Developing Custom RESTful Services with Web API

Lab Time: 60 minutes

**Lab Folder**: C:\Student\Modules\11_WebAPI\Lab

Lab Overview: In this lab you will create an OData service using Web API. After investigating the service with Fiddler, you will enable Cross Origin Resource Sharing (CORS), and create a SharePoint-Hosted app to consume the service using the data.js JavaScript library.

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

Exercise 1: Create and Populate the Wingtip CRM Database in SQL Server

In this exercise you will create a SQL Server database with a Customers table for use in later exercises.

1. Launch Visual Studio 2015.

2. Drop down the **View** menu and select the **SQL Server Object Explorer** command.
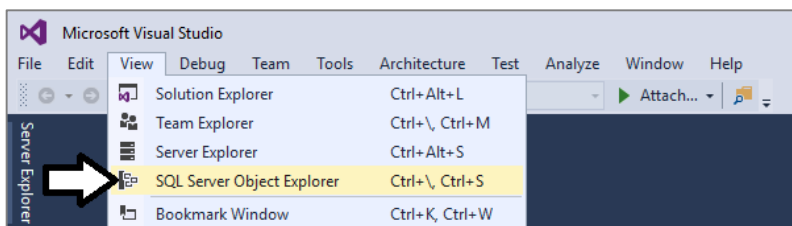
3. Wait a minute until the **SQL Server Object Explorer** pane has had time to initialize.

4. Expand the **SQL Server** node and locate the SQL Server instance named **(localdb)\MSSQLServerLocalDB** and then expand the **Database** node inside.
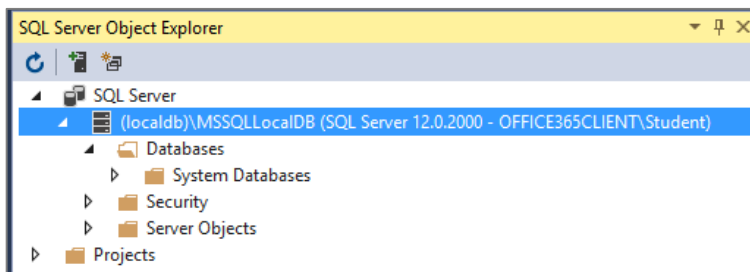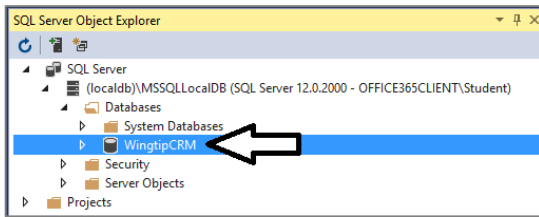

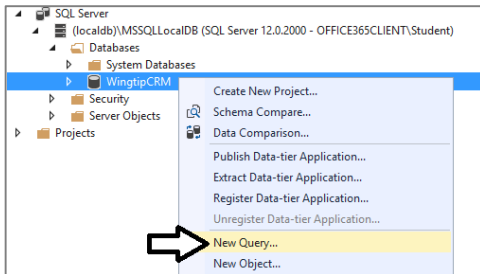5. Right-click on the **Databases** node and select he **Add New Database** command.
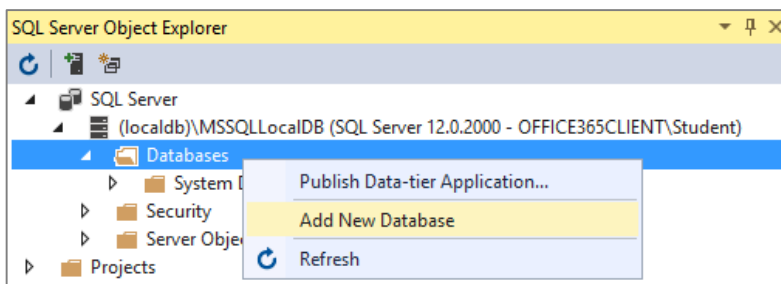
6. In the **Create Database** dialog, enter a **Database Name** of **WingtipCRM** and click **OK**.
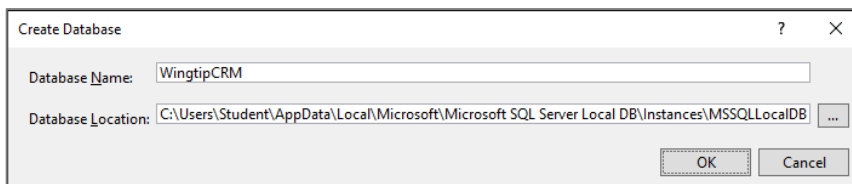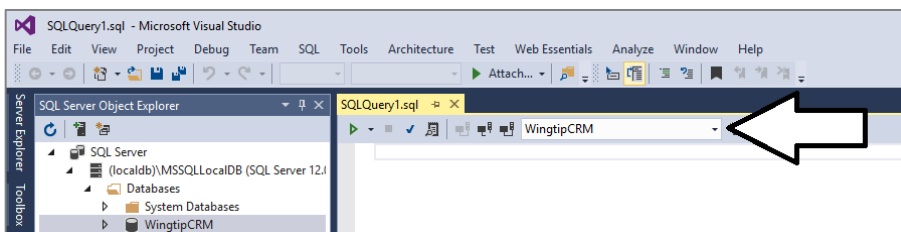
7. At this point you should see that a new database named **WingtipCRM** has been created.

8. Right-click on the **WingtipCRM** database and select the **New Query…** command.


9. You should now see a new window appear with the caption **SQLQuery1.sql**. Look at the drop down menu on this window and ensure that the currently selected database is **WingtipCRM**.

10. Using Windows Explorer, locate the file named **CreateCustomersTable.sql** at the following path.

```
C:\Student\Modules\11_WebAPI\Lab\StarterFiles\CreateCustomersTable.sql
```

11. Open the **CreateCustomersTable.sql** file using a utility such as Notepad.exe and copy its contents to the Windows clipboard.

12. Return to Visual Studio and paste the contents of **CreateCustomersTable.sql** into window for the **SQLQuery1.sql** file.

13. Execute this SQL script by clicking on the **Execute** button in the top right corner of the window.
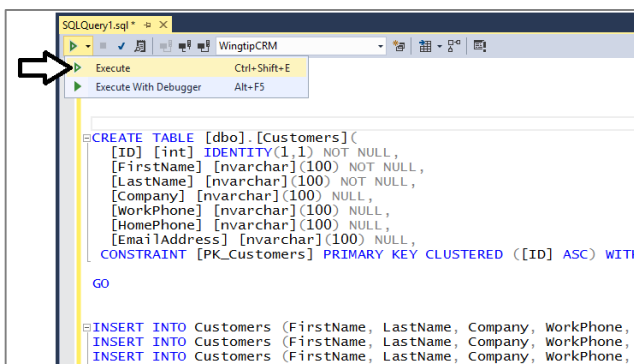
14. Refresh the view of the **WingtipCRM** database and verify that the **Customers** table has been created.
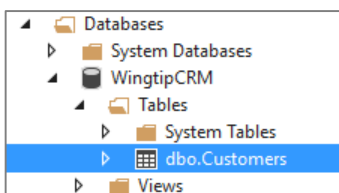
15. Right-click on the **Customers** table and select the **View Data** command.
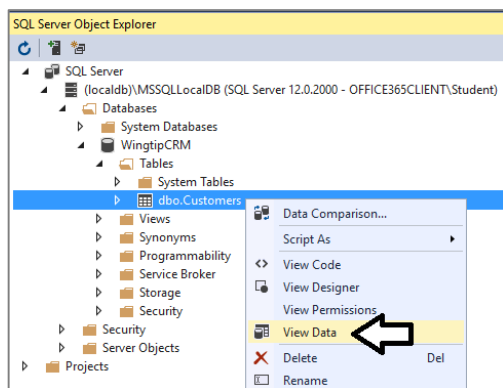
16. Verify that the **Customers** table contains a set of rows as see in the following screenshot.

17. Once you have verified the **Customers** table has been created with sample data, close all the open windows in Visual Studio.

You have now created the **WingtipCRM** database that you will use as a sample database for the remainder of this lab.

Exercise 2: Create an OData Service using Web API 2

In this exercise you create a new OData service using Web API scaffolding and Entity Framework 6.0.

1. Create the new solution in Visual Studio 2015:
   a) In Visual Studio select **File → New → Project**.
   b) In the **New Project** dialog:
      i) Select **Templates → Visual C# →Web**.
      ii) Click **ASP.NET Web Application**.
      iii) Name the new project **WingtipCRMService**
      iv) Set the Location to **C:\Student\Modules\11_WebAPI\Lab**
      v) Make sure to unselect the **Add Application Insights to project** checkbox on the right
      vi) Click **OK**.
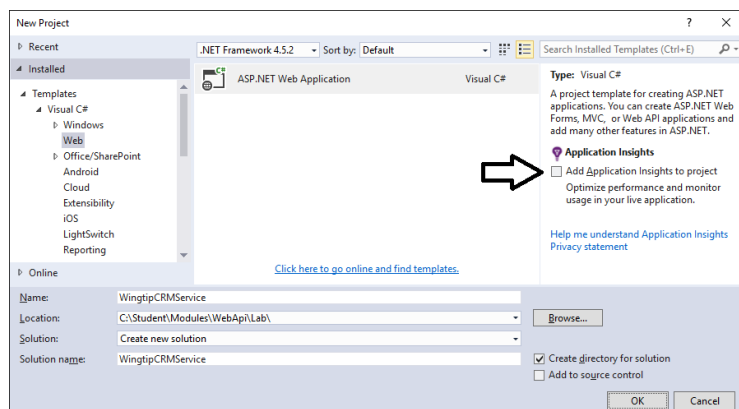
c) In the New ASP.NET Project dialog, select the **Web API** template and click the **Change Authentication** button on the right.
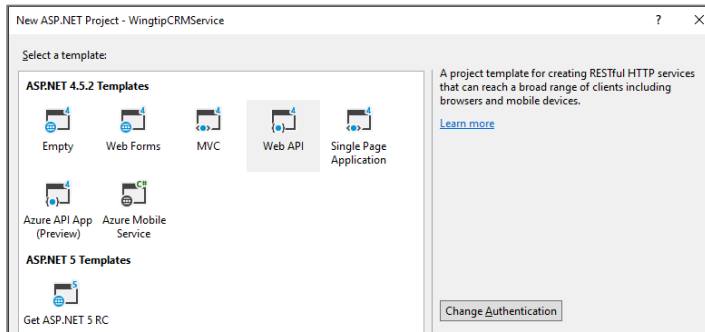
d) In the **Change Authentication** dialog, select **No Authentication** and click **OK**.
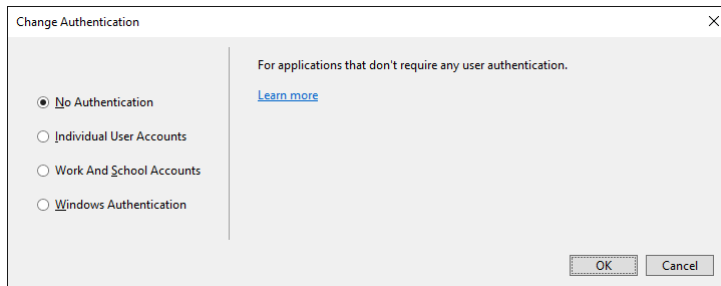
e) In the New ASP.NET Project dialog, verify that the Authentication setting is set to **No Authentication** and make sure the **Host in the cloud** checkbox is not selected.
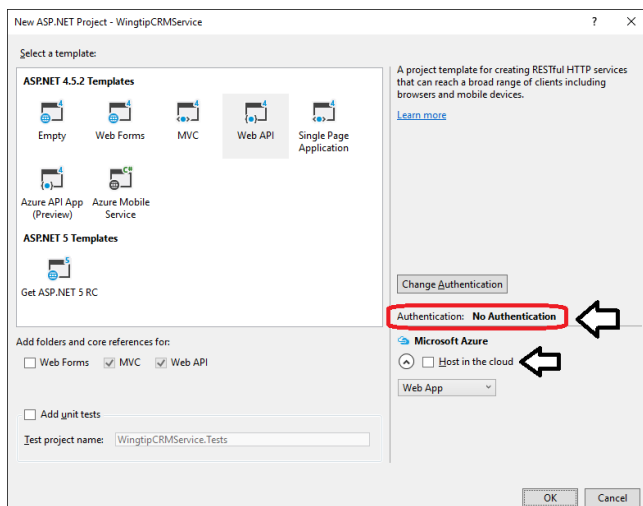
f) Click **OK**.to create the new project.
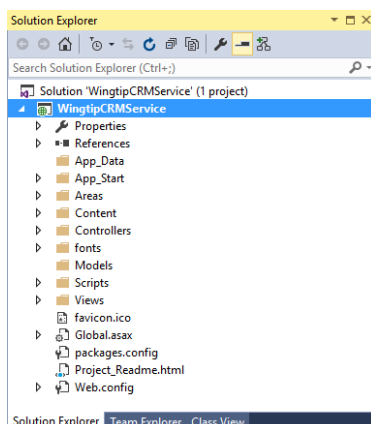
2. Add an Entity Framework Model
   a) In the **Solution Explorer**, right-click the **Models** folder.
   b) Select **Add→New Item** from the context menu.
   c) In the Add New item dialog:
      i) Click **Visual C#→Data**.
      ii) Select **ADO.NET Entity Data Model**
      iii) Name the model **WingtipCRM**.
      iv) Click **Add**.
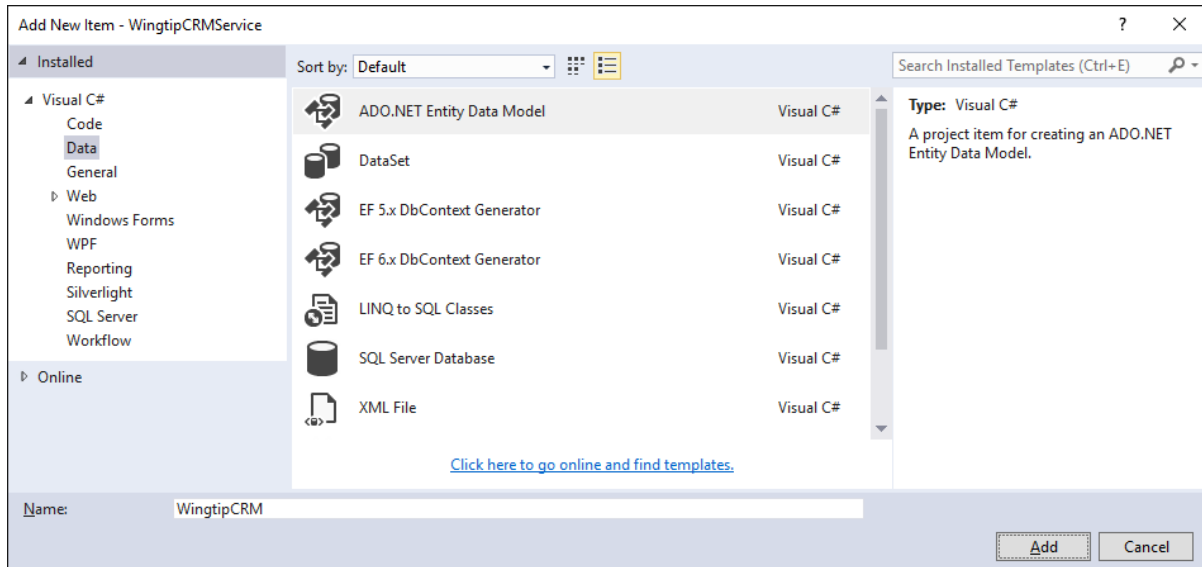
   d) In the first screen of the **Entity Data Model Wizard** select **EF Designer from Database** and click **Next**.
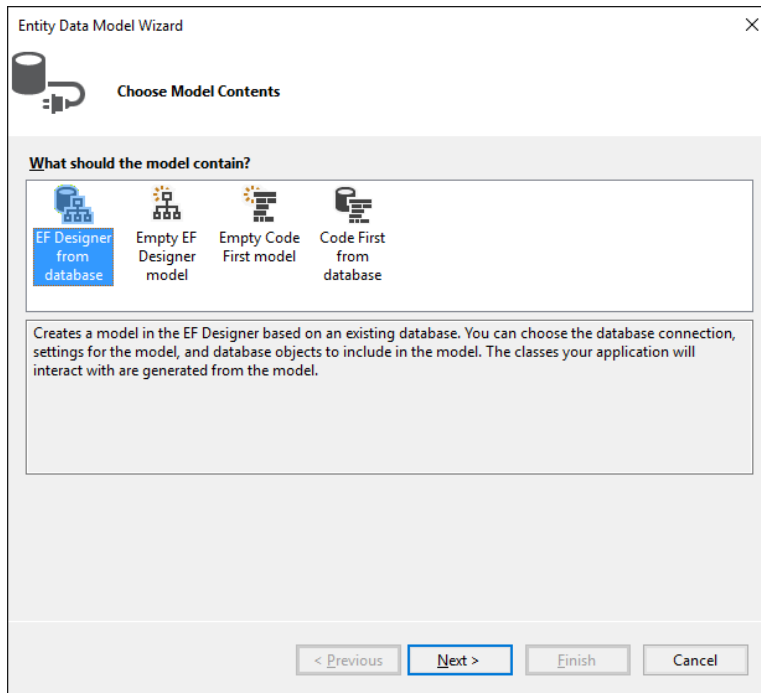
   e) In the second screen of the **Entity Data Model Wizard** click **New Connection**.
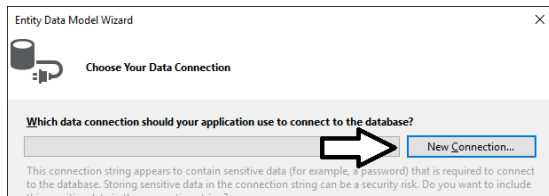
f) In the **Connection Properties** dialog, select **Microsoft SQL Server (SqlClient)** as the Data source and then enter a **Server name** value of **(localdb)\MSSQLLocalDB**. Next, move down to the **Connect to a database** section and select a database name of **WingtipCRM**. Once you have select the database, click OK to return to the **Entity Data Model Wizard** dialog.
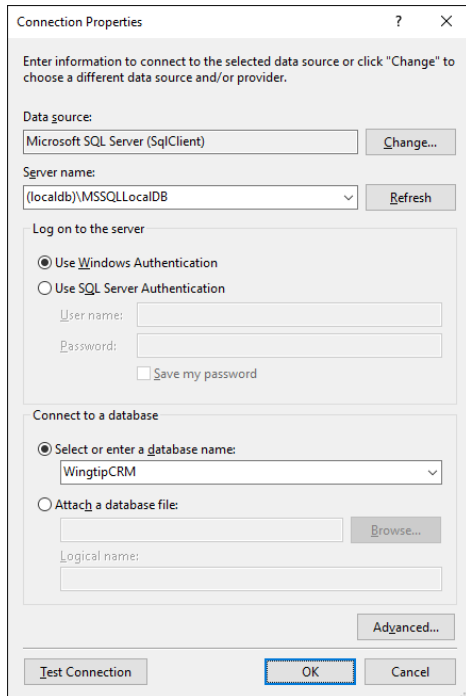
g) The **Entity Data Model Wizard** dialog should now show a connection screen like the one shown in the following screenshot. Also note that the wizard has generated the name **WingtipCRMEntities** to save connection settings for the data model in a connection string in the project's **web.config** file. Click Next to move to the next screen of the wizard.
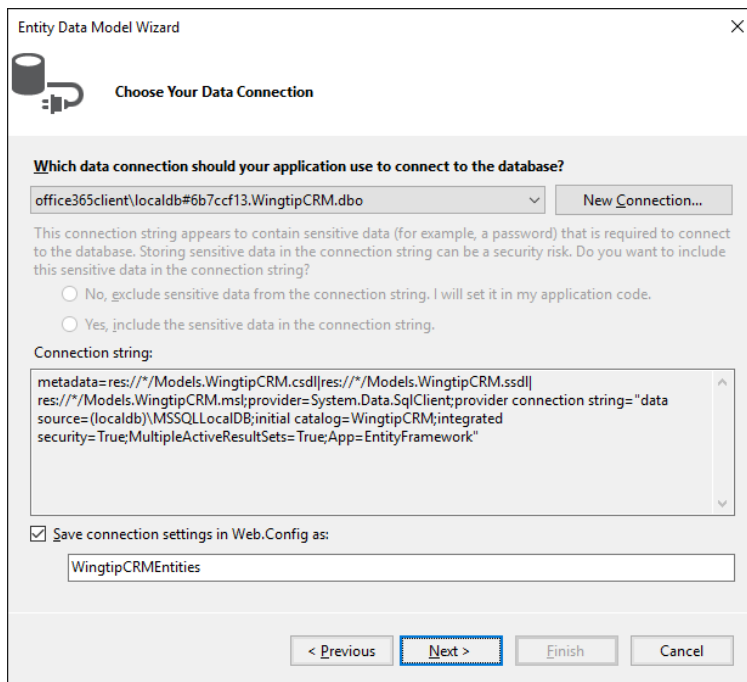
h) Keep the default selection of **Entity Framework 6.x** and click **Next**.
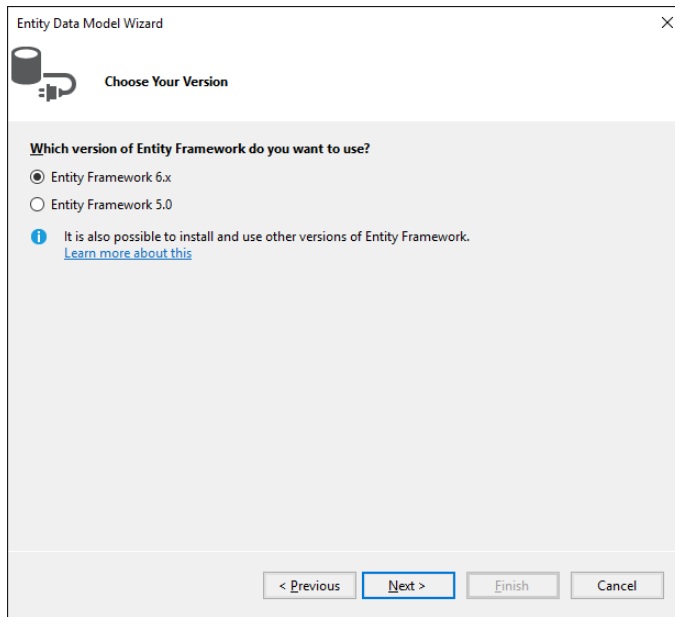
i) In the **Choose Your Database Objects and Settings** screen of the **Entity Data Model Wizard,** expand the **Tables** node and then select the **Customers** table as shown in the following screenshot. Note that the wizard has generated a **Model Namespace** value of **WingtipCRMEntities**. Click Finish to create the new Entity Framework data model.
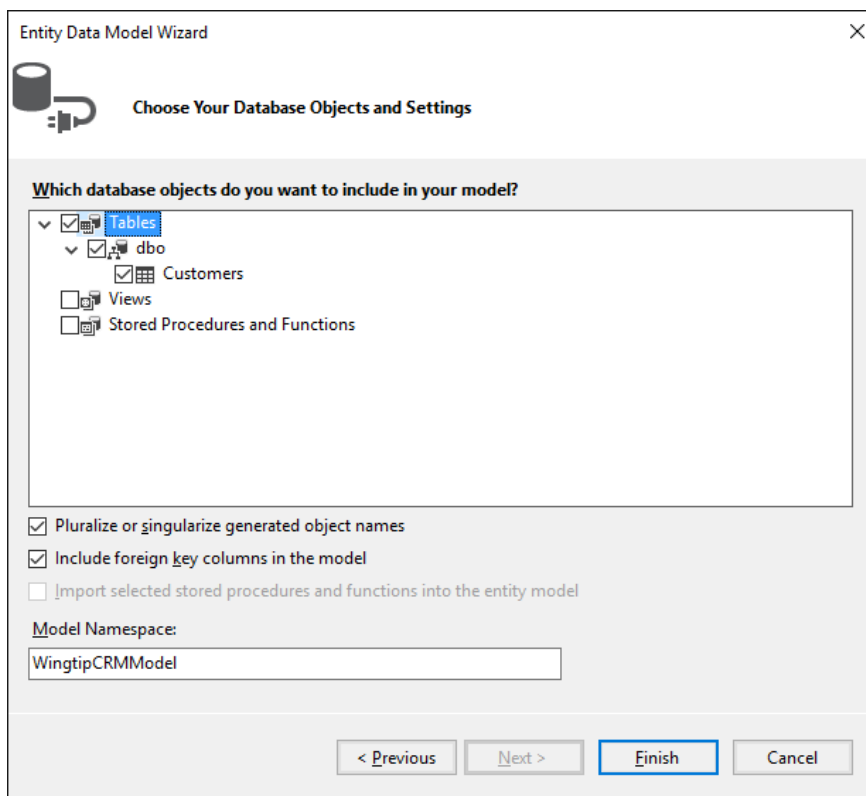
j) You might be prompted with the following **Security Warning** dialog. Click **OK** to continue.

Security Warning ? ✕

Running this text template can potentially harm your computer. Do not run it if you obtained it from an untrusted source.

Click OK to run the template.
Click Cancel to stop the process.

☑ Do not show this message again

OK   Cancel
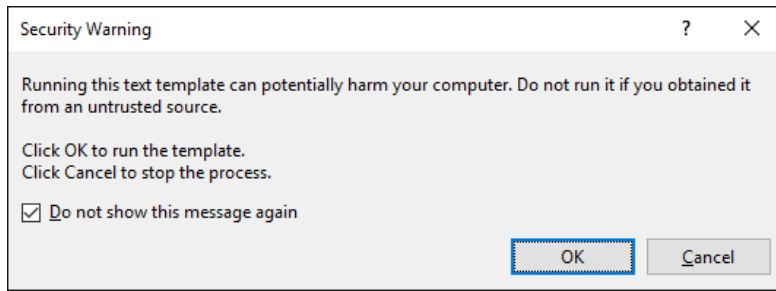
k) When the wizard completes it work, you should be able to see the new Entity Framework model as an EDMX file in a visual designer as shown in the following screenshot. Close this window.
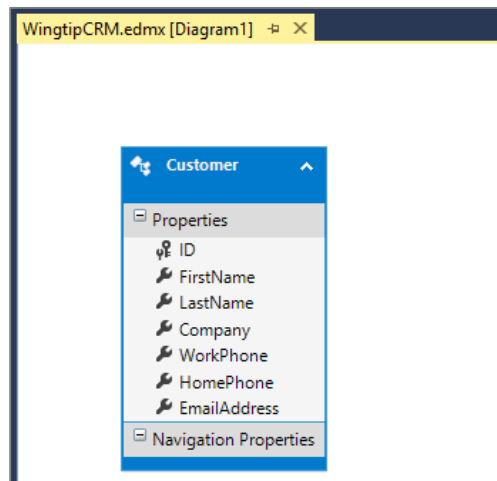
After adding an Entity Framework model, it's often recommended to build the project so that all the code for the model is generated. If you do not build the project, Visual Studio will not be able to properly create a strongly-type controller on top of the new Entity Framework model.

3. Build the project.
   a) Select **Build → Build Solution** from the main menu in Visual Studio
   b) Select **Window → Close All Documents** to clean up your workspace.
4. Add an OData Controller
   a) In the **Solution Explorer**, right click the **Controllers** folder.
   b) Select **Add→Controller** from the context menu.
   c) In the Add Scaffold dialog:
      i) Select **Web API 2 OData v3 Controller, with actions using Entity Framework.**
      ii) Click **Add**.
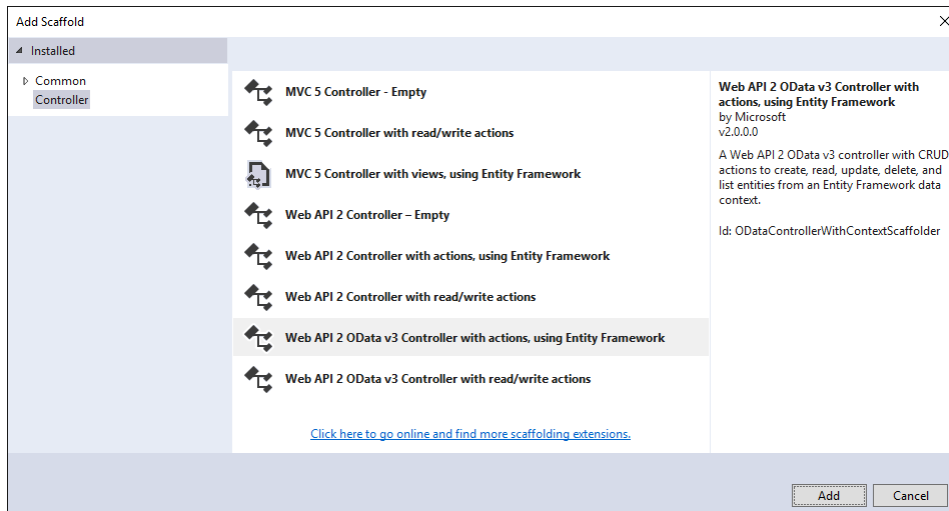
d) In the **Add Controller** dialog:

   i) Select **Customer (WingtipCRMService.Models)** in the Model Class drop-down list.

   ii) Select **WingtipCRMEntities (WingtipCRMService.Models)** in the Data Context Class drop-down list.

   iii) Accept the default **Controller name** of **CustomersController.**

   iv) Click **Add**.
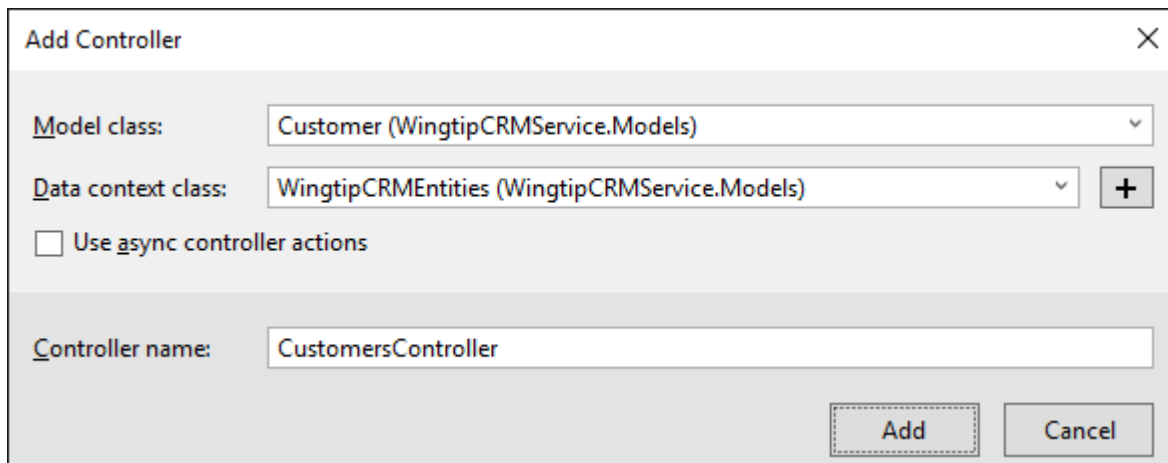
5. Once the Controller class has been created, you must now define the controller routes.

   a) Open the **CustomersController.cs** file that was just generated.

   b) Locate the following commented code and **Copy** it to the clipboard:

```
/*
The WebApiConfig class may require additional changes to add a route for this controller. Merge these
statements into the Register method of the WebApiConfig class as applicable. Note that OData URLs are
case sensitive.

using System.Web.Http.OData.Builder;
using System.Web.Http.OData.Extensions;
using WingtipCRMService.Models;
ODataConventionModelBuilder builder = new ODataConventionModelBuilder();
builder.EntitySet<Customer>("Customers");
config.Routes.MapODataServiceRoute("odata", "odata", builder.GetEdmModel());
*/
```

   c) Copy the following **using** statements to the Windows clipboard.

```
using System.Web.Http.OData.Builder;
using System.Web.Http.OData.Extensions;
using WingtipCRMService.Models;
```

   d) Expand the **App_Start** folder.

   e) Open the **WebApiConfig.cs** file.

f) **Paste** the using statements from the Windows clipboard under the existing using statements.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web.Http;

using System.Web.Http.OData.Builder;
using System.Web.Http.OData.Extensions;
using WingtipCRMService.Models;
```

g) Return to the **CustomerControllers.cs** file and copy the following code into the Windows clipboard.

```
ODataConventionModelBuilder builder = new ODataConventionModelBuilder();
builder.EntitySet<Customer>("Customers");
config.Routes.MapODataServiceRoute("odata", "odata", builder.GetEdmModel());
```

h) Return to the **WebApiConfig.cs** file and paste the code at the top of the **Register** method just below the comment that reads **//Web API configuration and services**.

```
public static void Register(HttpConfiguration config) {
    // Web API configuration and services
    ODataConventionModelBuilder builder = new ODataConventionModelBuilder();
    builder.EntitySet<Customer>("Customers");
    config.Routes.MapODataServiceRoute("odata", "odata", builder.GetEdmModel());

    // Web API routes
    config.MapHttpAttributeRoutes();

    config.Routes.MapHttpRoute(
        name: "DefaultApi",
        routeTemplate: "api/{controller}/{id}",
        defaults: new { id = RouteParameter.Optional }
    );
}
```

The routes for the OData controller are created by the code in the **Register** method. For an OData controller like the one you have just created, the **ODataConventionModelBuilder** class will build out routes that have an "odata" prefix and a "Customers" endpoint. For example, the URL to access this OData service will be **http://localhost:60982/odata/Customers**. The order in which routes are defined is important. If routes conflict, the first definition will be used.

i) **Add** the line of code **builder.EntitySet<Customer>("Customers")** to include namespace information in the OData model:

```
ODataConventionModelBuilder builder = new ODataConventionModelBuilder();
builder.EntitySet<Customer>("Customers");
builder.Namespace = "WingtipCRMService.Models";
config.Routes.MapODataServiceRoute("odata", "odata", builder.GetEdmModel());
```

The ODataConventionModelBuilder class will also build a schema that supports the $metadata endpoint. The class normally generates a schema without namespace information. So, you must manually add the namespace, which is normally the same as the namespace used in the Entity Framework model.

j) At this point, the **WebApiConfig.cs** file in your project should match the following code listing.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web.Http;

using System.Web.Http.OData.Builder;
using System.Web.Http.OData.Extensions;
using WingtipCRMService.Models;


namespace WingtipCRMService {
  public static class WebApiConfig {
    public static void Register(HttpConfiguration config) {
      // Web API configuration and services
      ODataConventionModelBuilder builder = new ODataConventionModelBuilder();
      builder.EntitySet<Customer>("Customers");
      builder.Namespace = "WingtipCRMService.Models";
      config.Routes.MapODataServiceRoute("odata", "odata", builder.GetEdmModel());

      // Web API routes
      config.MapHttpAttributeRoutes();
```

```
        config.Routes.MapHttpRoute(
            name: "DefaultApi",
            routeTemplate: "api/{controller}/{id}",
            defaults: new { id = RouteParameter.Optional }
        );
    }
  }
}
```

6. Build the project.
    a) Select **Build→Rebuild Solution** from the main menu in Visual Studio
    b) Select **Window→Close All Documents** to clean up your workspace.

Exercise 3: Testing the OData Service with Fiddler

In this exercise, you will use Fiddler to manually test the service.

1. Start the service for testing.
    a) Launch **Fiddler2**. (press your **Windows** key and then type "**fiddler**" and click on the **Fiddler2** tile).
    b) In Visual Studio 2015, open **CustomersController.cs**
        i) Place a breakpoint in the following methods: **GetCustomers**, **GetCustomer**, **Put**, **Post**, **Patch**, and **Delete**.
    c) In Visual Studio 2015, using the main menu, select **Debug→Start Debugging**.
        i) If prompted, log in with your Windows credentials.
        ii) Verify that the default page of the project appears in the browser.

The default Web API project includes a visual home page that you can use to provide information about the service. Remember that it's easy to mix MVC controllers with Web API controllers in the same project.

2. Testing methods from Fiddler
    a) In the browser, note of the root URI for the service. This should reference the localhost along with a port number (e.g., http://localhost:60982).
       Be certain to write your port number down you will need this later
    b) In Fiddler, select **Edit→Remove→All Sessions** to clear the display.
    c) In the **Composer** tab:
        i) Click the **Options** tab.
        ii) Check **Automatically Authenticate**.
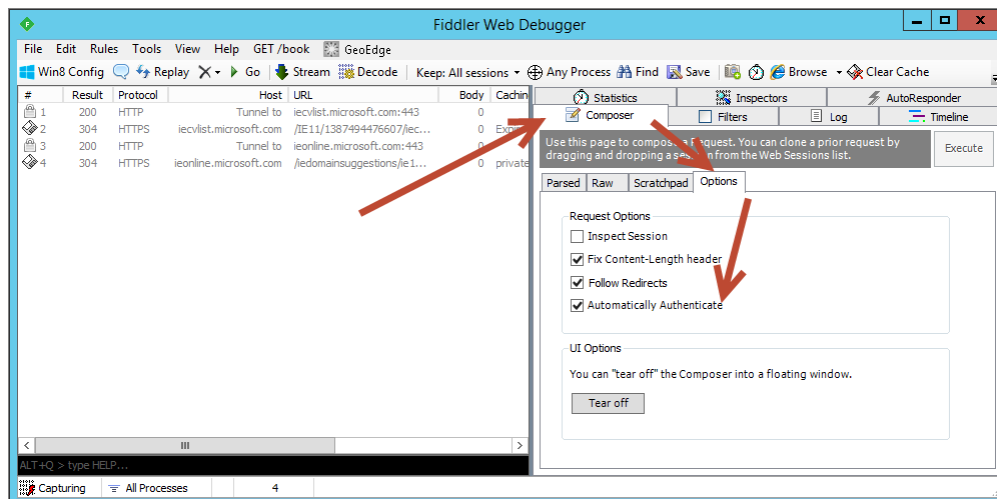
The option to "Automatically Authenticate" allows Fiddler to respond to Windows Authentication challenges from the OData service.

        i) On the **Composer** tab, click the **Parsed** tab.
        ii) Enter the root URI plus **/odata/$metadata** (e.g., http://localhost:60982/odata/$metadata).
        iii) Click **Execute**.
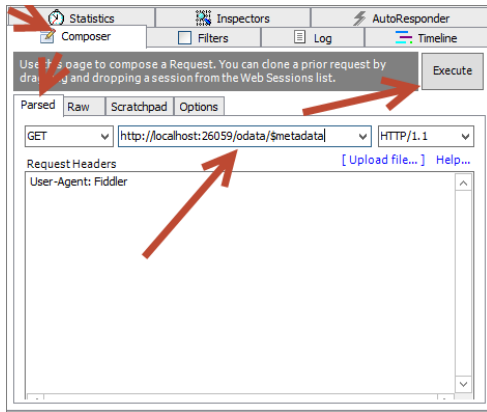
The Composer tab allows you to manually construct HTTP requests that Fiddler will send to the OData service.

d) After the service responds, double click the last session line in Fiddler, which represents the response from the server.

e) In the **Inspectors** tab:

    i) Click the **Raw** tab in the upper half.

    ii) Click the **Raw** tab in the lower half.
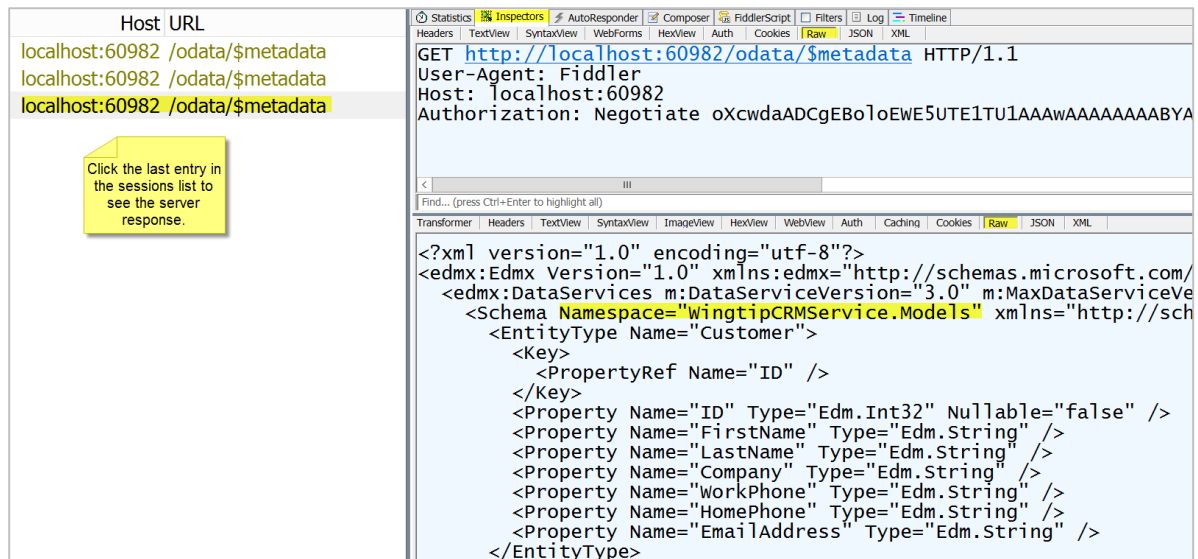
    iii) View the schema results and note the namespace you defined in the OData service.

The Inspectors tab allows you to see the request in the top-half and the response in the lower-half. Different renderings of the data (e.g., "Headers", "Raw", "XML", "JSON") allow you to see different parts of the request and response formatted appropriately.

f) In the **Composer** tab:

    i) Click the **Parsed** tab.

    ii) Enter the root URI plus **/odata/Customers** (e.g., http://localhost:60982/odata/Customers).

    iii) Click **Execute**.

    iv) On the breakpoint, note the method that was called and then press **F5** to continue.

g) After the service responds, double click the last session line in Fiddler, which represents the response from the server.

h) In the **Inspectors** tab:

    i) Click the **Headers** tab in the upper half.

    ii) Click the **JSON** tab in the lower half.

    iii) View the JSON results from the initial call.

```
Transformer | Headers | TextView | SyntaxView | ImageView | HexView | WebView | Auth | Caching | Cookies | Raw | JSON | XML
□ JSON
    odata.metadata=http://localhost:60982/odata/$metadata#Customers
  □ value
    □ {}
        Company=Benthic Petroleum
        EmailAddress=Quincy.Nelson@BenthicPetroleum.com
        FirstName=Quincy
        HomePhone=1(340)517-3737
        ID=1
        LastName=Nelson
        WorkPhone=1(340)608-7748
    □ {}
```

i) In the **Composer** tab:

    i) Change the URI to **/odata/Customers(2)** (e.g., http://localhost:60982/odata/Customers(2)).

    ii) Click **Execute**.
        Note: if you receive a 404 error… pick another number and try again (i.e. you may have deleted that record in a prior lab).

    iii) On the breakpoint, note the method that was called and then press **F5** to continue.

j) After the service responds, click the last session line in Fiddler, which represents the response from the server.

k) In the **Inspectors** tab:

    i) Click the **Headers** tab in the upper half.

    ii) Click the **JSON** tab in the lower half and verify a single record was returned.


```
Transformer | Headers | TextView | SyntaxView | ImageView | HexView | WebView | Auth | Caching | Cookies | Raw | JSON | XML
□ JSON
    Company=Cyberdine Systems
    EmailAddress=jude.mason@cyberdinesystems.com
    FirstName=Jude
    HomePhone=1(203)411-0071
    ID=2
    LastName=Mason
    odata.metadata=http://localhost:60982/odata/$metadata#Customers/@Element
    WorkPhone=1(203)408-0466
```
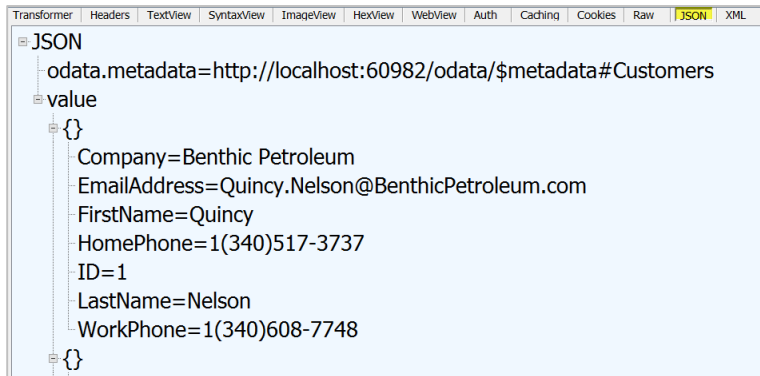
l) In the **Composer** tab:

    i) Change the HTTP verb to **PUT**.

    ii) Add a new request header **Content-Type:application/json**.
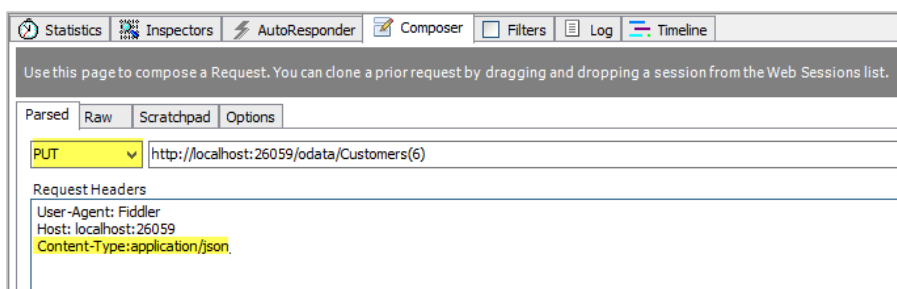
The Content-Type header specifies the form of the request data you will send to the server.

    iii) Add the following request body (in Fiddler2 **Request Body** is the next text box below the Request Headers) to provide updated information:
       Note: if ID 2 did not work for you above (in step 2.i) please pick the same number that you used above

```
{
  "ID":2,
  "Company":"Critical Path Training",
  "EmailAddress":"jude.mason@criticalpathtraining.com",
  "FirstName":"Jude",
  "HomePhone":"1(203)411-0071",
  "LastName":"Mason",
  "WorkPhone":"1(203)408-0466"
```

```
        }
```

    iv) Click **Execute**.

    v) On the breakpoint, hit **F5** to continue.

    vi) Back on the Composers tab, change the HTTP verb from PUT to **GET**.

    vii) Delete the **Request Body** text.

    viii)Delete the **Content-Type:application/json** Request Header.

    ix) Click **Execute**.

    x) On the breakpoint, hit **F5** to continue.

m) After the service responds, click the last session line in Fiddler, which represents the response from the server.

n) In the **Inspectors** tab:

    i)   Click the **Headers** tab in the upper half.
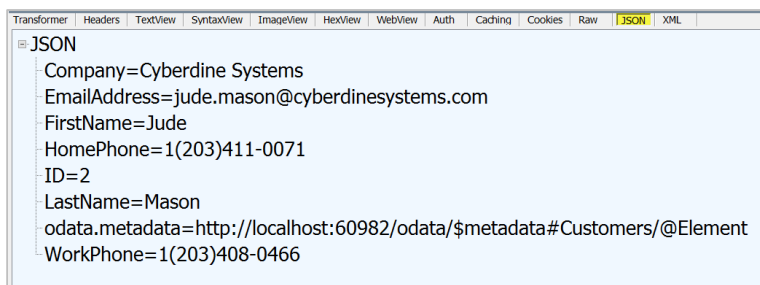
    ii) Click the **JSON** tab in the lower half and verify the data changes were made.

Transformer	Headers	TextView	SyntaxView	ImageView	HexView	WebView	Auth	Caching	Cookies	Raw	JSON	XML

```
JSON
  Company=Critical Path Training
  EmailAddress=jude.mason@criticalpathtraining.com
  FirstName=Jude
  HomePhone=1(203)411-0071
  ID=2
  LastName=Mason
  odata.metadata=http://localhost:60982/odata/$metadata#Customers/@Element
  WorkPhone=1(203)408-0466
```
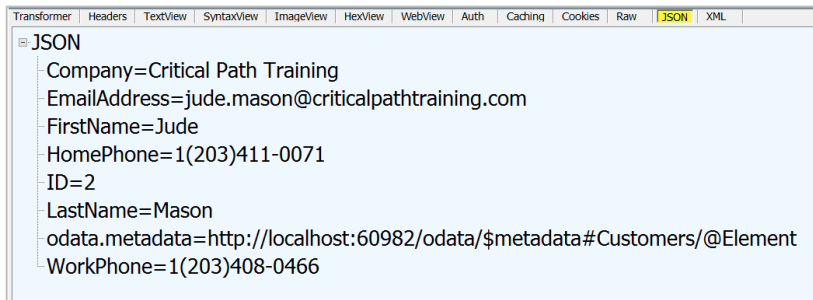
o) **Close** the browser to stop debugging.

p) **Close** Fiddler.

Exercise 5: Enable CORS Support in the WingtipCRM Service

In this exercise, you will enable Cross Origin Resource Sharing (CORS) for the OData service and create a SharePoint-Hosted app to consume the service.

1. Add the NuGet package to enable CORS support in a Web API 2 web service.

    a) In the **Solution Explorer**, right click the **WingtipCRM** project node.

    b) Select **Manage NuGet Packages** from the context menu.

    c) Type **CORS** in the search box and locate the **Microsoft.AspNet.WebApi.Cors** package and click Install.
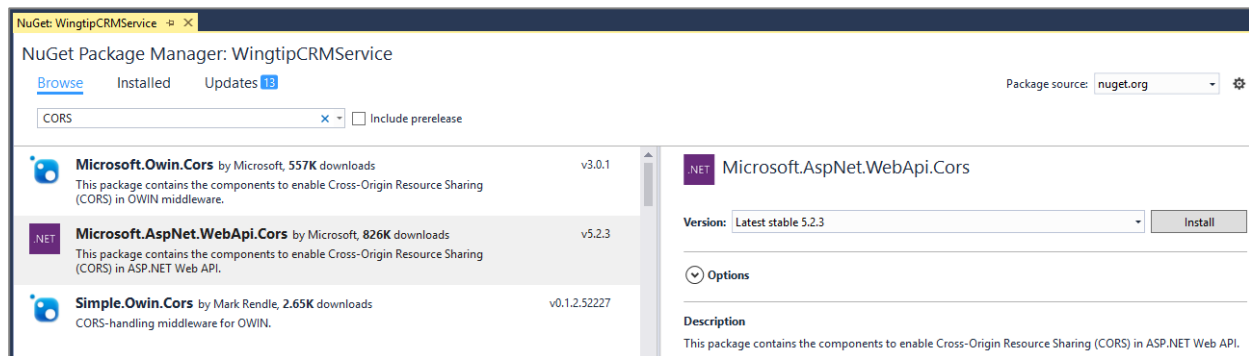
    d) Click **OK** in the next dialog.
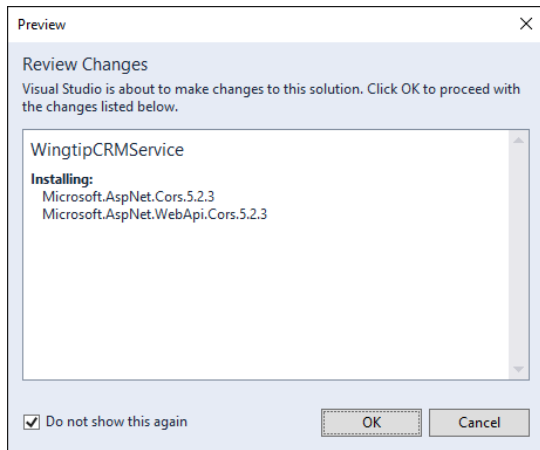
 e) **Close** the NuGet Package Manager after the installation is complete.

2. Add code to the project to enable CORS support.

 a) Expand the **App_Start** folder.

 b) Open the file **WebApiConfig.cs**.

 c) Add the line of code config.EnableCors(); at the bottom of the **Register** method to enable CORS for the entire service:

```
public static void Register(HttpConfiguration config) {
  // Web API configuration and services
  ODataConventionModelBuilder builder = new ODataConventionModelBuilder();
  builder.EntitySet<Customer>("Customers");
  builder.Namespace = "WingtipCRMService.Models";
  config.Routes.MapODataServiceRoute("odata", "odata", builder.GetEdmModel());

  // Web API routes
  config.MapHttpAttributeRoutes();

  config.Routes.MapHttpRoute(
      name: "DefaultApi",
      routeTemplate: "api/{controller}/{id}",
      defaults: new { id = RouteParameter.Optional }
  );

  config.EnableCors();
}
```

 d) Expand the **Controllers** folder.

 e) Open the file **CustomersController.cs**.

 f) Add the following using statement.

```
using System.Web.Http.Cors;
```

 g) Add the following attribute above the class definition (i.e. just above public class **CustomersController : ODataController**).

```
[EnableCors("*","*","*","DataServiceVersion,MaxDataServiceVersion")]
```

You may enable CORS for specific origins, methods, or custom "author" headers. You may also specify what additional headers to return. In the code above, all origins, methods, and headers are accepted. Additionally, the service returns the **DataServiceVersion** and **MaxDataServiceVersion** headers, which are expected by the Chrome browser. In a production application, you would always limit the accepted origins to those that you trust.

3. Add the client app to your solution

 a) In the Starter Files folder for this lab (C:/Student/Modules/WebAPI/StarterFiles), locate the folder **WingtipCRMClient**.

 b) Copy this entire folder into the working directory you are using for this lab (i.e. C:/Student/Modules/WebAPI/WingtipCRMService/).

 c) In Visual Studio 2015, select **File→Add→Exiting Project**.

 d) Browse to the Project folder you just copied (C:\Student\Modules\WebAPI\WingtipCRMService\WingtipCRMClient)

 e) Add the **WingtipCRMClient** project to your solution.

  i) Select the **WingtipCRMClient.csproj** file and click **Open**

Other labs in the class will deal explicitly with creating JavaScript-based apps and binding data to web pages. For this lab, you will focus on the operations necessary to read and write to the OData service.

f)   In the Solution explorer, expand the **scripts** folder and note the **datajs-1.1.2** library.

Datajs is a library specifically intended to be used with OData services. It supports data caching to make it easy to handle large data sets efficiently.

g)   In the scripts folder, open the **wingtip.crm.viewmodel.js** library for editing.

The wingtip.crm.viewmodel library is used to wrap the datajs operations and generate a view model that can be bound to the web page. In this exercise, you'll edit the parts of the library that deal with datajs.

h)   Locate the comment **//TO DO: Update the Service Root** and edit the service root to point to your OData endpoint.

Note: this is the port number you were asked to write down in the earlier exercise.

Warning: Be sure that the service root ends with a forward slash.

a)   Locate the comment **//TO DO: Edit refreshCache function** and add the following code as shown below:

```
var refreshCache = function () {
    //TO DO: Edit refreshCache function
    if (cache !== null) cache.clear();
    cache = datajs.createDataCache({
      name: "contacts",
      source: serviceRoot + "odata/Customers?$orderby=LastName,FirstName&$top=100",
      prefetchSize: 100,
      pageSize: 100
    });
    loadCustomers();

};
```

The refreshCache function loads the cache with data from the OData service. Read operations for the app can then be performed against the cache instead of the OData service.

a)   Locate the comment **//TO DO: Edit loadCustomers function** and add the following code as shown below:

```
var loadCustomers = function () {
    //TO DO: Edit loadCustomers function
    return cache.readRange(0, 100).then(
        function (data) {
            onLoadCustomersComplete(data);
        },
        function (err) {
            onError(err);
        }
    );
};
```

The loadCustomers function reads all of the data from the cache. Once read, it calls the onLoadCustomersComplete function, which will bind the data to a table for viewing.

b)   Locate the comment **//TO DO: Edit getCustomer function** and add the following code as shown below:

```
var getCustomer = function (Id) {
    //TO DO: Edit getCustomer function
    var deferred = jQuery.Deferred();
    cache.readRange(0, 100).always(function (data) {
        deferred.resolve(
            _.chain(data)
            .filter(function (contact) { return (contact.ID === Id); })
            .rest(0)
            .first()
            .value()
        );
    });
    return deferred.promise();
}
```

The getCustomer function returns a single customer record from the cache. This is accomplished using an additional third-party JavaScript library named underscore.js. Underscore provides many useful utility functions for manipulating data sets.

a)   Locate the comment **//TO DO: Edit addCustomer function** and add the following code as shown below:

```javascript
var addCustomer = function (FirstName, LastName, Company, EmailAddress, WorkPhone, HomePhone) {
    //TO DO: Edit addCustomer function
    OData.request(
        {
            requestUri: serviceRoot + "odata/Customers",
            method: "POST",
            contentType: "application/json",
            data: {
                'LastName': LastName,
                'FirstName': FirstName,
                'Company': Company,
                'EmailAddress': EmailAddress,
                'WorkPhone': WorkPhone,
                'HomePhone': HomePhone
            },
            headers: {
                "accept": "application/json"
            }
        }, function (data) {
            refreshCache();
        }, function (err) {
            refreshCache();
        }
    );
};
```

The addCustomer function makes a POST to the OData service to add a new customer then calls refreshCache to display the data.

a) Locate the comment **//TO DO: Edit updateCustomer function** and add the following code as shown below:

```javascript
var updateCustomer = function (Id, FirstName, LastName, Company, EmailAddress, WorkPhone, HomePhone) {
    //TO DO: Edit updateCustomer function
    OData.request(
        {
            requestUri: serviceRoot + "odata/Customers(" + Id + ")",
            method: "PUT",
            contentType: "application/json",
            data: {
                'ID': Id,
                'LastName': LastName,
                'FirstName': FirstName,
                'Company': Company,
                'EmailAddress': EmailAddress,
                'WorkPhone': WorkPhone,
                'HomePhone': HomePhone
            },
            headers: {
                "accept": "application/json"
            }
        }, function (data) {
            refreshCache();
        }, function (err) {
            refreshCache();
        }
    );
};
```

b) Locate the comment **//TO DO: Edit deleteCustomer function** and add the following code as shown below:

```javascript
var deleteCustomer = function (Id) {
    //TO DO: Edit deleteCustomer function
    OData.request(
        {
            requestUri: serviceRoot + "odata/Customers(" + Id + ")",
            method: "DELETE",
            headers: {
                "accept": "application/json"
            }
        }, function (data) {
            refreshCache();
        }, function (err) {
            refreshCache();
        }
    );
};
```

4. Test the solution

a) In **Visual Studio Solution Explorer** select **WingtipCRMClient** and in the **Properties** window below set the **Start Action** to **Google Chrome**.

b) Now, in Visual Studio, click on your **CustomersController.cs** tab (or if needed re-open this file from the WingtipCRMService project Controllers folder). (Note: we must be inside a file that renders in a web browser for this next feature of Visual Studio to work correctly)

c) In Visual Studio, change the browser that will run the application to Google Chrome; using the Run drop-down arrow as show below, select Google Chrome as your browser choice.
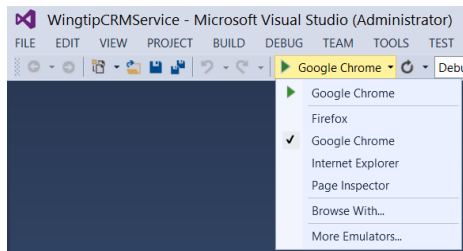
d) Press **F5** to start debugging the solution.

e) If prompted, log in using your Windows credentials.

f) After the OData service starts, launch **Fiddler**.

   i) Press your **windows** key and type "**fiddler**" then click on the Fiddler2 tile to start Fiddler.

g) In Visual Studio 2015, return to your **Solution Explorer** tab (Note: you may need to click on the Solution Explorer tab on the right hand side of the page) right click the WingtipCRMClient project node and select **Debug→Start New Instance** to run the associated app.
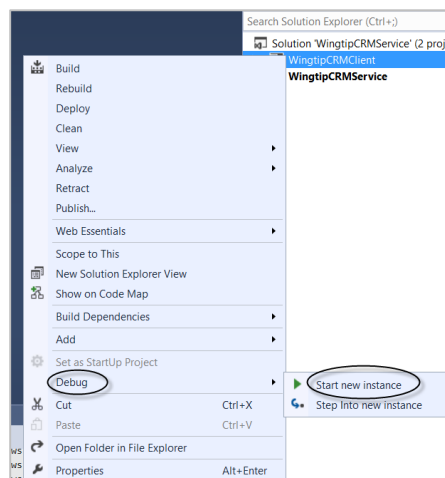
h) After the app starts, (This may take a while… be patient until a new tab opens in Chrome)

i) If you are prompted by Microsoft Office for permission to run at the top of this new tab in Chrome click **Always run on this site**.

j) Back in Fiddler, using the main menu, select **Edit→Remove→All Sessions** to clear the display.

k) Back in Chrome, Edit a record in the **Wingtip CRM Client** grid, by clicking on the Edit button on the far right hand side of the row, and edit part of the information as shown below:

l) After saving the changes, by clicking the **Update** button, examine the PUT operation in Fiddler by clicking on the correct operation in the results area on the left side of the application and note the Cross Origin Resource Headers that were sent between the client and server.

(Note: the operation you are looking for is the one that is the PUT operation; that is, the first **/odata/Customers(1)** operation with a Result code of 200 as shown in the sample image below)

When this is opened in Fiddler you should see text similar to the text below:

5. Feel free to use this same process to try and investigate other operations on the data (Add a new Customer or Delete a Customer (be sure to just delete the customer you created in the Add step to keep the Dataset in good shape))

6. When you are finished, you should close Chrome and Visual Studio 2013.

Congratulations! You have just created an OData service using Web API. You have learned how to use Fiddler to investigate the inner workings of this API. Next you successfully enabled Cross Origin Resource Sharing (CORS), and then created and tested, using Fiddler) a SharePoint-Hosted app to consume the service using the data.js JavaScript library.