.html**.

32. Open **App.js**. Remove all the existing code and replace it with the code shown in the following code listing.

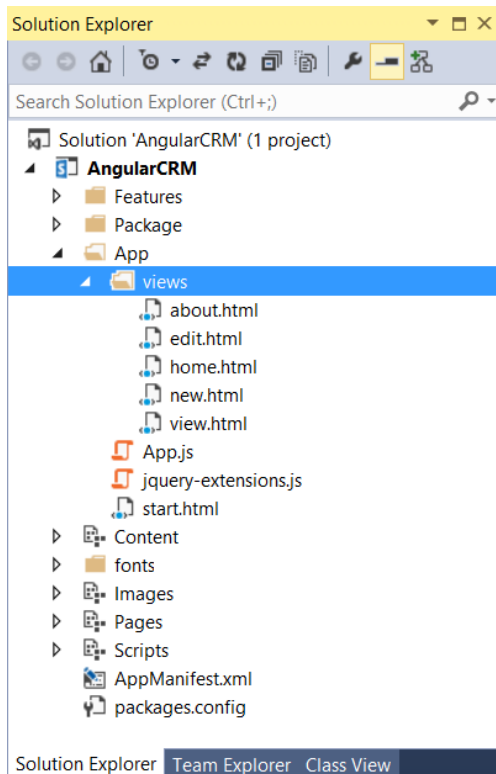
```
'use strict';  
  
var crmApp = angular.module("AngularCRM", []);  
  
crmApp.config(function () {  
    var hostweb = $.getQueryStringValue("SPHostUrl");  
    $("#lnkHostweb").attr("href", hostweb);  
});
```

33. Test the app by pressing **{F5}** to begin a new debugging sessions. Once the app has been installed, Visual Studio will start up Internet Explorer and redirect you to the app's start page. Test to ensure you can still click the **Back to Host Web** link and successfully navigate back to the host web. It should work just as it did before, it's just now you are using a more angular-specific way of initializing the app.
34. Close the Internet explorer to terminate the debugging session and return to Visual studio.

Exercise 2: Working with Views, Controllers and Routing

In this lab, you will continue working with the AngularCRM project you created in the previous lab exercise. You will extend this project by adding several new views and controllers and configuring the app's routing scheme.

1. Open the **AngularCRM** project in Visual Studio if it is not already open.
2. Create a new folder named **views** inside the **App** folder.
3. Create five new HTML files inside the **views** folder named **about.html**, **edit.html**, **home.html**, **new.html** and **view.html**.



4. Update the contents of **home.html** to match the following listing and save your changes.

```
<h3>Customer List</h3>
```

5. Update the contents of **new.html** to match the following listing and save your changes.

```
<h3>New Customer</h3>
```

6. Update the contents of **view.html** to match the following listing and save your changes.

```
<h3>View Customer</h3>
```

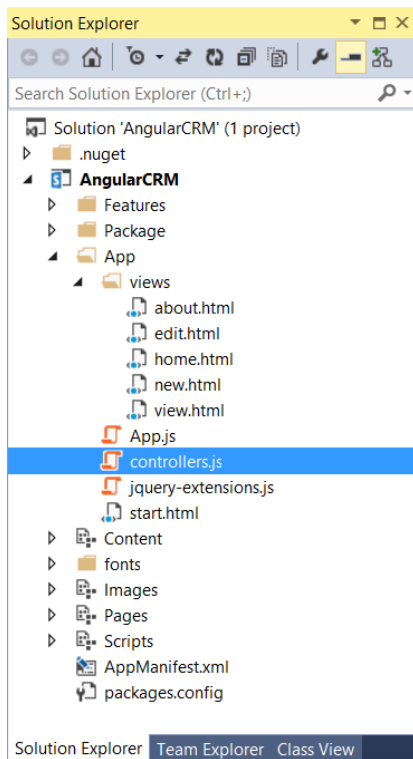
7. Update the contents of **edit.html** to match the following listing and save your changes.

```
<h3>Edit Customer</h3>
```

Update the contents of **about.html** to match the following listing and save your changes.

```
<h3 ng-bind="title"></h3>  
<p ng-bind="description"></p>
```

8. Create a new JavaScript file named **controllers.js** in the **App** folder.



9. Add the following code into controllers.js to provide a controller starting point for each of the views. If you'd rather not type all this code by hand, you can copy it from the **controllers_starter.js.txt** file in the **StarterFiles** folder for this lab.

```
'use strict';

var app = angular.module('AngularCRM');

app.controller('homeController',
  function ($scope) {
  }
);

app.controller('newController',
  function ($scope) {
  }
);

app.controller('viewController',
  function ($scope) {
  }
);

app.controller('editController',
  function ($scope) {
  }
);

app.controller('aboutController',
  function ($scope) {
  }
);
```

10. Implement the **aboutController** controller function to create a **title** property and a **description** property on the **\$scope** variable and to initialize their values using string literals.

```
app.controller('aboutController',
  function ($scope) {
```

```
$scope.title = "About the Angular CRM App"
$scope.description = "The Angular CRM App is a demo app which uses Bootstrap and AngularJS"
}
);
```

11. Update the head section in **start.html** to include a script link for **controllers.js** just after the script link to App.js and then save your changes.

```
<script src="App.js"></script>
<script src="controllers.js"></script>
```

12. Now that you have created the views and controllers, it's time to configure the app's routing scheme. Begin by opening App.js in an editor window.

13. Locate the line of code which calls `angular.module` and update it to include a dependency on `ngRoute` module.

```
var crmApp = angular.module("AngularCRM", ['ngRoute']);
```

14. Locate the line of code which calls `angular.config` and modify the controller function parameter list to accept a parameter named `$routeProvider`.

```
crmApp.config(function ($routeProvider) {
```

15. At this point, the code in App.js should match the following code listing.

```
'use strict';

var crmApp = angular.module("AngularCRM", ['ngRoute']);

crmApp.config(function ($routeProvider) {

    var hostWeb = $.getQueryStringValue("SPHostUrl");
    $("#lnkHostWeb").attr("href", hostWeb);

});
```

16. Add the following code to configure the app's routing map just after the code which updates the href attribute of the anchor element with an id of **lnkHostWeb**. If you'd rather not type all this code by hand, you can copy it from the **routing_map.js.txt** file in the **StarterFiles** folder for this lab.

```
$routeProvider.when("/", {
    templateUrl: 'views/home.html',
    controller: "homeController"
}).when("/view/:id", {
    templateUrl: 'views/view.html',
    controller: "viewController"
}).when("/edit/:id", {
    templateUrl: 'views/edit.html',
    controller: "editController"
}).when("/new", {
    templateUrl: 'views/new.html',
    controller: "newController"
}).when("/about", {
    templateUrl: 'views/about.html',
    controller: "aboutController"
}).otherwise({
    redirectTo: "/"
});
```

When you are done, the code you have written in App.js should match the following code listing.

```
'use strict';

var crmApp = angular.module("AngularCRM", ['ngRoute']);

crmApp.config(function ($routeProvider) {

    var hostWeb = $.getQueryStringValue("SPHostUrl");
```

```
$("#lnkHostWeb").attr("href", hostweb);

// config route map
$routeProvider.when("/", {
  templateUrl: 'views/home.html',
  controller: "homeController"
}).when("/view/:id", {
  templateUrl: 'views/view.html',
  controller: "viewController"
}).when("/edit/:id", {
  templateUrl: 'views/edit.html',
  controller: "editController"
}).when("/new", {
  templateUrl: 'views/new.html',
  controller: "newController"
}).when("/about", {
  templateUrl: 'views/about.html',
  controller: "aboutController"
}).otherwise({
  redirectTo: "/"
});
});
```

17. The last remaining task before testing is to update the navigation links on **start.html**. Begin by opening **start.html** and locating the elements with the links titled **Home**, **Add Customer** and **About**.
18. Update the href attribute for the Add Customer link to **#/new** and update the href attribute for the About link to **#/about**.

```
<div class="navbar-collapse collapse">
  <ul class="nav navbar-nav">
    <li><a href="#">Home</a></li>
    <li><a href="#/new">Add Customer</a></li>
    <li><a href="#/about">About</a></li>
```

19. Down in the body section of start.html, locate the div element with the id of content-box and add the ng-view directive.

```
<div class="container">
  <div id="content-box" ng-view ></div>
</div>
```

20. Save your changes to start.html.
21. Test the routing scheme of the app by pressing {F5} to begin a new debugging sessions. Once the app has been installed, Visual Studio will start up Internet Explorer and redirect you to the app's start page. At this point, you should be able to click on the navbar links titled Home, Add Customer and About to navigate between these three views.
22. Click on the About link to navigate to the app's About page. You should be able to verify that the about view is properly displaying the values of the title property and the description property that were written to the \$scope variable by aboutController.

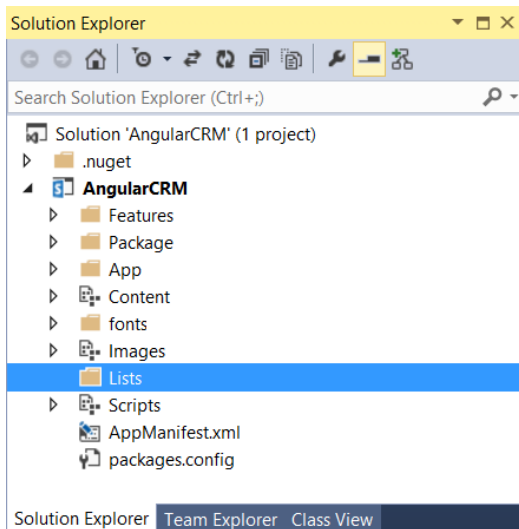


You have now successfully set up the routing scheme for the app. Close the Internet explorer to terminate the debugging session and return to Visual studio.

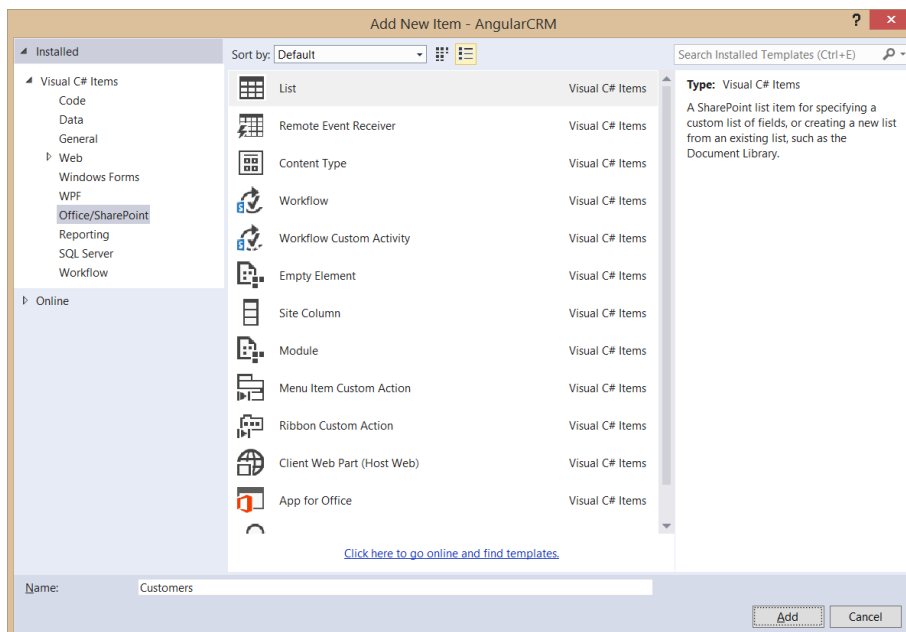
Exercise 3: Extending the AngularCRM Project with a Custom Service

In this exercise you will extend the AngularCRM app by adding a SharePoint list and then creating a custom angular service to read and write items to and from the list.

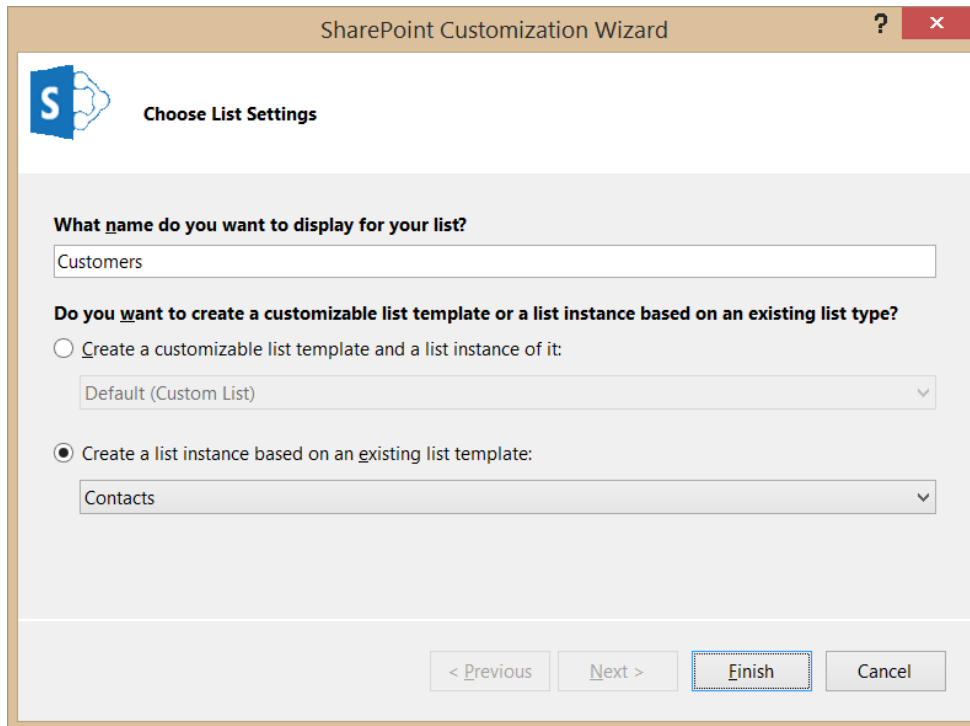
1. Open the AngularCRM project in Visual Studio if it is not already open.
2. Create a new top-level folder named Lists at the root of the AngularCRM project.



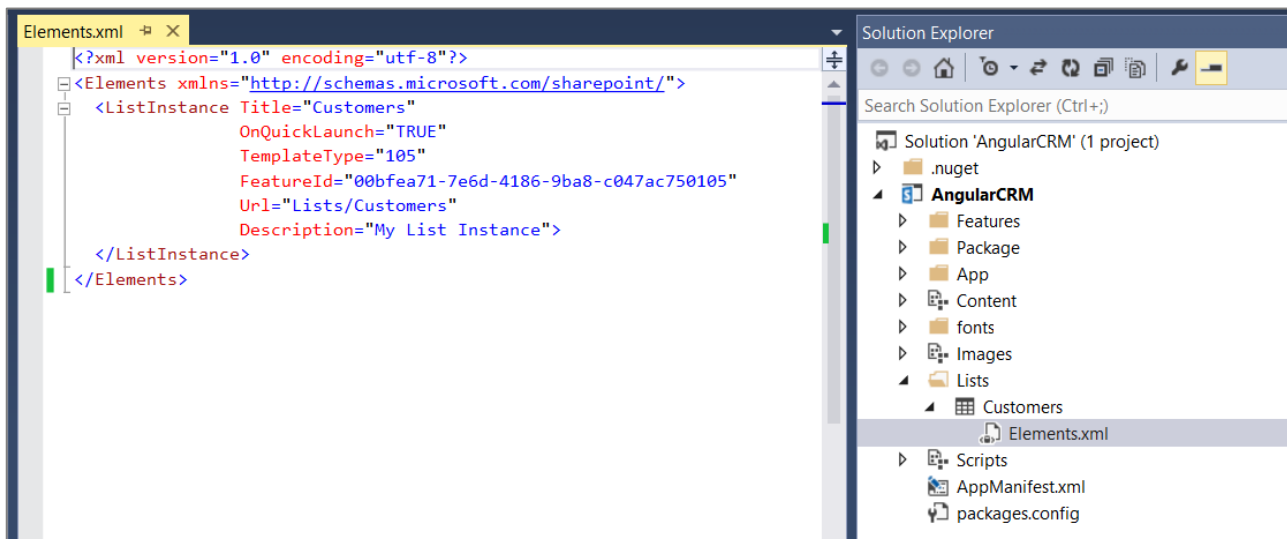
3. Right-click on the Lists folder and select Add > New Item.
4. In the Add New Item dialog, click the Office/SharePoint group on the left and then select the List template. Enter a name of Customers and click Add.



5. In the SharePoint Customization Wizard dialog, enter a list display name of Customers. Select the option Create a list instance based on an existing list template and set the list template type to Contacts.

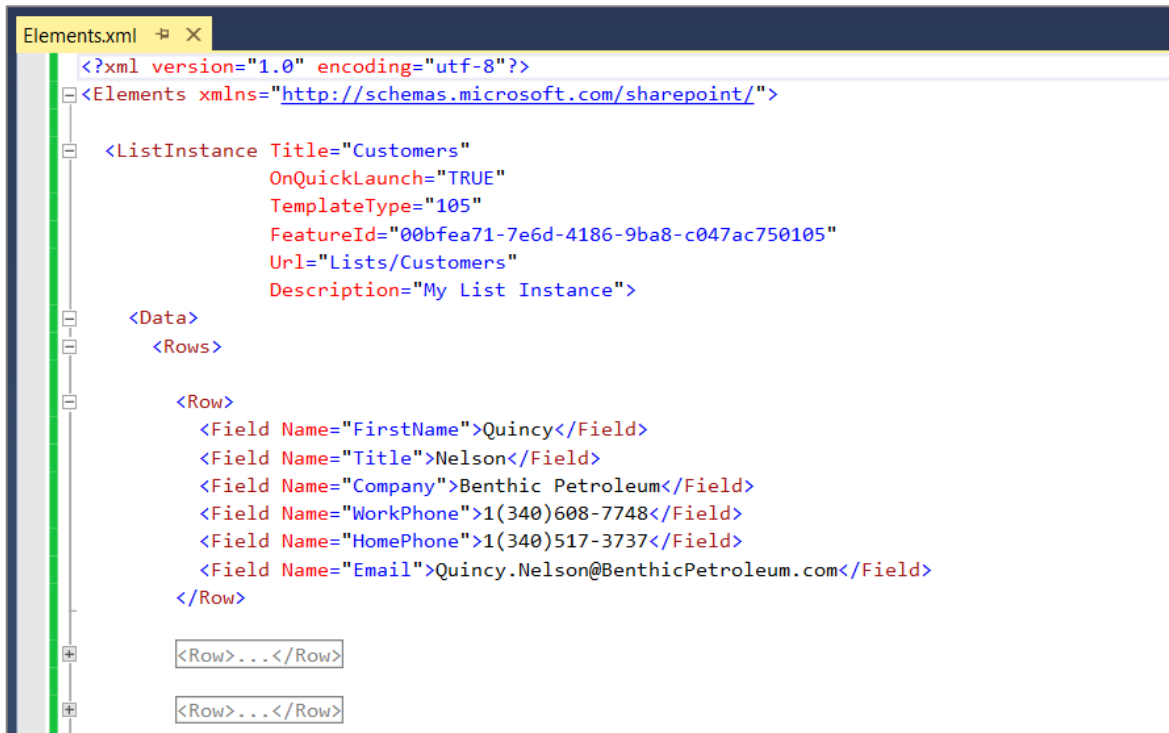


6. Click the Finish button in the SharePoint Customization Wizard to create the new project item for the list. Inside the Lists folder, you should be able to see a Customers folder which contains an element manifest named elements.xml.



7. In this step you will modify the elements.xml so that the Customers list will be created with a pre-populated set of customer items.
8. Using Windows Explorer, look inside the Starter Files folder for this lab and locate the file named Customers_Elements.xml.txt.
- Open Customers_Elements.xml.txt in NOTEPAD and copy all its contents into the Windows clipboard.
 - Return to Visual Studio and make sure that the elements.xml file for the Customers list is open in an editor window.
 - Delete all the existing content from elements.xml.
 - Paste in the contents of the clipboard.
 - Save your changes to elements.xml.

9. Look at the XML content inside elements.xml and examine how it uses a Data element with an inner Rows element to pre-populate the Customers list with a set of sample customer items to assist in your testing and debugging.



```
<?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"
    Description="My List Instance">

    <Data>
      <Rows>

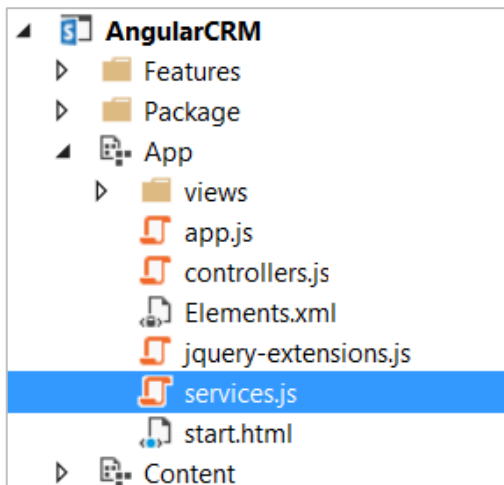
        <Row>
          <Field Name="FirstName">Quincy</Field>
          <Field Name="Title">Nelson</Field>
          <Field Name="Company">Benthic Petroleum</Field>
          <Field Name="WorkPhone">1(340)608-7748</Field>
          <Field Name="HomePhone">1(340)517-3737</Field>
          <Field Name="Email">Quincy.Nelson@BenthicPetroleum.com</Field>
        </Row>

        <Row>...</Row>

        <Row>...</Row>

      </Rows>
    </Data>
  </ListInstance>
</Elements>
```

10. Save and close elements.xml.
11. In the Apps folder, create a new JavaScript file named services.js.



12. Open start.html and add a script link for services.js just after the script link to controllers.js.

```
<script src="App.js"></script>
<script src="controllers.js"></script>
<script src="services.js"></script>
```

13. Save and close start.html.
14. Open **services.js** in an editor window.

15. Add the following code into **services.js** to provide a starting point for your service implementation.

```
'use strict';

var app = angular.module('AngularCRM');

app.factory("wingtipCrmService",
function ($http) {
    // create service object
    var service = {};

    // TODO: add behavior to service object

    // return service object to angular framework
    return service;
});
```

16. Add the following code just below the line with the comment **TODO: add behavior to service object**. This code will initialize the service by retrieving and caching a form digest value in a variable named **requestDigest**.

```
// retrieve and cache SharePoint form digest value
var requestDigest;

$http({
    method: 'POST',
    url: "../_api/contextinfo",
    headers: { "Accept": "application/json; odata=verbose" }
}).success(function (data) {
    requestDigest = data.d.GetContextWebInformation.FormDigestValue
});
```

17. Just below the code you added in the previous step, add the following code to add a function to the service named **getCustomers**.

```
service.getCustomers = function () {
    var restQueryUrl = "../_api/web/lists/getByTitle('Customers')/items/" +
        "?$select=ID,Title,FirstName,WorkPhone,HomePhone,Email";

    return $http({
        method: 'GET',
        url: restQueryUrl,
        headers: { "Accept": "application/json; odata=verbose" }
    })
}
```

18. Open **controllers.js** in an editor window.

19. Locate the code which defines the controller function for **homeController**. In the function parameter list, add a second parameter named **wingtipCrmService** in addition to the **\$scope** parameter that is already defined.

```
app.controller('homeController',
function ($scope, wingtipCrmService) {
}
);
```

20. Implement the **homeController** function as shown in the following code listing.

```
app.controller('homeController',
function ($scope, wingtipCrmService) {
    wingtipCrmService.getCustomers().success(function (data) {
        $scope.customers = data.d.results;
    });
}
);
```

21. Open **Home.html** in an editor window. Copy the following HTML layout and paste it into **Home.html** to replace the existing content. If you'd rather not type all this code by hand, you can copy it from the **home.html.txt** file in the **StarterFiles** folder for this lab.

```
<h3>Customer List</h3>

<table class="table table-striped table-hover table-responsive ">
```



```
<thead>
  <tr>
    <td>ID</td>
    <td>First Name</td>
    <td>Last Name</td>
    <td>Work Phone</td>
    <td>Home Phone</td>
    <td>Email Address</td>
    <td>&nbsp;</td>
    <td>&nbsp;</td>
    <td>&nbsp;</td>
  </tr>
</thead>
<tbody ng-repeat="customer in customers">
  <tr>
    <td>{{customer.Id}}</td>
    <td>{{customer.FirstName}}</td>
    <td>{{customer.Title}}</td>
    <td>{{customer.WorkPhone}}</td>
    <td>{{customer.HomePhone}}</td>
    <td>{{customer.Email}}</td>
    <td><a href="#/view/{{customer.Id}}" class="btn-mini">View</a></td>
    <td><a href="#/edit/{{customer.Id}}" class="btn-mini">Edit</a></td>
    <td><a href="#" data-ng-click="deleteCustomer(customer.Id)" class="btn-mini">Delete</a></td>
  </tr>
</tbody>
</table>
```

22. Test your work by pressing {F5} to begin a debugging session. The app should initialize the start page using the view defined in Home.html which should display a table of customers as shown in the following screenshot.

Angular CRM

Home

Add Customer

About

Back to Host Web

Customer List

ID	First Name	Last Name	Work Phone	Home Phone	Email Address			
1	Quincy	Nelson	1(340)608-7748	1(340)517-3737	Quincy.Nelson@BenthicPetroleum.com	View	Edit	Delete
2	Jude	Mason	1(203)408-0466	1(203)411-0071	Jude.Mason@CyberdyneSystems.com	View	Edit	Delete
3	Sid	Stout	1(518)258-6571	1(518)376-8576	Sid.Stout@Roxxon.com	View	Edit	Delete
4	Gilberto	Gillespie	1(270)510-1720	1(270)755-7810	Gilberto.Gillespie@ShinraElectricPowerCompany.com	View	Edit	Delete
5	Diane	Strickland	1(407)413-4851	1(407)523-5411	Diane.Strickland@Izon.com	View	Edit	Delete
6	Jacqueline	Zimmerman	1(844)234-0550	1(844)764-3522	Jacqueline.Zimmerman@ZorgIndustries.com	View	Edit	Delete
7	Naomi	Schroeder	1(204)355-6648	1(204)356-2831	Naomi.Schroeder@ComTron.com	View	Edit	Delete
8	Lynne	Stephens	1(407)787-7308	1(407)732-1700	Lynne.Stephens@TradeFederation.com	View	Edit	Delete
9	Luther	Sullivan	1(323)755-3404	1(323)684-7814	Luther.Sullivan@Metacortex.com	View	Edit	Delete
10	Rose	Parsons	1(802)357-5583	1(802)727-0246	Rose.Parsons@HansoFoundation.com	View	Edit	Delete

23. Close Internet Explorer to terminate your debugging session and return to Visual Studio.
24. Now it is time to implement the remaining functionality needed for the service. Begin by opening **services.js** and positioning your cursor just below the **getCustomers** function you added earlier.
25. Add the following function implementation to the service for the **getCustomer** function.

```
service.getCustomer = function (id) {
  var restQueryUrl = ".../_api/web/lists/getByTitle('Customers')/items(" + id + ")/" +
    "?$select=ID,Title,FirstName,WorkPhone,HomePhone,Email";
  return $http({
    method: 'GET',
    url: restQueryUrl,
```

```
    headers: { "Accept": "application/json; odata=verbose" }  
  })  
}
```

26. Add the following function implementation to the service for the **deleteCustomer** function.

```
service.deleteCustomer = function (id) {  
  var restQueryUrl = "../_api/web/lists/getByTitle('Customers')/items(" + id + ")";  
  return $http({  
    method: 'DELETE',  
    url: restQueryUrl,  
    headers: {  
      "Accept": "application/json; odata=verbose",  
      "X-RequestDigest": requestDigest,  
      "If-Match": "*"   
    }  
  });  
}
```

27. Add the following function implementation to the service for the addCustomer function. If you'd rather not type all this code by hand, you can copy it from the **addCustomer.js.txt** file in the **StarterFiles** folder for this lab.

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

28. Add the following function implementation to the service for the updateCustomer function. If you'd rather not type all this code by hand, you can copy it from the **updateCustomer.js.txt** file in the **StarterFiles** folder for this lab.

```
service.updateCustomer = function (id, FirstName, LastName, WorkPhone, HomePhone, Email, etag) {  
  var restQueryUrl = "../_api/web/lists/getByTitle('Customers')/items(" + id + ")";  
  
  var customerData = {  
    __metadata: { "type": "SP.Data.CustomersListItem" },  
    Title: LastName,  
    FirstName: FirstName,  
    WorkPhone: WorkPhone,  
    HomePhone: HomePhone,  
    Email: Email  
  };  
  
  var requestBody = JSON.stringify(customerData);  
  
  return $http({  
    method: 'POST',  
    url: restQueryUrl,  
    contentType: "application/json;odata=verbose",  
    data: requestBody,  
    headers: {  
      "Accept": "application/json; odata=verbose",  
      "X-RequestDigest": requestDigest,  
      "If-Match": etag  
    }  
  });  
}
```

```
    data: requestBody,
    headers: {
      "Accept": "application/json; odata=verbose",
      "X-RequestDigest": requestDigest,
      "content-type": "application/json;odata=verbose",
      "If-Match": etag,
      "X-HTTP-METHOD": "PATCH"
    }
  });
}
```

29. Save your changes to **services.js**.
30. Open **controllers.js** in an editor window.
31. Replace the code which creates **homeController** to add the **deleteCustomer** function to the **\$scope** variable. You can either type the following code or copy and paste it from the **homeController.js.txt** file in the **Starter Files** folder.

```
app.controller('homeController',
function ($scope, wingtipCrmService) {
  wingtipCrmService.getCustomers().success(function (data) {
    $scope.customers = data.d.results;
    // add behavior function for view to call
    $scope.deleteCustomer = function (id) {
      wingtipCrmService.deleteCustomer(id).success(function (data) {
        $scope.$apply();
      });
    };
  });
});
};
```

32. Replace the code which creates the **newController** with the following code. If you'd rather not type all this code by hand, you can copy it from the **newController.js.txt** file in the **StarterFiles** folder for this lab.

```
app.controller('newController',
function ($scope, $location, wingtipCrmService) {

  $scope.customer = {};
  $scope.customer.FirstName = "";
  $scope.customer.Title = "";
  $scope.customer.WorkPhone = "";
  $scope.customer.HomePhone = "";
  $scope.customer.Email = "";

  $scope.addCustomer = function () {
    var firstName = $scope.customer.FirstName;
    var lastName = $scope.customer.Title;
    var workPhone = $scope.customer.WorkPhone;
    var homePhone = $scope.customer.HomePhone;
    var email = $scope.customer.Email;
    wingtipCrmService.addCustomer(firstName, lastName, workPhone, homePhone, email)
      .success(function (data) {
        $location.path("/");
      });
  };
});
};
```

33. Replace the code which creates the **viewController** with the following code.

```
app.controller('viewController',
function ($scope, $routeParams, wingtipCrmService) {
  var id = $routeParams.id;
  wingtipCrmService.getCustomer(id).success(function (data) {
    $scope.customer = data.d;
  });
});
};
```

34. Replace the code which creates the **editController** with the following code. If you'd rather not type all this code by hand, you can copy it from the **editController.js.txt** file in the **StarterFiles** folder for this lab.

```
app.controller('editController',
function ($scope, $routeParams, $location, wingtipCrmService) {
    var id = $routeParams.id;
    wingtipCrmService.getCustomer(id).success(function (data) {
        $scope.customer = data.d;

        $scope.updateCustomer = function () {
            var firstName = $scope.customer.FirstName;
            var lastName = $scope.customer.Title;
            var workPhone = $scope.customer.WorkPhone;
            var homePhone = $scope.customer.HomePhone;
            var email = $scope.customer.Email;
            var etag = $scope.customer.__metadata.etag;
            wingtipCrmService.updateCustomer(id, firstName, lastName, workPhone, homePhone, email, etag)
                .success(function (data) {
                    $location.path("/");
                });
        };
    });
});
};
```

35. Save your changes to **controllers.js**.
36. Open the view file in the **views** folder named **new.html** and replace the contents with the following HTML layout. If you'd rather not type all this code by hand, you can copy it from the **new.html.txt** file in the **StarterFiles** folder for this lab.

```
<h3>New Customer</h3>
<div class="form-horizontal">
    <fieldset>
        <div class="form-group">
            <label for="txtFirstName" class="col-lg-2 control-label">First Name:</label>
            <div class="col-lg-6">
                <input id="txtFirstName" type="text" class="form-control" ng-model="customer.FirstName">
            </div>
        </div>
        <div class="form-group">
            <label for="txtLastName" class="col-lg-2 control-label">Last Name:</label>
            <div class="col-lg-6">
                <input id="txtLastName" type="text" class="form-control" ng-model="customer.Title">
            </div>
        </div>
        <div class="form-group">
            <label for="txtWorkPhone" class="col-lg-2 control-label">Work Phone:</label>
            <div class="col-lg-6">
                <input id="txtWorkPhone" type="text" class="form-control" ng-model="customer.WorkPhone">
            </div>
        </div>
        <div class="form-group">
            <label for="txtHomePhone" class="col-lg-2 control-label">Home Phone:</label>
            <div class="col-lg-6">
                <input id="txtHomePhone" type="text" class="form-control" ng-model="customer.HomePhone">
            </div>
        </div>
        <div class="form-group">
            <label for="txtEmailAddress" class="col-lg-2 control-label">Email Address:</label>
            <div class="col-lg-6">
                <input id="txtEmailAddress" type="text" class="form-control" ng-model="customer.Email">
            </div>
        </div>
        <div class="form-group">
            <div class="col-lg-offset-2">
                <input id="cmdSave" type="button" class="button" value="Save" data-ng-
click="addCustomer()" />
            </div>
        </div>
    </div>
</div>
```

```
</fieldset>
</div>

<hr />

<a href="#">Return to customers list</a>
```

37. Save and close **new.html**.

38. Open the view file in the **views** folder named **view.html** and replace the contents with the following HTML layout. If you'd rather not type all this code by hand, you can copy it from the **view.html.txt** file in the **StarterFiles** folder for this lab.

```
<h3>View Customer</h3>

<div class="form-horizontal">
  <fieldset>
    <div class="form-group">
      <label for="txtID" class="col-lg-2 control-label">ID:</label>
      <div class="col-lg-6">
        <input id="txtID" type="text" readonly class="form-control"
          ng-model="customer.ID">
      </div>
    </div>
    <div class="form-group">
      <label for="txtFirstName" class="col-lg-2 control-label">First Name:</label>
      <div class="col-lg-6">
        <input id="txtFirstName" type="text" readonly class="form-control"
          ng-model="customer.FirstName">
      </div>
    </div>
    <div class="form-group">
      <label for="txtLastName" class="col-lg-2 control-label">Last Name:</label>
      <div class="col-lg-6">
        <input id="txtLastName" type="text" readonly class="form-control"
          ng-model="customer.Title">
      </div>
    </div>
    <div class="form-group">
      <label for="txtWorkPhone" class="col-lg-2 control-label">Work Phone:</label>
      <div class="col-lg-6">
        <input id="txtWorkPhone" type="text" readonly class="form-control"
          ng-model="customer.WorkPhone">
      </div>
    </div>
    <div class="form-group">
      <label for="txtHomePhone" class="col-lg-2 control-label">Home Phone:</label>
      <div class="col-lg-6">
        <input id="txtHomePhone" type="text" readonly class="form-control"
          ng-model="customer.HomePhone">
      </div>
    </div>
    <div class="form-group">
      <label for="txtEmailAddress" class="col-lg-2 control-label">Email Addresss:</label>
      <div class="col-lg-6">
        <input id="txtEmailAddress" type="text" readonly class="form-control"
          ng-model="customer.Email">
      </div>
    </div>
  </fieldset>
</div>

<hr />

<a href="#">Edit this Customer</a>
<br />
<a href="#">Return to Customers List</a>
```

39. Save and close **view.html**.

40. Open the view file in the **views** folder named **edit.html** and replace the contents with the following HTML layout. If you'd rather not type all this code by hand, you can copy it from the **edit.html.txt** file in the **StarterFiles** folder for this lab.

```
<h3>Edit Customer</h3>

<div class="form-horizontal">
  <fieldset>
    <div class="form-group">
      <label for="txtID" class="col-lg-2 control-label">ID:</label>
      <div class="col-lg-6">
        <input id="txtID" type="text" readonly class="form-control"
          ng-model="customer.ID">
      </div>
    </div>
    <div class="form-group">
      <label for="txtFirstName" class="col-lg-2 control-label">First Name:</label>
      <div class="col-lg-6">
        <input id="txtFirstName" type="text" class="form-control"
          ng-model="customer.FirstName">
      </div>
    </div>
    <div class="form-group">
      <label for="txtLastName" class="col-lg-2 control-label">Last Name:</label>
      <div class="col-lg-6">
        <input id="txtLastName" type="text" class="form-control"
          ng-model="customer.Title">
      </div>
    </div>
    <div class="form-group">
      <label for="txtWorkPhone" class="col-lg-2 control-label">Work Phone:</label>
      <div class="col-lg-6">
        <input id="txtWorkPhone" type="text" class="form-control"
          ng-model="customer.WorkPhone">
      </div>
    </div>
    <div class="form-group">
      <label for="txtHomePhone" class="col-lg-2 control-label">Home Phone:</label>
      <div class="col-lg-6">
        <input id="txtHomePhone" type="text" class="form-control"
          ng-model="customer.HomePhone">
      </div>
    </div>
    <div class="form-group">
      <label for="txtEmailAddress" class="col-lg-2 control-label">Email Addresss:</label>
      <div class="col-lg-6">
        <input id="txtEmailAddress" type="text" class="form-control"
          ng-model="customer.Email">
      </div>
    </div>
    <div class="form-group">
      <label for="txtID" class="col-lg-2 control-label">ETag:</label>
      <div class="col-lg-6">
        <input id="txtID" type="text" readonly class="form-control"
          ng-model="customer.__metadata.etag">
      </div>
    </div>
    <div class="form-group">
      <div class="col-lg-offset-2">
        <input id="cmdSave" type="button" class="button" value="Save"
          data-ng-click="updateCustomer()" />
      </div>
    </div>
  </fieldset>
</div>

<hr />

<a href="#">Return to customers list</a>
```

41. Save and close **edit.html**.

42. Test your work by pressing **{F5}** to begin a debugging session. The app should initialize the start page using the view defined in Home.html which should display a table of customers. However, now the app should support full CRUD functionality.
43. Click on the **Add Customer** link and make sure you are able to add a new customer item.
44. Test the links in the table in the home view. You should be able to click **View** and navigate to a view which displays the details of a single item.
45. You should be able to click **Edit** and navigate to a view which allows you to edit an existing item and save your changes.
46. You should be able to click **Delete** and delete a customer item from the list.

You have now completed this lab.