Developing Provider-hosted Add-ins using MVC

Lab Time: 60 minutes

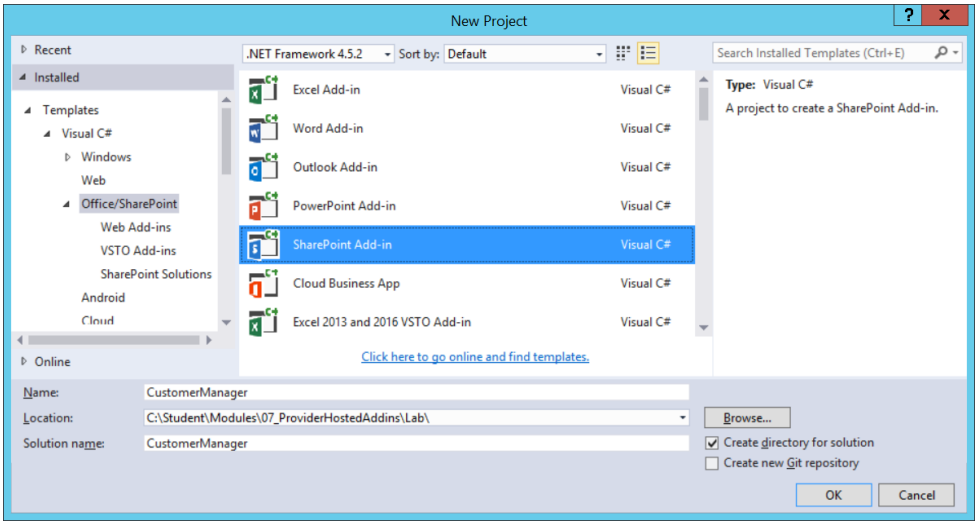
Lab Folder: C:\Student\Modules\07_ProviderHostedAddins\Lab

Lab Overview: In this lab you will create a provider-hosted add-in for SharePoint Online using ASP.NET and the MVC framework. You will work with MVC controllers, views and models to implement a user interface experience as well as a scheme for managing SharePoint session state across requests. You will also use Entity Framework to create a custom database for tracking customer data. After creating the customer database, you will use the MVC scaffolding support in Visual Studio to create a strongly-typed controller class that extends your add-in with a user interface experience for viewing, creating, updating and deleting customer records.

Exercise 1: Creating a Provider-Hosted Add-in that uses the MVC Framework

In this exercise you will use Visual Studio to create a new SharePoint Add-in project that is implemented as a provider-hosted add-in which uses the MVC framework in ASP.NET. Once you have created the new project for the add-in, you will make some basic changes to the app manifest and to MVC components user to generate the user interface. After that, you get your add-in project up and running in the Visual Studio debugger so you can test it out.

1. Before you can start working on this lab, you must ensure the S2S certificate configuration on your VM is setup correctly by determining whether you have already completed **Exercise 3: Create a Test Certificate for Working with S2S Trusts Authentication** in the earlier lab titled **Lab06: Configuring App Permissions and S2S Authentication**.
   a) If you are not sure whether you have completed this prerequisite exercise, you can examine the **C:\Certs** folder on your student VM to see if there are two certificate files named **WingtipAppCetificate01.cer** and **WingtipAppCetificate01.pfx**. If these files do not exist in the **C:\Certs** folder, you have not yet completed Exercise 3.
   b) If you have already completed **Exercise 3: Create a Test Certificate for Working with S2S Trusts Authentication** from lab 6, you can now move ahead to Exercise 2 in this lab.
   c) If you have *NOT* already completed **Exercise 3: Create a Test Certificate for Working with S2S Trusts Authentication** from lab 6 titled **Configuring App Permissions and S2S Authentication**, you must go back and complete that exercise now. After that you can then proceed with Exercise 2 in this lab.
2. Launch **Visual Studio**.
3. Create the new solution in Visual Studio:
   a) In Visual Studio select **File → New → Project**.
   b) In the **New Project** dialog:
      i) Select **Templates → Visual C# → Office/SharePoint → Apps**.
      ii) Click **App for SharePoint**
      iii) Name the new project **CustomerManager**.
      iv) Add the new project into the folder at **C:\Student\Modules\07_ProviderHostedAddins\Lab**.
      v) Click **OK** to display the **New Add-in for SharePoint** wizard.

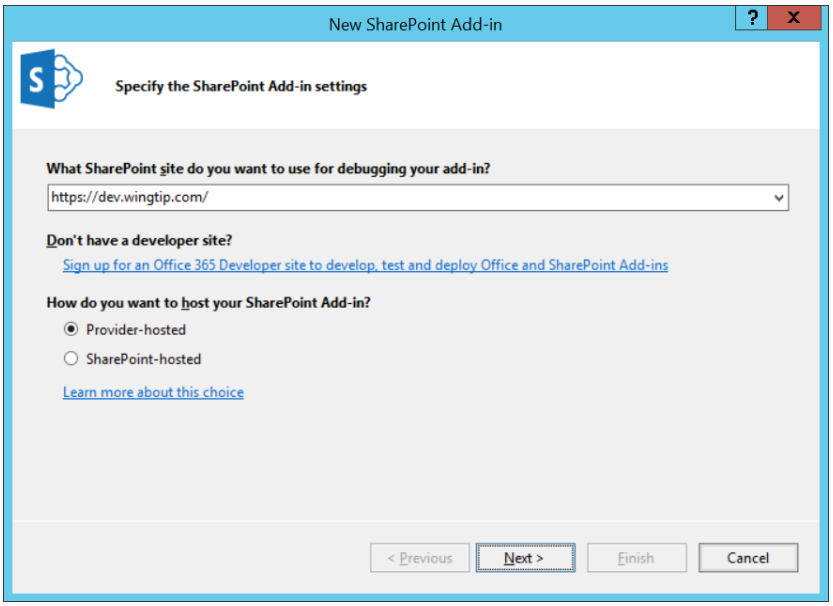


c) In the **New Add-in for SharePoint** wizard:

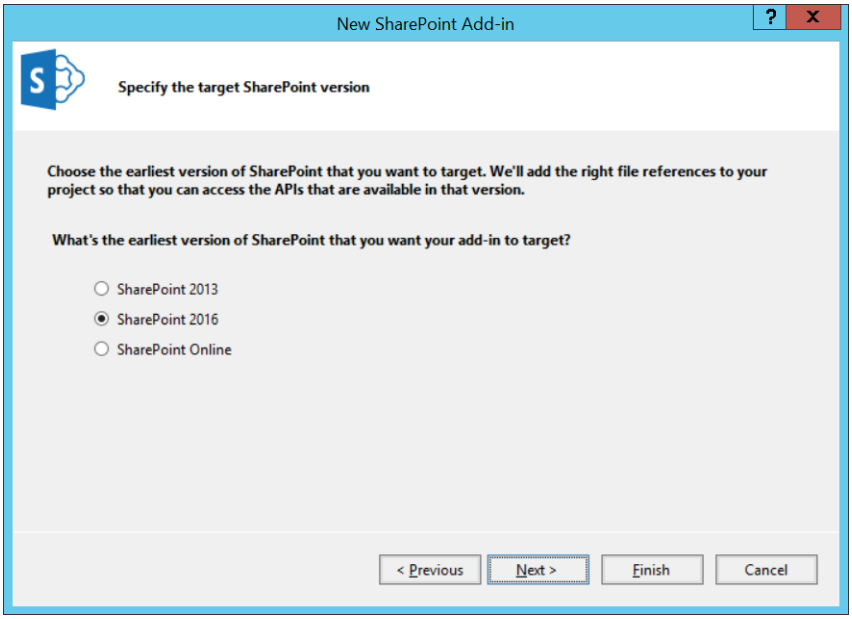
i) Enter the URL for the de site at **https://dev.wingtip.com**.

ii) Select **Provider-Hosted** as the hosting model.

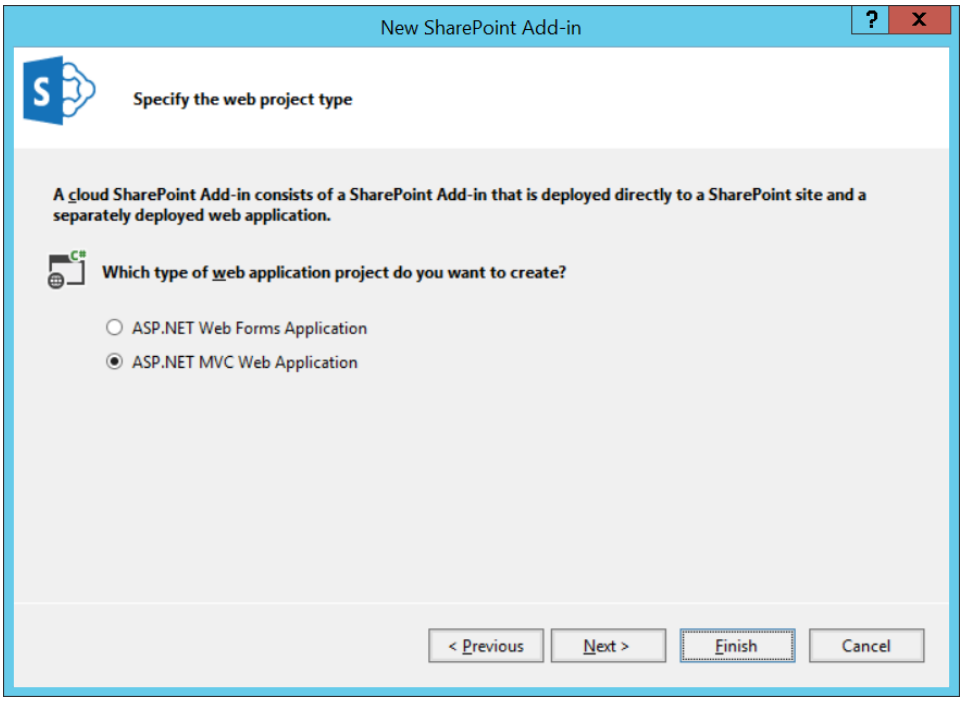
iii) Click **Next**.



d) On the **Specify the target SharePoint version** dialog, select **SharePoint 2016** and click **Next**.



e) In the **Specify the web project type** dialog, select **ASP.NET MVC Web Application** and click **Next**.



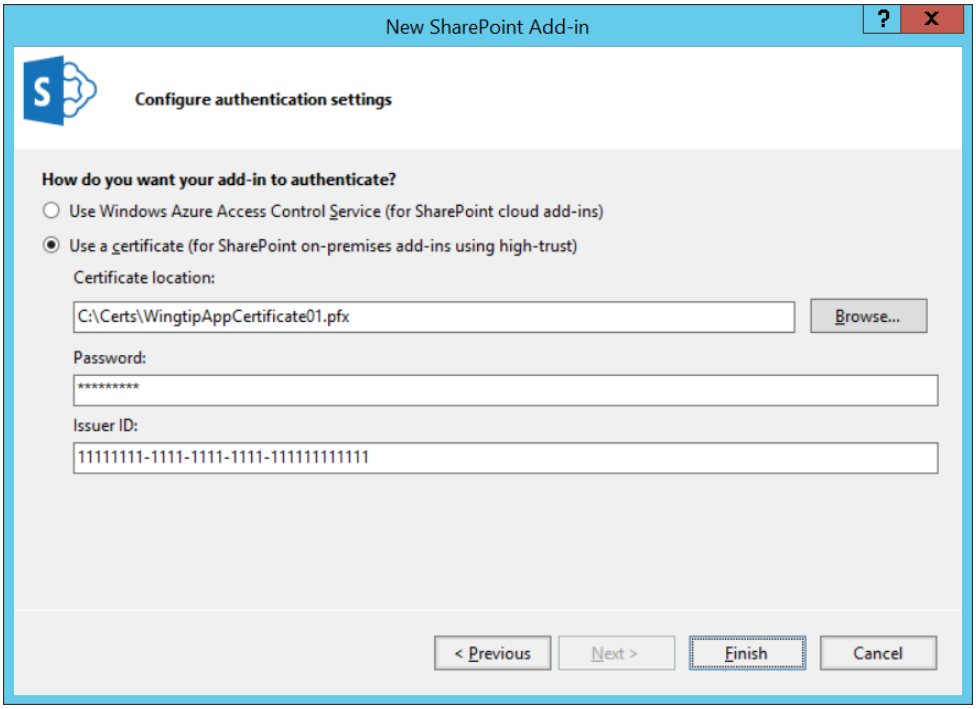
f) In the **Configure authentication settings** dialog, select the option to **Use a certificate (for SharePoint 0n-premises add-ins using high-trust)**. Enter the certification information for the certificate you created in the previous lab on add-in security.

i) Enter a **Certification location** of **C:\Certs\WingtipAppCertificate01.pfx**.

ii) Enter a **Password** of **Password1**.

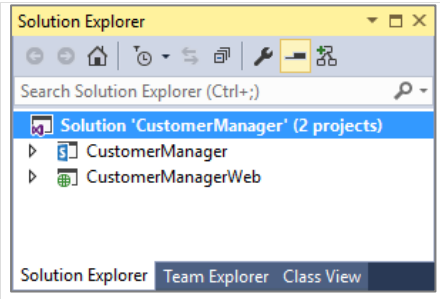
iii) Enter an **Issuer ID** of **11111111-1111-1111-1111-111111111111**.

iv) Click **Finish**.



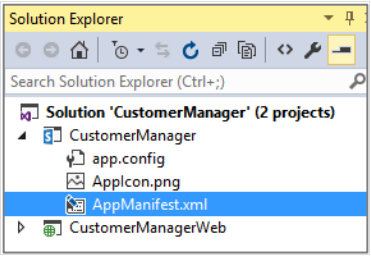
g) Wait until Visual Studio and the **New app for SharePoint** wizard completes its work and generates a new solution.

h) You should see a new Visual Studio solution which contains two projects. The top project named **CustomerManager** is the main project for the add-in whose primary purpose is to generate the app package. The bottom project named **CustomerManagerWeb** is an ASP.NET project based on MVC that will be used to implement the remote web.

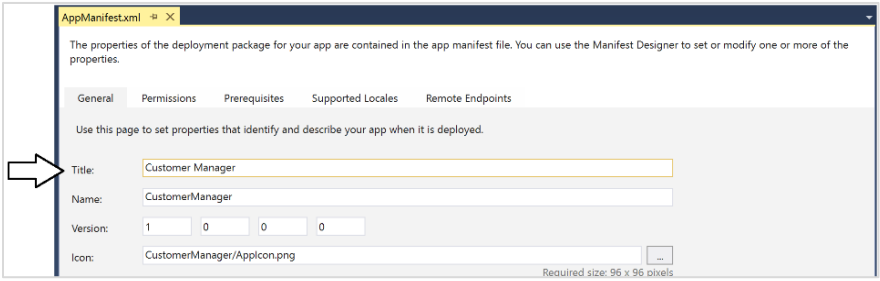


4. Update the title for the add-in using the app manifest.

a) Expand the top project named **CustomerManager** and double-click **AppManifest.xml** to open the app manifest in the Visual Studio app manifest designer.



b) Set the **Title** to **Customer Manager**.



c) Save and close the **AppManifest.xml** file.

5. Replace the generic app icon with a custom app icon.

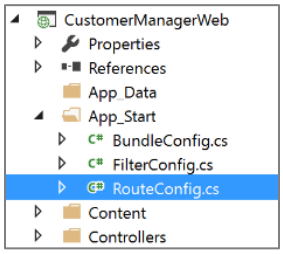
a) Using Windows Explorer, locate the icon file at the following location inside your Student folder.

```
C:\Student\Modules\07_ProviderHostedAddins\Lab\StarterFiles\AppIcon.png
```

b) Use this **AppIcon.png** file to replace the generic **AppIcon.png** file located at the root of the **CustomerManager** project.

You have finished working on the top project named **CustomerManager**. Now you will begin to work with the ASP.NET project named **CustomerManagerWeb** which will implement the remote web for the add-in using the MVC framework.

6. Examine the code supplied by Visual Studio to initialize the route map for this MVC application.
   a) In the **CustomerManagerWeb** project, expand the **App_Start** folder
   b) Locate the source file named **RouteConfig.cs**.

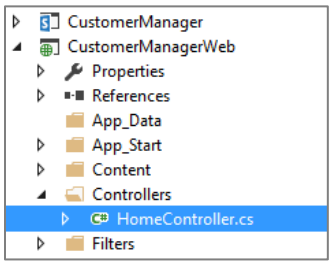


   c) Double-click on **RouteConfig.cs** to open this source file in a Code View window.
   d) You should see a **RouteConfig** class that contains a method named **RegisterRoutes**.
   e) Inside the implementation of the **RegisterRoutes** method, there is a call to the **MapRoute** method.

```
routes.MapRoute(
    name: "Default",
    url: "{controller}/{action}/{id}",
    defaults: new { controller = "Home", action = "Index", id = UrlParameter.Optional }
);
```

   f) You should be able to see that that code generated by Visual Studio configures a controller named **"Home"** as the default controller in this MVC application's top-level routing scheme.
   g) Close **RouteConfig.cs** and make sure not to save any changes.

7. Examine the existing code inside **HomeController.cs** file.
   a) In the **CustomerManagerWeb** project, expand the **Controllers** folder
   b) Locate the source file named **HomeController.cs**.



   c) Double click on the file named **HomeController.cs** to open the C# source file in a Code View window.
   d) Examine the Controller class that Visual Studio has generated for you. You should be able to see that there is a class named **HomeController** that inherits from the **Controller** class.
   e) You should be able to see that the **HomeController** class contains three action methods named **Index, About** and **Contact**.
   f) Remove the **SharePointContextFilter** from the **Index** method.
   g) Delete all the code inside the **Index** method and replace it with the following code.

```
public ActionResult Index() {
  ViewBag.Message = "Hello MVC!";
  return View();
}
```

   h) Delete the **About** method

i) Delete the **Contact** method.

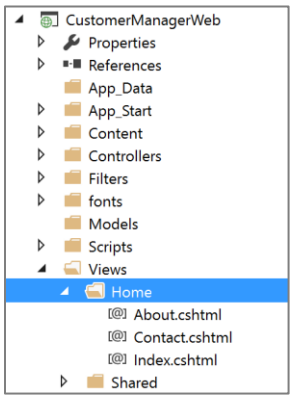
j) The **HomeController** class in your project should now be simplified and match the following code listing.

```
namespace CustomerManagerWeb.Controllers {
  public class HomeController : Controller {

    public ActionResult Index() {
      ViewBag.Message = "Hello MVC!";
      return View();
    }

  }
}
```

8. Modify the views for the **Home** controller.

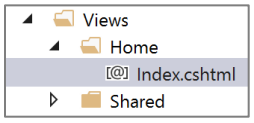
a) Expand the **Views** folder in the **CustomerManagerWeb** project.

b) Expand the **Home** folder inside the **Views** folder to see the razor views associated with the **Home** controller.



c) Delete the razor view files named **About.cshtml** and **Contact.cshtml**.

d) The **Home** folder should now only contain the razor view file named **Index.cshtml**.



e) Double-click **Home.cshtml** to open this file in a code editor window.

f) Delete all the existing code within **Home.cshtml** and replace it with the following code

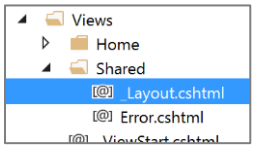
```
<h2>Welcome to Customer Manager</h2>

<div class="jumbotron">
  <p>@ViewBag.Message</p>
</div>
```

g) Save and close **Home.cshtml**.

9. Modify the Shared view for this MVC application.

a) Expand the **Shared** folder inside the **Views** folder.

b) Locate and double-click the shared view file named **_Layouts.cshtml** to open it in a code editor window,



c) Take a moment to inspect the code that Visual Studio has added into **_Layouts.cshtml**. Note that the file is used to generate the common layout for all the pages in your MVC application.

d) Look at the code inside **_Layouts.cshtml** and locate the **title** element inside the **head** element.

```
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>@ViewBag.Title - My ASP.NET Application</title>
    @Styles.Render("~/Content/css")
    @Scripts.Render("~/bundles/modernizr")
</head>
```

e) Update the **title** element to match the following code listing.

```
<title>Customer Manager</title>
```

f) Move down inside **_Layouts.cshtml** and locate the first **div** element in the **body** section which defines the navigation bar.

```
<body>
    <div class="navbar navbar-inverse navbar-fixed-top">
        <div class="container">
            <div class="navbar-header">
```

g) Modify the **class** attribute of this div by removing the **navbar-fixed-top** class so that the div matches the following code listing.

```
<div class="navbar navbar-inverse">
```

h) Move down the code in **_Layouts.cshtml**, and locate the line of code which matches the following code listing.

```
@Html.ActionLink("Application name", "Index", "Home", new {area =""}, new { @class = "navbar-brand" })
```

i) Replace the text value of **Application name** with **Customer Manager**.

```
@Html.ActionLink("Customer Manager", "Index", "Home", new {area =""}, new { @class = "navbar-brand" })
```

j) Move down the code in **_Layouts.cshtml**, and locate the code which adds navigation links to the navbar.

```
<div class="navbar-collapse collapse">
    <ul class="nav navbar-nav">
        <li>@Html.ActionLink("Home", "Index", "Home")</li>
        <li>@Html.ActionLink("About", "About", "Home")</li>
        <li>@Html.ActionLink("Contact", "Contact", "Home")</li>
    </ul>
</div>
```

k) Remove the lines of code that add navigation links for **About** and **Contact** and leave a single navigation link for **Home**.

```
<div class="navbar-collapse collapse">
    <ul class="nav navbar-nav">
        <li>@Html.ActionLink("Home", "Index", "Home")</li>
    </ul>
</div>
```

l) Move down and locate the **div** element which contains **@RenderBody**.

```
div class="container body-content">
    @RenderBody()
    <hr />
    <footer>
        <p>&copy; @DateTime.Now.Year - My ASP.NET Application</p>
    </footer>
</div>
```

m) Remove the **hr** element and the **footer** element so the div matches the following code listing.

```
<div class="container body-content">
    @RenderBody()
</div>
```

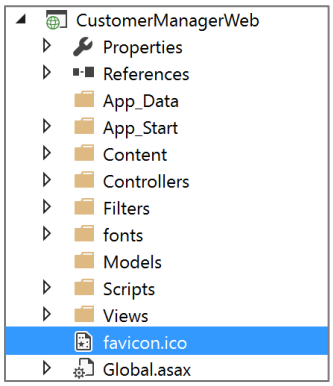
n) Save and close _**Layouts.cshtml**.

10. Replace the generic **favicon.ico** file in the **CustomerManagerWeb** project with a custom **favicon.ico** file.

a) Using Windows Explorer, locate the icon file at the following location inside your Student folder.

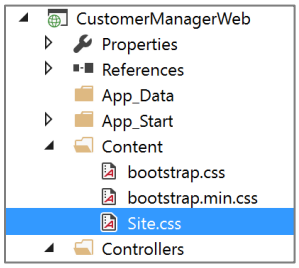
```
C:\Student\Modules\07_ProviderHostedAddins\Lab\StarterFiles\favicon.ico
```

b) Use this custom **favicon.ico** file to replace the generic **favicon.ico** file in the **CustomerManagerWeb** project.



11. Add a few custom CSS rules to style the user interface for your MVC application.

a) Open the **Contents** folder and locate the project's main CSS file named **Site.css**.



Note that the shared view _**Layouts.cshtml** already contains code to process a CSS bundle that will automatically add a link to **Sites.css** for all pages. Once you have added custom CSS styles to **Sites.css**, they will be automatically applied to all pages.

b) Double-click on **Sites.css** to open it in a code editor window.

c) Delete the existing contents inside **Sites.css**.

d) Add two new CSS rules to style **h2** header elements and the first column of a **table** element.

```
h2{
  font-size: 16pt;
  line-height: 1.0;
  color: darkblue;
  padding-bottom: 2px;
  border-bottom: 1px solid darkblue;
  margin-top: 0;
  margin-bottom: 12px;
}

.table tr td:first-child {
  width: 200px;
}
```

e) Save and close **Sites.css**.

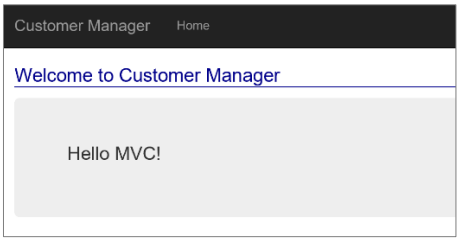
12. Test your work by running your new provider-hosted add-in in the Visual Studio debugger.

a) Press **F5** to begin debugging.

b) If you are prompted with a SharePoint page that asks whether you to trust the add-in, click **Trust it**.

c) Once the Visual Studio debugger has installed the add-in, you will be redirected to the add-in start page.

d) Verify that the start page for the **Customer Manager** add-in displays properly as shown in the following screenshot.



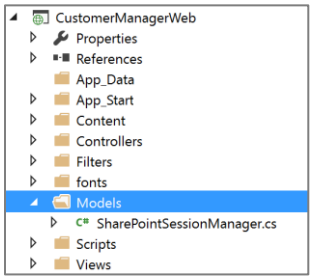
e) Once you have tested the start page, close the browser window, return to Visual Studio and terminate the debugging session.

At this point, you have created a new Visual Studio solution for a provider-hosted add-in and you have removed some of the extraneous project elements that were added by Visual Studio. Now you can begin to implement the core functionality for your add-in.

Exercise 2: Tracking SharePoint Session State in a Provider-Hosted Add-in

In this exercise, you will continue working the Customer Manager add-in that you created in the previous exercise. You will extend the Customer Manager add-in by adding code to track SharePoint session state across requests using an ASP.NET session object.

1. Add a new class named **SharePointSessionManager** to track SharePoint session state.
   a) Right-click on the **Models** folder of the **CustomerManagerWeb** and click the **Add >> Class…** command.
   b) Give the new class a name of **SharePointSessionManager**.
   c) Verify that Visual Studio has added a new C# source file named **SharePointSessionManager** into the **Models** folder.



   d) Examine the code in **SharePointSessionManager.cs** containing the starting point for the **SharePointSessionManager** class.

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

namespace CustomerManagerWeb.Models {
  public class SharePointSessionManager {
  }
}
```

Note that this lab will now require you to add quite a bit of code. It will take quite a while to complete this section if you plan to type in all this code by hand. Please make use of the electronic version of this lab so you can copy-and-paste code fragments to save time.

2. Update the **using** statements at the top of **SharePointSessionManager.cs** to match the following code listing.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net.Http;
using System.Web;
using System.Web.Mvc;
using System.Web.SessionState;
using Microsoft.IdentityModel.S2S.Protocols.OAuth2;
using Newtonsoft.Json;
```

3. Insert a few lines between the namespace definition and class definition.

```
namespace CustomerManagerWeb.Models {


  public class SharePointSessionManager {
  }
}
```

4. Add two new classes named SharePointSessionState and SharePointUserResult.

```
namespace CustomerManagerWeb.Models {

  public class SharePointSessionState {
  }

  public class SharePointUserResult {
  }

  public class SharePointSessionManager {
  }
}
```

5. Implement the **SharePointSessionState** class using the following code.

```
public class SharePointSessionState {
    public string RemoteWebUrl { get; set; }
    public string HostWebUrl { get; set; }
    public string HostWebDomain { get; set; }
    public string HostWebTitle { get; set; }
    public string CurrentUserAccountName { get; set; }
    public string CurrentUserDisplayName { get; set; }
    public string CurrentUserEmail { get; set; }
}
```

6. Implement the **SharePointUserResult** class using the following code.

```
public class SharePointUserResult {
  public string Title { get; set; }
  public string Email { get; set; }
  public string IsSiteAdmin { get; set; }
}
```

7. Extend the **SharePointSessionManager** class by adding three new static fields named **request**, **session** and **sessionState** using the following code.

```
public class SharePointSessionManager {

  static HttpRequest request = HttpContext.Current.Request;
  static HttpSessionState session = HttpContext.Current.Session;
  static SharePointSessionState sessionState = new SharePointSessionState();

}
```

8. Extend the **SharePointSessionManager** class by adding five new static methods named **ExecuteGetRequest**, **AuthenticateUser**, **UserIsAthenticated**, **InitializeRequest** and **GetSharePointSessionState** using the following code.

```
public class SharePointSessionManager {

  static HttpRequest request = HttpContext.Current.Request;
  static HttpSessionState session = HttpContext.Current.Session;
  static SharePointSessionState sessionState = new SharePointSessionState();

  private static string ExecuteGetRequest(string restUri, string accessToken) { }

  private static void AuthenticateUser() { }

  private static bool UserIsAuthentiated() { }

  public static void InitializeRequest(ControllerBase controller) { }

  public static SharePointSessionState GetSharePointSessionState() { }

}
```

9. Implement the **ExecuteGetRequest** method using the following code.

```
private static string ExecuteGetRequest(string restUri, string accessToken) {
  // setup request
  HttpClient client = new HttpClient();
  client.DefaultRequestHeaders.Add("Authorization", "Bearer " + accessToken);
  client.DefaultRequestHeaders.Add("Accept", "application/json");
  // execute request
  HttpResponseMessage response = client.GetAsync(restUri).Result;
  // handle response
  if (response.IsSuccessStatusCode) {
    return response.Content.ReadAsStringAsync().Result;
  }
  else {
    // ERROR during HTTP GET operation
    return string.Empty;
  }
}
```

10. Implement the **AuthenticateUser** method using the following code.

```
private static void AuthenticateUser() {

    sessionState.RemoteWebUrl = request.Url.Authority;
    sessionState.HostWebUrl = request["SPHostUrl"];
    sessionState.HostWebDomain = (new Uri(sessionState.HostWebUrl)).Authority;
    sessionState.HostWebTitle = request.Form["SPSiteTitle"];

    // get windows authenticated user name
    sessionState.CurrentUserAccountName = HttpContext.Current.User.Identity.Name;
    // get access token to make RESt call
    var spContext = SharePointContextProvider.Current.GetSharePointContext(HttpContext.Current);
    string accessToken = spContext.UserAccessTokenForSPHost;
    // call SharePoint REST API to get information about current user
    string restUri = sessionState.HostWebUrl + "/_api/web/currentUser/";
    string jsonCurrentUser = ExecuteGetRequest(restUri, accessToken);

    // convert json result to strongly-typed C# object
    SharePointUserResult userResult = JsonConvert.DeserializeObject<SharePointUserResult>(jsonCurrentUser);
    sessionState.CurrentUserDisplayName = userResult.Title;
    sessionState.CurrentUserEmail = userResult.Email;

    // write session state out to ASP.NET session object
    session["SharePointSessionState"] = sessionState;

    // update UserIsAuthenticated session variable
    session["UserIsAuthenticated"] = "true";
}
```

11. Implement the **UserIsAuthentiated** method using the following code.

```
private static bool UserIsAuthentiated() {
  return (session["UserIsAuthenticated"] != null) &&
         (session["UserIsAuthenticated"].Equals("true"));
}
```

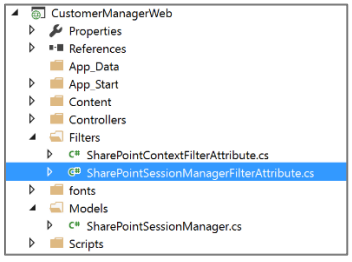
12. Implement the **InitializeRequest** method using the following code.

```
public static void InitializeRequest(ControllerBase controller) {

  if (!UserIsAuthentiated()) {
    AuthenticateUser();
  }
  else {
    // if user is authenticated, copy session state from previous request
    sessionState = (SharePointSessionState)session["SharePointSessionState"];
  }

  // add session state to ViewBag to make it accessible in views
  controller.ViewBag.HostWebUrl = sessionState.HostWebUrl;
  controller.ViewBag.HostWebTitle = sessionState.HostWebTitle;
  controller.ViewBag.CurrentUserDisplayName = sessionState.CurrentUserDisplayName;
}
```

13. Implement the **GetSharePointSessionState** method using the following code.

```
public static SharePointSessionState GetSharePointSessionState() {
  return sessionState;
}
```

14. Create a new MVC filter named **SharePointSessionManagerFilterAttribute**.
    a) Expand the **Filters** folder of the **CustomerManagerWeb** project.
    b) You should see that the **Filters** folder already contains a file named **SharePointContextFilterAttribute.cs** that was created by Visual Studio when you created the project. You can ignore this filter because you will not be using it in your add-in project.
    c) Right-click on the **Filters** folder and click the **Add >> Class…** menu command.
    d) Give the new class a name of **SharePointSessionManagerFilterAttribute**.
    e) Verify that Visual Studio has added a new C# source file named **SharePointSessionManagerFilterAttribute.cs** into the **Filters** folder. Note that Visual Studio



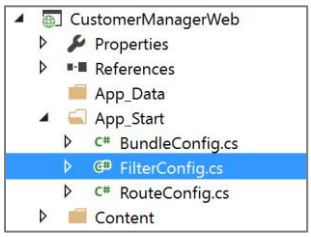
    f) Delete all the code inside **SharePointSessionManagerFilterAttribute.cs** and replace it with the following code.

```
using System.Web.Mvc;
using CustomerManagerWeb.Models;

namespace CustomerManagerWeb.Filters {
  public class SharePointSessionManagerFilterAttribute : ActionFilterAttribute {

    public override void OnActionExecuting(ActionExecutingContext filterContext) {
      SharePointSessionManager.InitializeRequest(filterContext.Controller);
    }

  }
}
```

g) Save and close **SharePointSessionManagerFilterAttribute.cs**.

15. Configure the **SharePointSessionManagerFilterAttribute** class to run as a global filter.

a) Expand the **App_Start** folder inside the **CustomerManagerWeb** project and locate the source file named **FilterConfig.cs**.



b) Double-click on **FilterConfig.cs** to open it inside a code editor window.

c) At the top of FilterConfig.cs, add a using statement for the

```
using System.Web;
using System.Web.Mvc;
using CustomerManagerWeb.Filters;
```

d) Locate the line of code in the **RegisterGlobalFilters** method that adds a global filter for **HandleErrorAttribute**.

```
filters.Add(new HandleErrorAttribute());
```

e) Add a new line of code to add the **SharePointSessionManagerFilterAttribute** class as a global filter.

```
namespace CustomerManagerWeb {
  public class FilterConfig {
    public static void RegisterGlobalFilters(GlobalFilterCollection filters) {
      filters.Add(new HandleErrorAttribute());
      filters.Add(new SharePointSessionManagerFilterAttribute());
    }
  }
}
```

f) Save and close **FilterConfig.cs**.

16. Update the view for the **Index** action method of the **Home** controller home page to display the name of the current user.

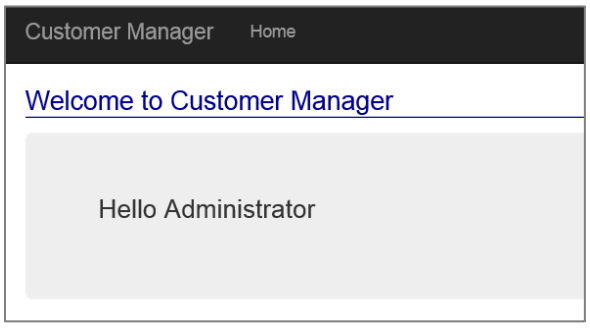
a) Open the **Index.cshtml** file in the **Home** folder inside the **Views** folder.

b) Update the contents of the **p** element as shown in the following code listing.

```
<h2>Welcome to Customer Manager</h2>

<div class="jumbotron">
  <p>Hello @ViewBag.CurrentUserDisplayName</p>
</div>
```

c) Press **F5** to run the add-in project in the Visual Studio debugger.

d) Verify that the Customer Manager add-start page displays the name of the current user.



Every SharePoint add-in has an important requirement for its user interface to provide a link which allows the user to navigate back to the host web. In the next step you will meet this requirement by adding a new action link to the add-in main navigation bar.

17. Update the view for the shared layout to provide a link to navigate back to the host web.
    a) Open the **_Layout.cshtml** file in the **Shared** folder inside the **Views** folder.
    b) Inside **_Layouts.cshtml**, locate the **div** element which contains the **ul** element with the action link to the **Index** action.

```
<div class="navbar-collapse collapse">
  <ul class="nav navbar-nav">
    <li>@Html.ActionLink("Home", "Index", "Home")</li>
  </ul>
</div>
```

    c) Add a new **ul** element under the existing **ul** element and

```
<div class="navbar-collapse collapse">
  <ul class="nav navbar-nav">
    <li>@Html.ActionLink("Home", "Index", "Home")</li>
  </ul>
  <ul>

  </ul>
</div>
```

    d) Add a **class** attribute to the **ul** element and reference the bootstrap CSS classes named **nav**, **navbar-nav**, **pull-right**.

```
<div class="navbar-collapse collapse">
  <ul class="nav navbar-nav">
    <li>@Html.ActionLink("Home", "Index", "Home")</li>
  </ul>
  <ul class="nav navbar-nav pull-right">
  </ul>
</div>
```

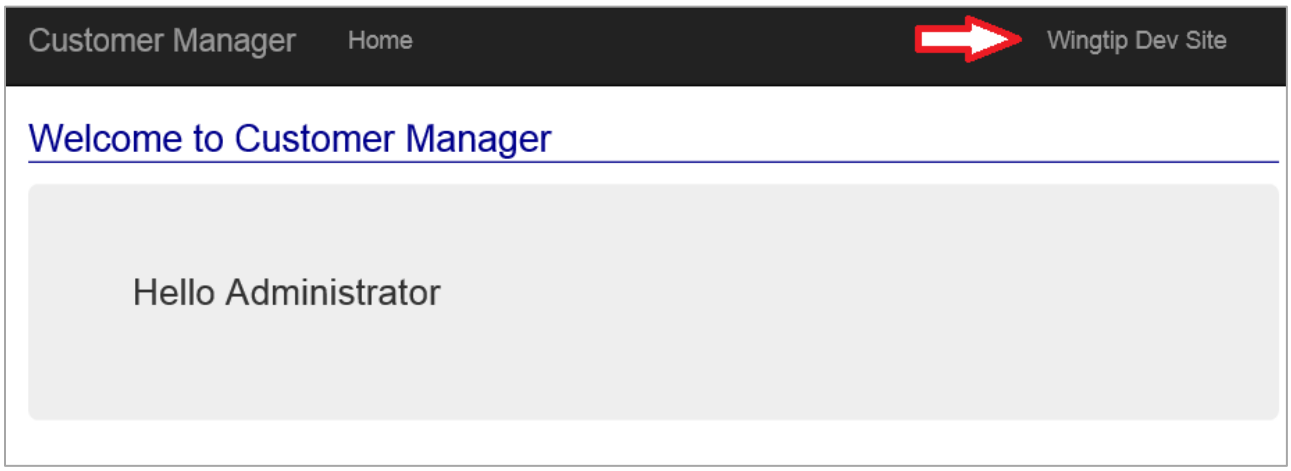
    e) Inside the **ul** element, add an **li** element with an inner anchor (**a**) element to provide a link back to the host web.

```
<ul class="nav navbar-nav pull-right">
  <li><a href="@ViewBag.HostWebUrl">@ViewBag.HostWebTitle</a></li>
</ul>
```

    f) Your code should now match the following code listing.

```
<div class="navbar-collapse collapse">
  <ul class="nav navbar-nav">
    <li>@Html.ActionLink("Home", "Index", "Home")</li>
  </ul>
  <ul class="nav navbar-nav pull-right">
    <li><a href="@ViewBag.HostWebUrl">@ViewBag.HostWebTitle</a></li>
  </ul>
</div>
```

    g) Save and close **_Layouts.cshtml**.
    h) Press **F5** to run the add-in project in the Visual Studio debugger.
    i) Verify that the Customer Manager add-start page displays the link back to the host web.



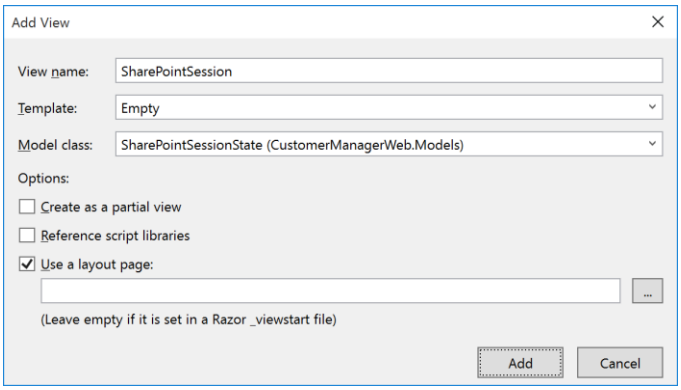
18. Add a new action method named **SharePointSession** to create a page which displays SharePoint session state.
    a) Open **HomeController.cs** in a code editor window.
    b) Add a using statement to import the **CustomerManagerWeb.Models** namespace.

```
using CustomerManagerWeb.Models;
```

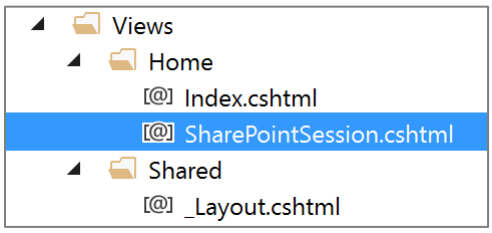
    c) Add a new action methods named **SharePointSession**.
    d) Implement the **SharePointSession** action method using the following code.

```
public ActionResult SharePointSession() {
  return View(SharePointSessionManager.GetSharePointSessionState());
}
```

    e) Save your changes to **HomeController.cs**.
19. Create a razor view file for the **SharePointSession** action method.
    a) Right-click the whitespace inside the **SharePointSession** method and select the **Add View…** menu command.
    b) You will be prompted with the **Add View** dialog.
    c) Change the **Template** selection from **Empty (without model)** to **Empty**.
    d) Set the **Model class** property to **SharePointSessionState**.
    e) Click the **Add** button to create the new razor view file.



    f) You should now see a new file named **SharePointSession.cshtml** inside the view folder for the Home controller.



20. Implement the view definition in **SharePointSession.cshtml**.
    a) Inspect the code that was automatically added to **SharePointSession.cshtml**.

```
@model CustomerManagerWeb.Models.SharePointSessionState

@{
    ViewBag.Title = "SharePointSession";
}

<h2>SharePointSession</h2>
```

    b) Remove the code that updates the **ViewBag.Title** property and modify the content of the h2 element as shown below.

```
@model CustomerManagerWeb.Models.SharePointSessionState

<h2>SharePoint Session Information</h2>
```

    c) Below the **h2** element, add the following HTML code to create a table that displays SharePoint session information to the user.

```
<table class="table table-bordered">
    <tr>
        <td>Remote Web Url:</td>
        <td>@Model.RemoteWebUrl</td>
    </tr>
    <tr>
        <td>Host Web Url:</td>
        <td>@Model.HostWebUrl</td>
    </tr>
    <tr>
        <td>Host Web Domain:</td>
        <td>@Model.HostWebDomain</td>
    </tr>
    <tr>
        <td>Host Web Title:</td>
        <td>@Model.HostWebTitle</td>
    </tr>
    <tr>
        <td>Current User Account Name:</td>
        <td>@Model.CurrentUserAccountName</td>
    </tr>
    <tr>
        <td>Current User Display Name:</td>
        <td>@Model.CurrentUserDisplayName</td>
    </tr>
    <tr>
        <td>Current User Email:</td>
        <td>@Model.CurrentUserEmail</td>
    </tr>
</table>
```

    d) Save and close **SharePointSession.cshtml**.

21. Update the shared layout to provide an action link to the **SharePointSession** action method.
    a) Open the **_Layout.cshtml** file in the **Shared** folder inside the **Views** folder.
    b) Inside **_Layouts.cshtml**, locate the **div** element which contains the **ul** element with the action link to the **Index** action.
    c) Add a new **li** element with an action link to navigate to the **SharePointSession** action as shown in the following code listing.

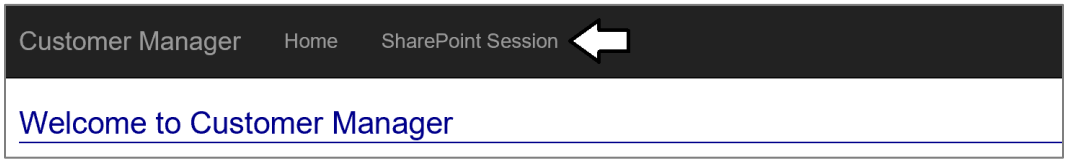
```
<div class="navbar-collapse collapse">
  <ul class="nav navbar-nav">
    <li>@Html.ActionLink("Home", "Index", "Home")</li>
    <li>@Html.ActionLink("SharePoint Session", "SharePointSession", "Home")</li>
  </ul>
  <ul class="nav navbar-nav pull-right">
    <li><a href="@ViewBag.HostWebUrl">@ViewBag.HostWebTitle</a></li>
  </ul>
</div>
```

d) Save and close _**Layouts.cshtml**.

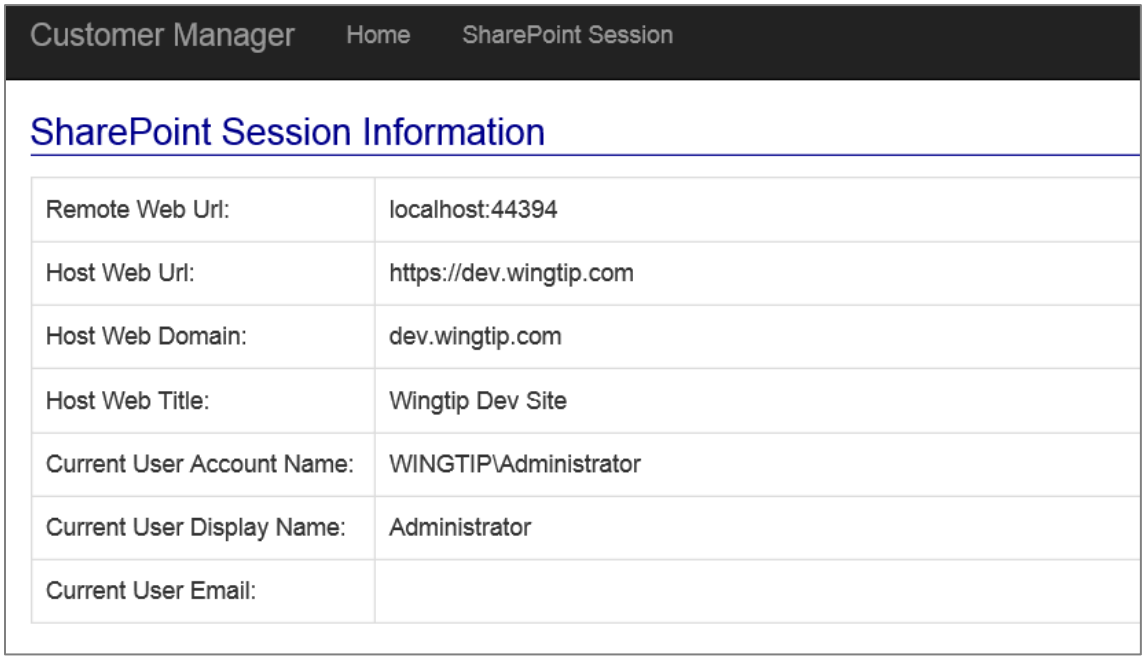
22. Test out the **SharePointSession** action method in the Visual Studio debugger.

a) Press **F5** to run the add-in project in the Visual Studio debugger.

b) Verify that the Customer Manager start page displays the new navigation link for the **SharePointSession** action method.



c) Click on the navigation link for the **SharePointSession** action method so you can test out the view you have just implemented. You should see a HTML table showing SharePoint session information as shown in the following screenshot.



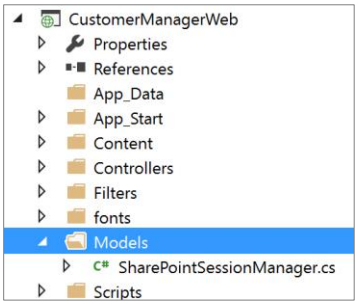
d) Once you have tested the page for the **SharePointSession** action, close the browser window, return to Visual Studio and terminate the debugging session.

At this point, you have extended the provider-hosted add-in project to track SharePoint session state. Adding this type of code is very common undertaking in the development of provided-hosted add-ins because you must track essential session data such as the Host Web URL and user profile information across requests.

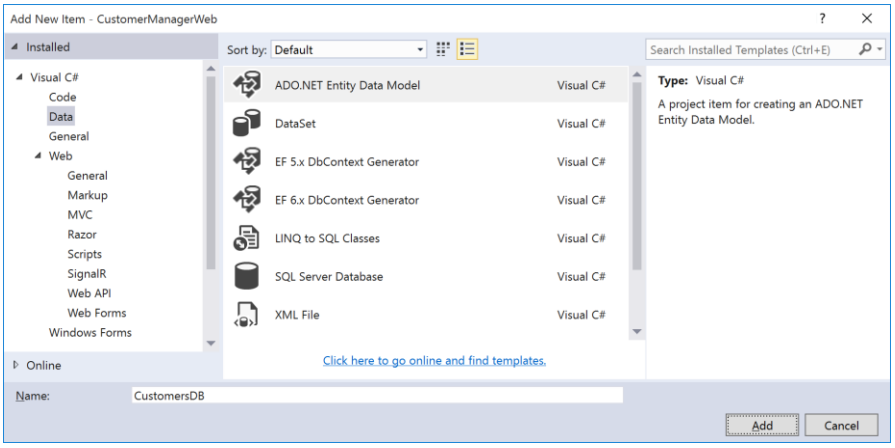
Exercise 3: Creating a Custom Database using the Entity Framework

In this exercise, you will continue working on the Customer Manager add-in that you have worked on in previous exercises. You will extend the Customer Manager add-in by creating an Entity Framework data model to generate a custom SQL Server database to track customer data.

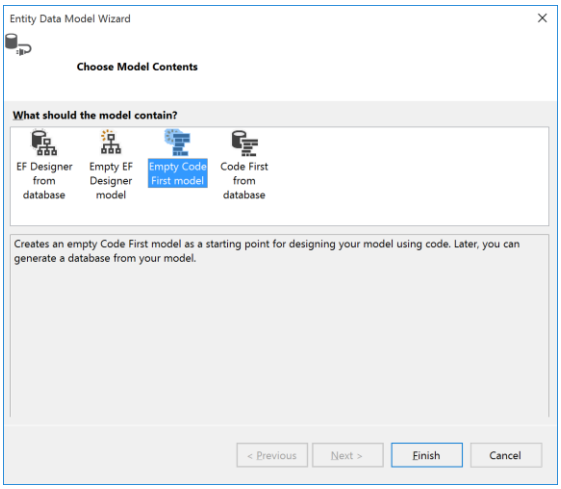
23. Add a new project item for a new Entity Framework data model named **CustomerDB** using the code-first approach.
    a) In the **CustomerManagerWeb** project, expand the **Models** folder.
    b) At this point, the **Models** folder should contain one C# file named **SharePointSessionManager.cs**.



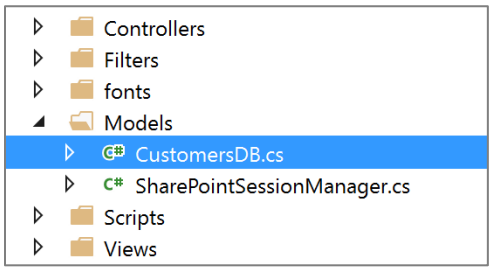
    c) Right-click on the **Models** folder and select the **Add > New Item…** menu command.
    d) When you are prompted with the **Add New Item** dialog, select **Visual C# > Data** in the left-hand filter panel and then select the **ADO.NET Entity Data Model** project item template. Enter a name of **CustomersDB** and click **Add**.



    e) When you are prompted by the **Entity Data Model Wizard** dialog to **Choose Model Contents**, select the **Empty Code First Model** as shown in the following screenshot.



f) Verify that Visual Studio added a new C# source file named **CustomersDB.cs** in the **Models** folder.



g) Inspect the code that Visual Studio added in **CustomersDB.cs**.

24. Implement the classes required in the Entity Framework data model.

a) Update the **using** statements in from **CustomersDB.cs** and simplify the code to match the following code listing,

```
using System;
using System.ComponentModel;
using System.ComponentModel.DataAnnotations;
using System.Data.Entity;

namespace CustomerManagerWeb.Models {

  public class CustomersDB : DbContext {
    public CustomersDB()
      : base("name=CustomersDB") {
    }
  }
}
```

b) Add a new public class named **Customer** beneath the **CustomersDB** class.

```
namespace CustomerManagerWeb.Models {

  public class CustomersDB : DbContext {
    public CustomersDB()
      : base("name=CustomersDB") {
    }
  }

  public class Customer {
  }
}
```

c) Implement the **Customers** class by adding properties for **Id**, **FirstName**, **LastName**, **Company**, **Email**, **WorkPhone** and **HomePhone** as shown in the following code listing.

```
public class Customer {
  public int Id { get; set; }
  public string FirstName { get; set; }
  public string LastName { get; set; }
  public string Company { get; set; }
  public string Email { get; set; }
  public string WorkPhone { get; set; }
  public string HomePhone { get; set; }
}
```

d) Apply the **[Key]** attribute to the **Id** property to indicate that the **Id** value for each customer should be used as the primary key.

```
public class Customer {
  [Key]
  public int Id { get; set; }
```

e) Apply the **[DisplayName]** attribute to the **FirstName** property, the **LastName** property, the **WorkPhone** property and the **HomePhone** property to provide a more human-readable display name with a space as shown in the following code listing.

```
public class Customer {
  [Key]
  public int Id { get; set; }
  [DisplayName("First Name")]
  public string FirstName { get; set; }
  [DisplayName("Last Name")]
  public string LastName { get; set; }
  public string Company { get; set; }
  public string Email { get; set; }
  [DisplayName("Work Phone")]
  public string WorkPhone { get; set; }
  [DisplayName("Home Phone")]
  public string HomePhone { get; set; }
}
```

f) Extend the **CustomersDB** class by adding a new virtual property named **Customers** based on the type **DbSet<Customer>**.

```
public class CustomersDB : DbContext {
  public CustomersDB()
    : base("name=CustomersDB") {
  }
  public virtual DbSet<Customer> Customers { get; set; }
}
```

g) At this point, the contents of the **CustomersDB.cs** file should match the code following listing.

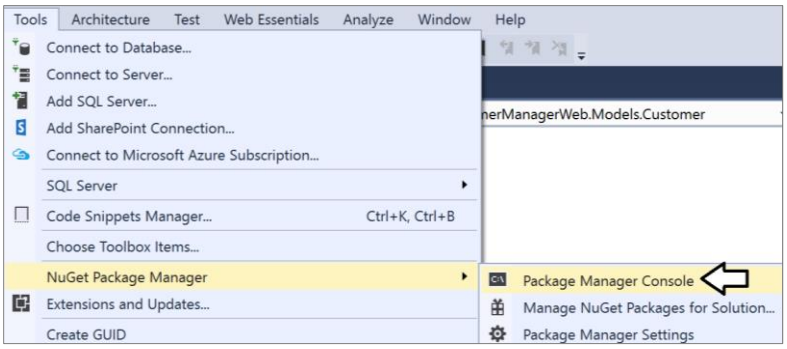
```
using System;
using System.ComponentModel;
using System.ComponentModel.DataAnnotations;
using System.Data.Entity;

namespace CustomerManagerWeb.Models {

  public class CustomersDB : DbContext {
    public CustomersDB()
        : base("name=CustomersDB") {
    }
    public virtual DbSet<Customer> Customers { get; set; }
  }

  public class Customer {
    [Key]
    public int Id { get; set; }
    [DisplayName("First Name")]
    public string FirstName { get; set; }
    [DisplayName("Last Name")]
    public string LastName { get; set; }
    public string Company { get; set; }
```

```
    public string Email { get; set; }
    [DisplayName("Work Phone")]
    public string WorkPhone { get; set; }
    [DisplayName("Home Phone")]
    public string HomePhone { get; set; }
  }
}
```

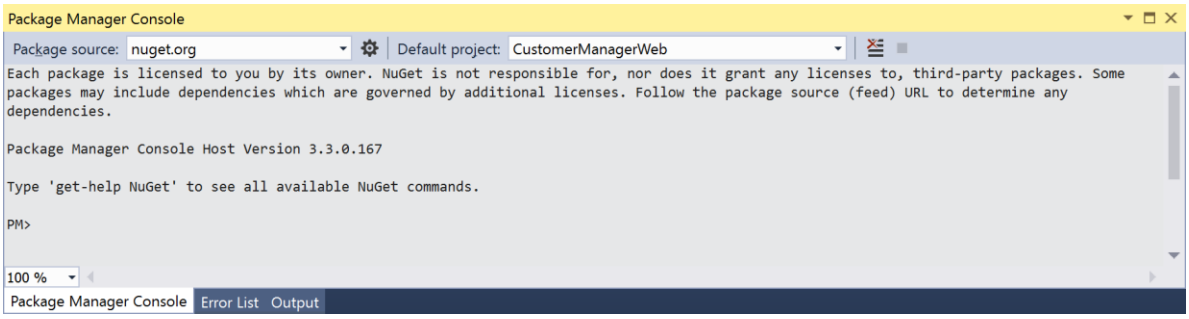
h) Save and close **CustomersDB.cs**.

25. Enable Entity Framework migrations for the current project.

a) Drop down the Visual Studio **Tools** menu and select **NuGet Package Manager > Package Manager Console**.



b) You should see the **Package Manager Console** appears at the bottom of the Visual Studio window.

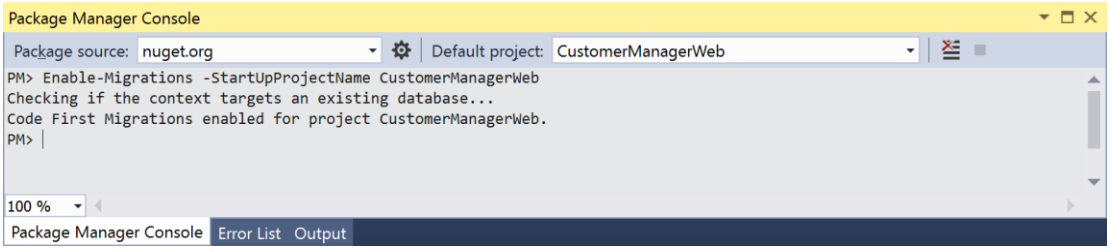


c) Using the mouse, place the cursor to the right of the **PM>** prompt in the Package Manager Console.

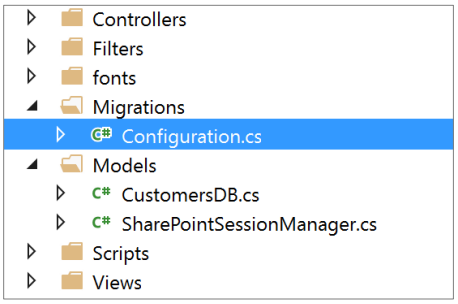
d) Type the following command and press the **ENTER** key to execute it.

```
Enable-Migrations -StartUpProjectName CustomerManagerWeb
```

e) Wait until you see the message that Code First Migrations have been enabled for the **CustomerManagerWeb** project.



f) Once Migrations have been enabled, you should see that Visual Studio has created a new **Migrations** folder in the **CustomerManagerWeb** project and inside that folder you should see a new C# source file named **Configuration.cs**.



g) Examine the code that Visual Studio has generated inside **Configuration.cs**. You should see that there is a class named **Configuration** inheriting from **DbMigrationsConfiguration** which contains a public default constructor and a **Seed** method.

```
internal sealed class Configuration : DbMigrationsConfiguration<CustomerManagerWeb.Models.CustomersDB>{

  public Configuration() {
    AutomaticMigrationsEnabled = false;
  }

  protected override void Seed(CustomerManagerWeb.Models.CustomersDB context) {
  }

}
```

26. Use the **Seed** method to populate the **Customers** table with sample customer records.

a) Inside **Configuration.cs**, add a new **using** statement to import the **CustomerManagerWeb.Models** namespace.

```
namespace CustomerManagerWeb.Migrations {
  using System;
  using System.Data.Entity;
  using System.Data.Entity.Migrations;
  using System.Linq;
  using CustomerManagerWeb.Models;
```

b) The **Seed** method is used to add sample data into tables when the Entity Framework creates a database using Code First migration. For example, you can modify the **Seed** method as shown in the following code listing to add a customer record.

```
protected override void Seed(CustomersDB context) {

  context.Customers.Add(
    new Customer {
      FirstName = "Buck",
      LastName = "Adams",
      Company = "The Hanso Foundation",
      Email = "Buck.Adams@TheHansoFoundation.com",
      WorkPhone = "1(503)777-1111",
      HomePhone = "1(503)666-8888"
    }
  );
}
```

Instead of having you type in lots of code to add a dozen customer records, you will copy-and-paste code we have provided to you in a text file in your student folder.

c) Using Windows Explorer, locate and open the text file at the following path in your student folder.

```
C:\Student\Modules\MVC\Lab\Snippets\Configuration.Seed.cs.txt
```

d) Copy the code for the **Seed** method out of the text file and paste into **Configuration.cs** to replace the existing **Seed** method.

e) At this point, the **Configuration** class in your project should match the following screenshot.

```
internal sealed class Configuration : DbMigrationsConfiguration<CustomerManagerWeb.Models.CustomersDB> {
  public Configuration() {
    AutomaticMigrationsEnabled = false;
  }

  protected override void Seed(CustomersDB context) {
    context.Customers.Add(new Customer { FirstName = "Buck", LastName = "Adams", Company = "The Hanso Foundation",
    context.Customers.Add(new Customer { FirstName = "Barbra", LastName = "Wiggins", Company = "Soylent Corporati
    context.Customers.Add(new Customer { FirstName = "Austin", LastName = "Small", Company = "LuthorCorp", Email =
    context.Customers.Add(new Customer { FirstName = "Rufus", LastName = "McMahon", Company = "ComTron", Email = '
    context.Customers.Add(new Customer { FirstName = "Irving", LastName = "McGee", Company = "Krusty Burger", Emai
    context.Customers.Add(new Customer { FirstName = "Winston", LastName = "Burke", Company = "Peach Pit", Email =
    context.Customers.Add(new Customer { FirstName = "Roscoe", LastName = "Park", Company = "Astromech", Email = '
    context.Customers.Add(new Customer { FirstName = "Ivy", LastName = "Gibbs", Company = "Brown Streak Railroad",
    context.Customers.Add(new Customer { FirstName = "Zack", LastName = "Miller", Company = "Groovy Smoothie", Ema
    context.Customers.Add(new Customer { FirstName = "Bob", LastName = "Carson", Company = "Contoso", Email = "Bo
    context.Customers.Add(new Customer { FirstName = "Dexter", LastName = "Vargas", Company = "The Hanso Foundati
    context.Customers.Add(new Customer { FirstName = "Pedro", LastName = "McCray", Company = "Itex", Email = "Ped
  }
}
```

f) Save and close **Configuration.cs**.

27. Update the connection string for the Entity Framework data model in the **web.config** file.

a) Open the **web.config** for the **CustomerManagerWeb** project and scroll down to the bottom.

b) Locate the **connectionStrings** section and the **add** element for the connection string named **CustomersDB**.

c) Inspect the value for the **initial catalog** property which has a value of **CustomerManagerWeb.Models.CustomerDB**.

```
      </parameters>
    </defaultConnectionFactory>
    <providers>
      <provider invariantName="System.Data.SqlClient" type="System.Data.Entity.SqlServer.SqlProviderServices, EntityFramework
    </providers>
  </entityFramework>
  <connectionStrings>
    <add name="CustomersDB"
         connectionString="data source=(LocalDb)\MSSQLLocalDB;initial catalog=CustomerManagerWeb.Models.CustomersDB;integrate
         providerName="System.Data.SqlClient" />
  </connectionStrings>
</configuration>
```

d) Modify the value for the **initial catalog** property to **CustomerDB_DEV**.

```
<add name="CustomersDB"
     connectionString="data source=(LocalDb)\MSSQLLocalDB;initial catalog=CustomerDB_DEV;
```

e) Save and close **web.config**.

28. Use the Package Manager Console to create a new SQL Server database from the Code First data model.

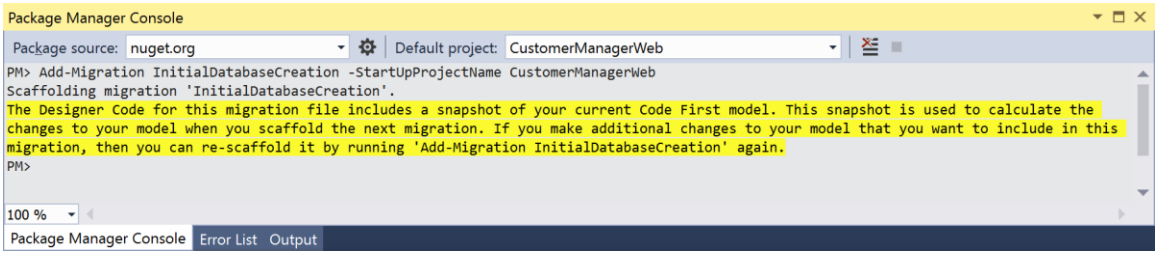
a) Navigate back to the **Package Manager Console** window.

b) Using the mouse, place the cursor to the right of the **PM>** prompt in the Package Manager Console.

c) Type the following command and press the **ENTER** key to execute it.

```
Add-Migration InitialDatabaseCreation -StartUpProjectName CustomerManagerWeb
```

d) Wait until the command completes its execution.

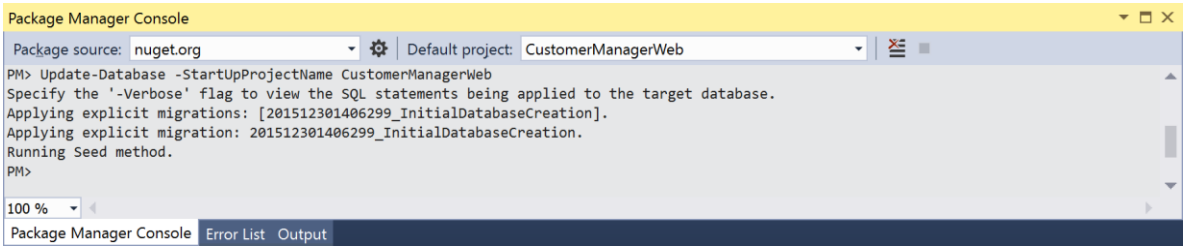


Now that you have created the first migration, you can now create a SQL Server database using the **Update-Database** command.

e) Type the following command and press the **ENTER** key to execute it.

```
Update-Database -StartUpProjectName CustomerManagerWeb
```

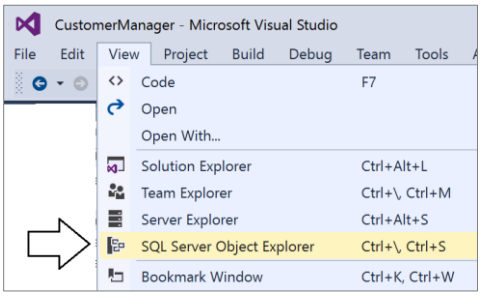
f) Wait until the command completes its execution.



At this point, you have gone through all the steps required to create a SQL Server database for your application. In the next step, you will use a Visual Studio utility named the **SQL Server Object Explorer** to inspect the new database and the data inside.

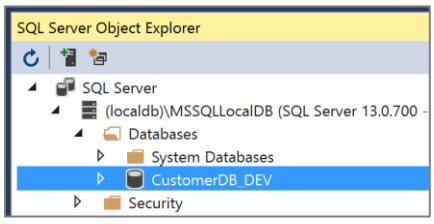
29. Verify that the SQL Server database named **CustomerDB_DEV** has been created.

a) Drop down **View** menu in Visual Studio and select the menu command for the **SQL Server Object Explorer**.



b) In the **SQL Server Object Explorer** pane, expand the **Databases** node for **(localdb)\MSSQLLocalDB**.

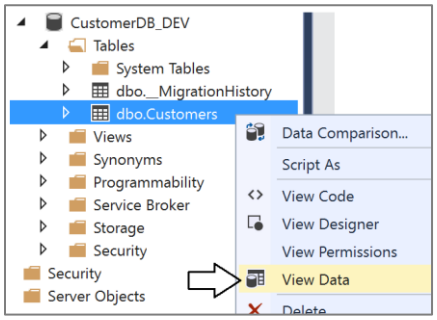
c) Verify that that the **Databases** contains the new database named **CustomerDB_DEV**.



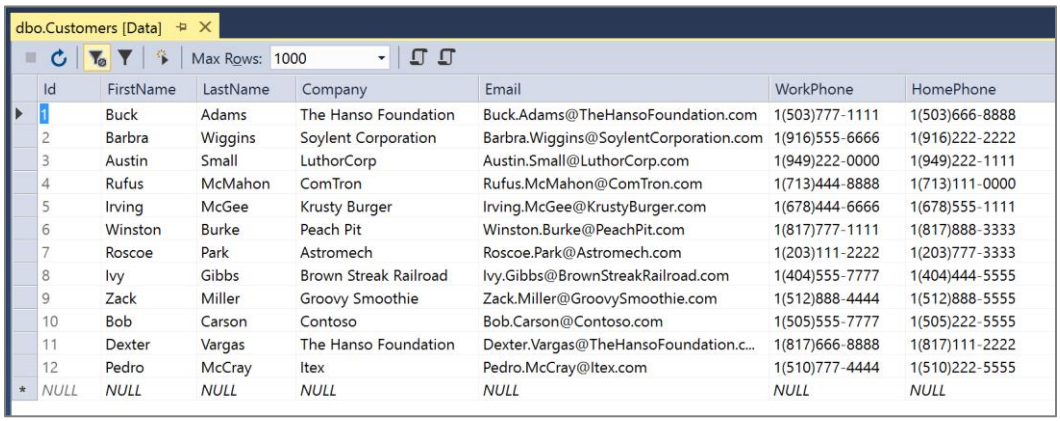
d) Expand the **CustomerDB_DEV** node and then expand the **Tables** node inside of it.

e) Verify that you see the **Customers** table inside the **Tables** node.

f) Right-click on the **Customers** table and click the **View Data** menu command.



g) At this point you should be able to see the sample records in the **Customers** table. These customer records were created from the code that you added to the **Seed** method earlier in this lab exercise.



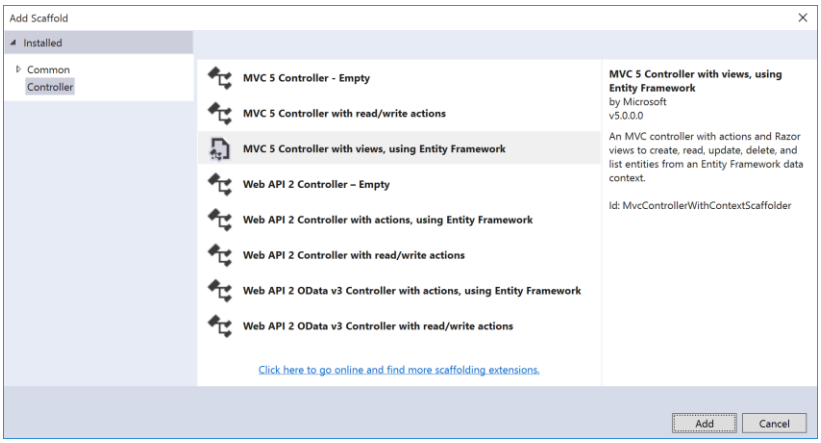
Id	FirstName	LastName	Company	Email	WorkPhone	HomePhone
1	Buck	Adams	The Hanso Foundation	Buck.Adams@TheHansoFoundation.com	1(503)777-1111	1(503)666-8888
2	Barbra	Wiggins	Soylent Corporation	Barbra.Wiggins@SoylentCorporation.com	1(916)555-6666	1(916)222-2222
3	Austin	Small	LuthorCorp	Austin.Small@LuthorCorp.com	1(949)222-0000	1(949)222-1111
4	Rufus	McMahon	ComTron	Rufus.McMahon@ComTron.com	1(713)444-8888	1(713)111-0000
5	Irving	McGee	Krusty Burger	Irving.McGee@KrustyBurger.com	1(678)444-6666	1(678)555-1111
6	Winston	Burke	Peach Pit	Winston.Burke@PeachPit.com	1(817)777-1111	1(817)888-3333
7	Roscoe	Park	Astromech	Roscoe.Park@Astromech.com	1(203)111-2222	1(203)777-3333
8	Ivy	Gibbs	Brown Streak Railroad	Ivy.Gibbs@BrownStreakRailroad.com	1(404)555-7777	1(404)444-5555
9	Zack	Miller	Groovy Smoothie	Zack.Miller@GroovySmoothie.com	1(512)888-4444	1(512)888-5555
10	Bob	Carson	Contoso	Bob.Carson@Contoso.com	1(505)555-7777	1(505)222-5555
11	Dexter	Vargas	The Hanso Foundation	Dexter.Vargas@TheHansoFoundation.c...	1(817)666-8888	1(817)111-2222
12	Pedro	McCray	Itex	Pedro.McCray@Itex.com	1(510)777-4444	1(510)222-5555
NULL	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*

h) Close the window which displays the data from the Customers table.

Exercise 4: Creating a Strongly-typed Controller using the MVC Framework

In this exercise you will complete your work on the Customer Manager add-in by creating a strongly-typed controller class which provides users with the ability to view, create, update and delete customer records.

30. Create a strongly-typed controller class using the Entity Framework data model named **CustomersDB**.
    a) In the **Solution Explorer**, expand the **CustomerManagerMVCWeb** project node.
    b) Right-click the **Controllers** folder and select **Add > Controller…** to display the **Add Scaffold** dialog.
    c) In the **Add Scaffold** dialog, select **MVC 5 Controller with views, using Entity Framework**
    d) Click the **Add** button to display the **Add Controller** dialog.



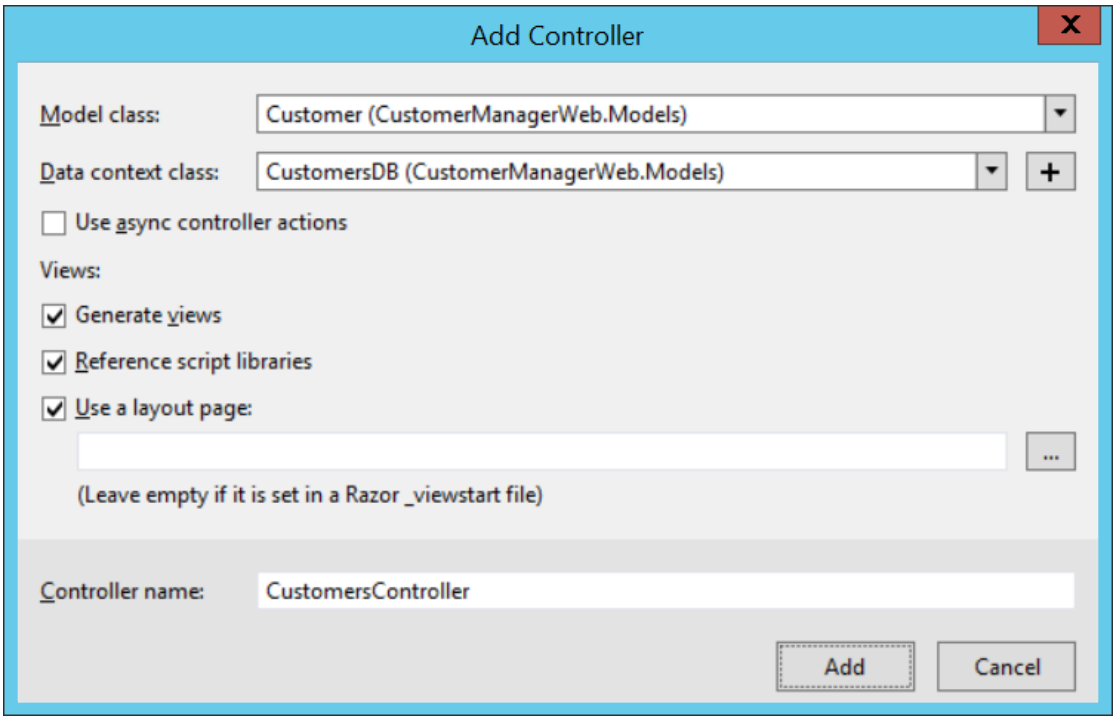
e) You should now be prompted by the **Add Controller** dialog,

f) Set the **Model class** to **Customer (CustomerManagerWeb.Models)**.

g) Set the **Data context class** to **CustomersDB (CustomerManagerWeb.Models)**.

h) Leave the **Controller name** setting with its default value of **CustomersController**.

i) Leave all other settings in the dialog with their default values and click **Add**.



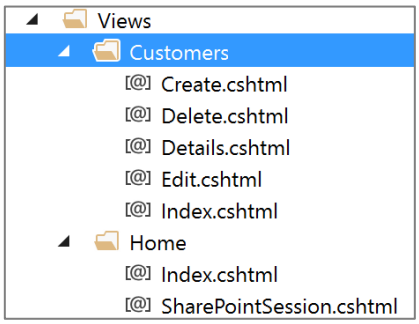
j) When you add the controller named **CustomersController**, Visual Studio creates a file named **CustomersController.cs** and opens this C# source file a Code View window.

k) Examine the **CustomersController** class that Visual Studio has generated for you. You will see that there are many action methods that have been added and implemented by Visual Studio.

Please note that you do not need to modify anything in the **CustomersController** class. It will work with just the code that has been generated by Visual Studio. However, as mentioned above, you should take a little time reviewing the code in each of the action methods in the **CustomersController** class to get an idea of how the code is written.

l) When you are done, close **CustomersController.cs** without saving any changes.

31. Inspect the views that have been generated for the **CustomersController.**
    a) In Solution Explorer, expand the project node for the **CustomerManagerWeb** project.
    b) Expand the top-level **Views** folder and then expand the child folder named **Customers**.
    c) Within the **Customers** folder, you should be able to see that five views that have been created for the **Customers** controller.



    d) Double-click on the razor view file named **Index.cshtml** to open it in a Code View window.
    e) As you can see, there is quite a bit of razor code that has been generated to implement this razor view.
    f) Inside **Index.cshtml**, remove the code that updates the **ViewBag.Title** property and modify the text inside the **h2** element to display “Customers List”.

```
@model IEnumerable<CustomerManagerWeb.Models.Customer>

<h2>Customers List</h2>
```

    g) Save and close **Index.cshtml**.
    h) Take a minute to quickly review the code in each of the other views for the **Customers** controller. There is no need to update any of the other views. However, it will be helpful if you have a basic understanding of how these views are implemented.
    i) When you are done, close any of the open razor view files that are still open.

32. Modify the navigation links in the shared view to incorporate the **CustomersController** in the application’s navigation menu.
    a) Locate the **_Layout.cshtml** file in the **Shared** folder inside the **Views** folder.
    b) Double-click on **_Layout.cshtml** to open the view file in a Code View window.
    c) Locate the **ul** element which contains action links for **Index** and **SharePointSession** action methods of the **Home** controller.

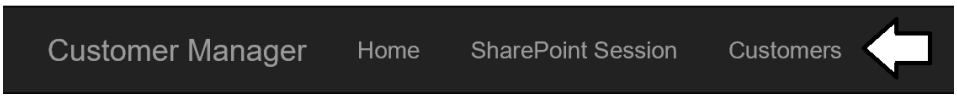
```
<ul class="nav navbar-nav">
  <li>@Html.ActionLink("Home", "Index", "Home")</li>
  <li>@Html.ActionLink("SharePoint Session", "SharePointSession", "Home")</li>
</ul>
```

    d) Add a third action link for the **Index** action method of the **Customers** controller as shown in the following code.

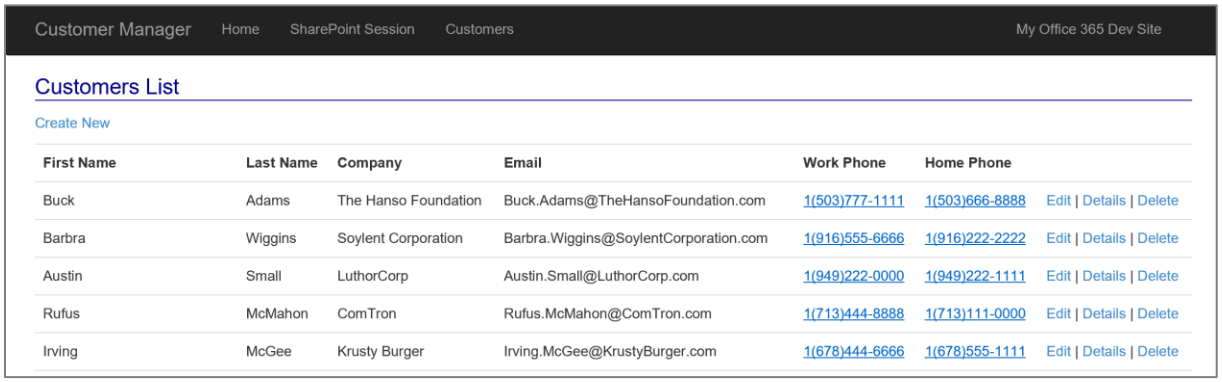
```
<ul class="nav navbar-nav">
  <li>@Html.ActionLink("Home", "Index", "Home")</li>
  <li>@Html.ActionLink("SharePoint Session", "SharePointSession", "Home")</li>
  <li>@Html.ActionLink("Customers", "Index", "Customers")</li>
</ul>
```

    e) When you are done, save your changes and close **_Layouts.cshtml**.

33. Test your work by running the Customer Manager add-in in the Visual Studio debugger.
    a) Press **F5** to begin a Visual Studio debugging session for the add-in.
    b) Once the add-in has been installed, you should be redirected to the add-in start page.
    c) Verify that your add-in is showing the same links as before along with the new link for **Customers**.



d) Click on the link for **Customers**. You should see a list of customers that matches the screenshot below.



First Name	Last Name	Company	Email	Work Phone	Home Phone	
Buck	Adams	The Hanso Foundation	Buck.Adams@TheHansoFoundation.com	1(503)777-1111	1(503)666-8888	Edit | Details | Delete
Barbra	Wiggins	Soylent Corporation	Barbra.Wiggins@SoylentCorporation.com	1(916)555-6666	1(916)222-2222	Edit | Details | Delete
Austin	Small	LuthorCorp	Austin.Small@LuthorCorp.com	1(949)222-0000	1(949)222-1111	Edit | Details | Delete
Rufus	McMahon	ComTron	Rufus.McMahon@ComTron.com	1(713)444-8888	1(713)111-0000	Edit | Details | Delete
Irving	McGee	Krusty Burger	Irving.McGee@KrustyBurger.com	1(678)444-6666	1(678)555-1111	Edit | Details | Delete

e) Try the following operations to better understand the user interface that has been created for you.
   i) Create a new customer.
   ii) View the details of an existing customer.
   iii) Edit an existing customer.
   iv) Delete an existing customer.

f) When you have finished your testing, close the browser to stop the current debugging session and return to Visual Studio.

You have now completed this lab successfully.