

ASP.Net MVC 5.0

Lab Book

Document Revision History

Date	Revision No.	Author	Summary of Changes

Table of Contents

Document Revision History	2
Table of Contents	3
Getting Started	4
Overview	4
Setup Checklist for ASP.Net MVC 5	4
Instructions	4
Lab 1. Creating a new Web Forms Project	5
Lab 2. Creating an MVC Controller Using Scaffolding	13

Getting Started

Overview

This lab book is a guided tour for learning **ASP.Net MVC 5**. It comprises solved examples and 'To Do' assignments. Follow the steps provided in the solved examples and work out the given 'To Do' assignments.

Setup Checklist for ASP.Net MVC 5

For the lab to work, following are the system requirements:

Minimum System Requirements

- Networked PCs with access to SQL Server 2012 and Minimum 256 MB RAM, 600 MB Hard Disk.
- OS: Windows XP with SP2 or Windows 2003 or Windows Vista
- .NET 4.5.1 Framework Installed
- Visual Studio.NET 2013 with .NET 4.5 extensions.
- MS SQL Server 2012

Instructions

- Create a directory by your name in drive <drive>. In this directory, create a subdirectory aspnet_mvc5_assgn. For each lab exercise create a directory as lab <lab number>.
- You may also look up the on-line help provided in the MSDN library.

Lab 1. Creating a new Web Forms Project

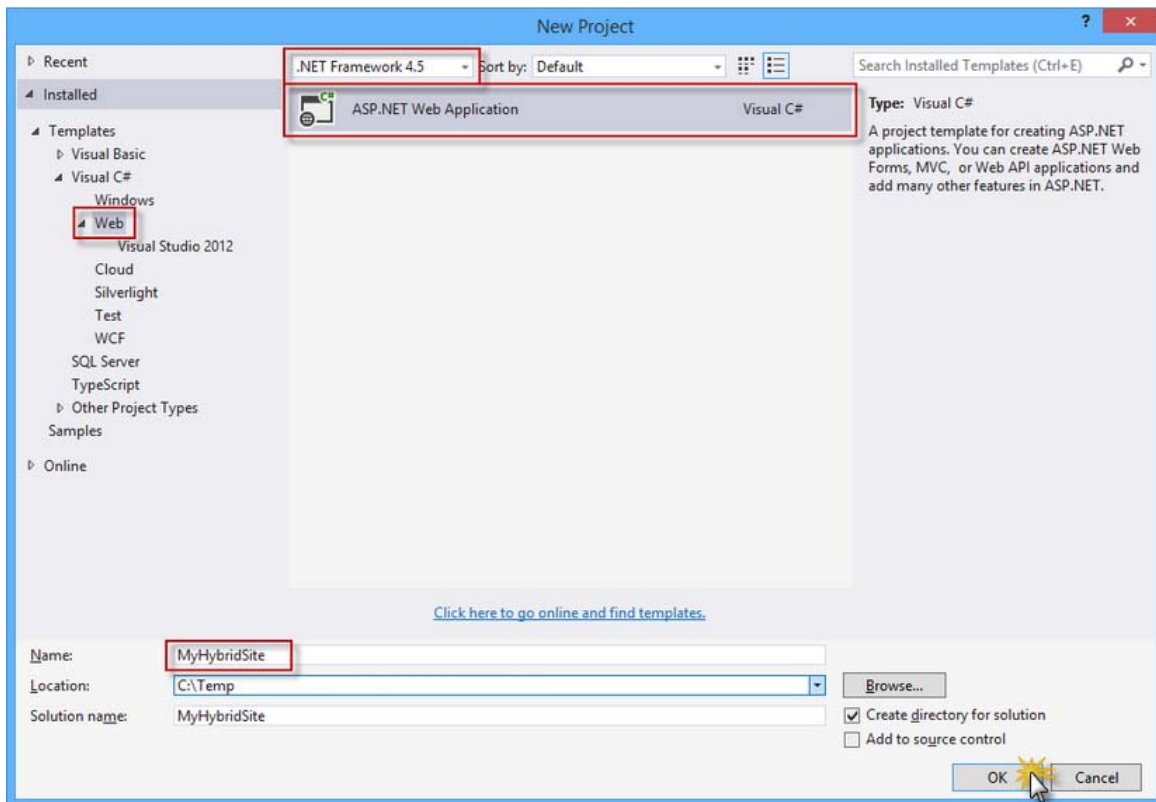
Goals	<ul style="list-style-type: none">• To Understand One ASP.Net unified project experience which allows us to easily integrate Web Forms, MVC and Web API components in the same application
Time	20 minutes

Task 1: Creating a new site using One ASP.Net experience

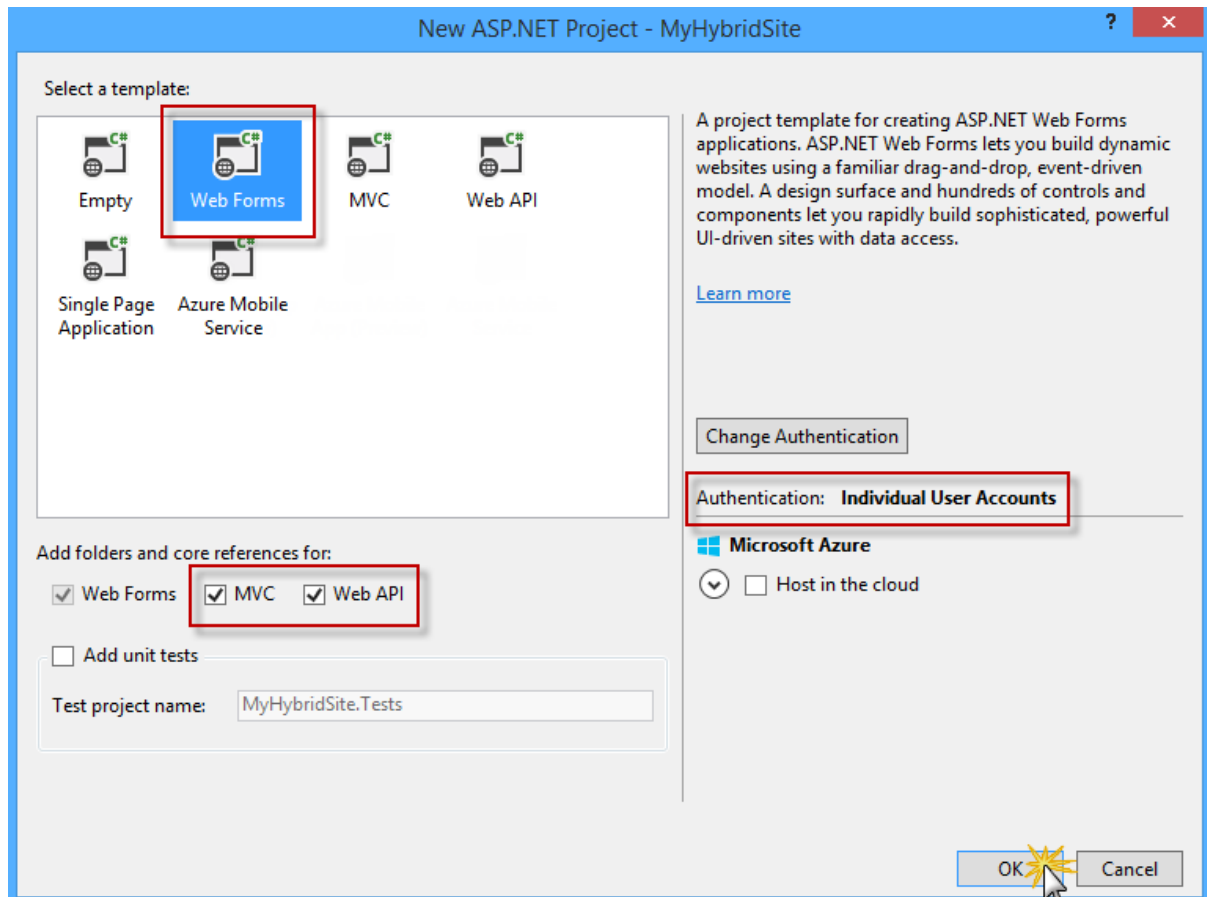
In this task you will start creating a new Web site in Visual Studio based on the **One ASP.NET** project type. **One ASP.NET** unifies all ASP.NET technologies and gives you the option to mix and match them as desired.

Step 1: Open Visual Studio 2013 and select File|New Project to start a new Solution.

Step 2: In the New Project dialog box, select ASP.Net Web Application under the Visual C# | Web tab and make sure .Net Framework 4.5 is selected. Name the project MyHybridSite, choose a location and click ok.



Step 3: In the New ASP.Net Project, select the Web Forms template and select the MVC and Web API options. Also make sure the Authentication option is set to Individual User Accounts. Click Ok to continue.



Step 4: You can explore the structure of the generated solution.

- i. **Account:** This folder contains the Web Form pages to register, log in to and manage the application's user accounts. This folder is added when the **Individual User Accounts** authentication option is selected during the configuration of the Web Forms project template.
- ii. **Models:** This folder will contain the classes that represent your application data.
- iii. **Controllers and Views:** These folders are required for the **ASP.NET MVC** and **ASP.NET Web API** components. You will explore the MVC and Web API technologies in the next exercises.

- iv. The **Default.aspx**, **Contact.aspx** and **About.aspx** files are pre-defined Web Form pages that you can use as starting points to build the pages specific to your application. The programming logic of those files resides in a separate file referred to as the "code-behind" file, which has an ".aspx.vb" or ".aspx.cs" extension (depending on the language used). The code-behind logic runs on the server and dynamically produces the HTML output for your page.
- v. The **Site.Master** and **Site.Mobile.Master** pages define the look and feel and the standard behavior of all the pages in the application.

Step 5: Double click the Default.aspx file to explore the content of the page



```
<%@ Page Title="Home Page" Language="C#" MasterPageFile="~/Site.Master"
AutoEventWireup="true" CodeBehind="Default.aspx.cs" Inherits="MyHybridSite._Default" %>

<asp:Content ID="BodyContent" ContentPlaceHolderID="MainContent" runat="server">

    <div class="jumbotron">
        <h1>ASP.NET</h1>
        <p class="lead">ASP.NET is a free web framework for building great Web sites and Web application
        <p><a href="http://www.asp.net" class="btn btn-primary btn-large">Learn more &raquo;</a></p>
    </div>

    <div class="row">
        <div class="col-md-4">
            <h2>Getting started</h2>
            <p>
                ASP.NET Web Forms lets you build dynamic websites using a familiar drag-and-drop, event-
                A design surface and hundreds of controls and components let you rapidly build sophisticated
            </p>
            <p>
                <a class="btn btn-default" href="http://go.microsoft.com/fwlink/?LinkId=301948">Learn mo
            </p>
        </div>
        <div class="col-md-4">
            <h2>Get more libraries</h2>
```


Step 6: Expand the **App_Start** folder and notice the **WebApiConfig.cs** file. Visual Studio included that file in the generated solution because you included Web API when configuring your project with the One ASP.NET template.

Step 7: Open the **WebApiConfig.cs** file. In the *WebApiConfig* class you will find the configuration associated with Web API, which maps HTTP routes to **Web API controllers**.

```
public static void Register(HttpConfiguration config)
{
    // Web API configuration and services

    // Web API routes
    config.MapHttpAttributeRoutes();

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

Step 8: Open the **RouteConfig.cs** file. Inside the *RegisterRoutes* method you will find the configuration associated with MVC, which maps HTTP routes to **MVC controllers**.

```
public static void RegisterRoutes(RouteCollection routes)
{
    var settings = new FriendlyUrlSettings();
    settings.AutoRedirectMode = RedirectMode.Permanent;
    routes.EnableFriendlyUrls(settings);

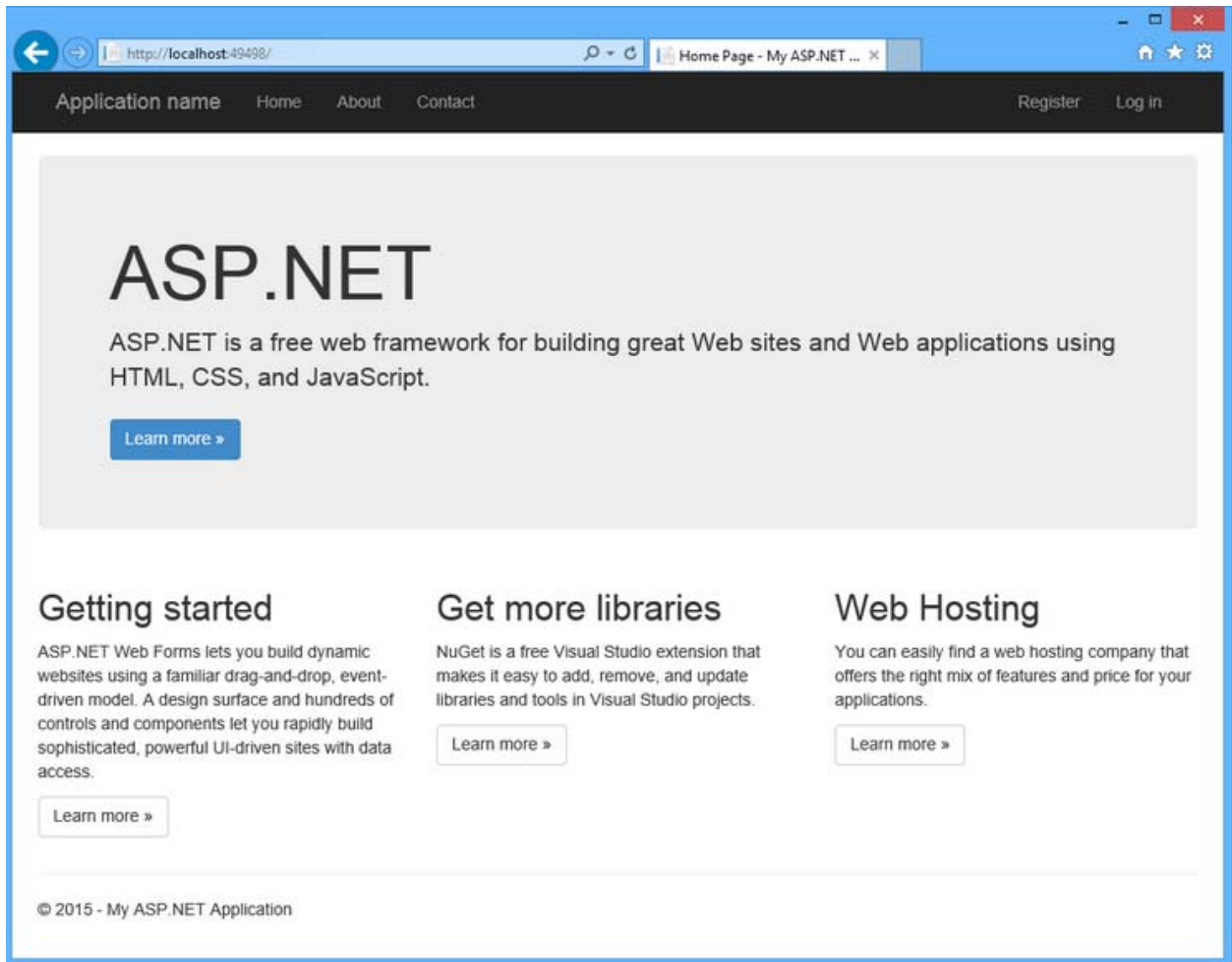
    routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

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

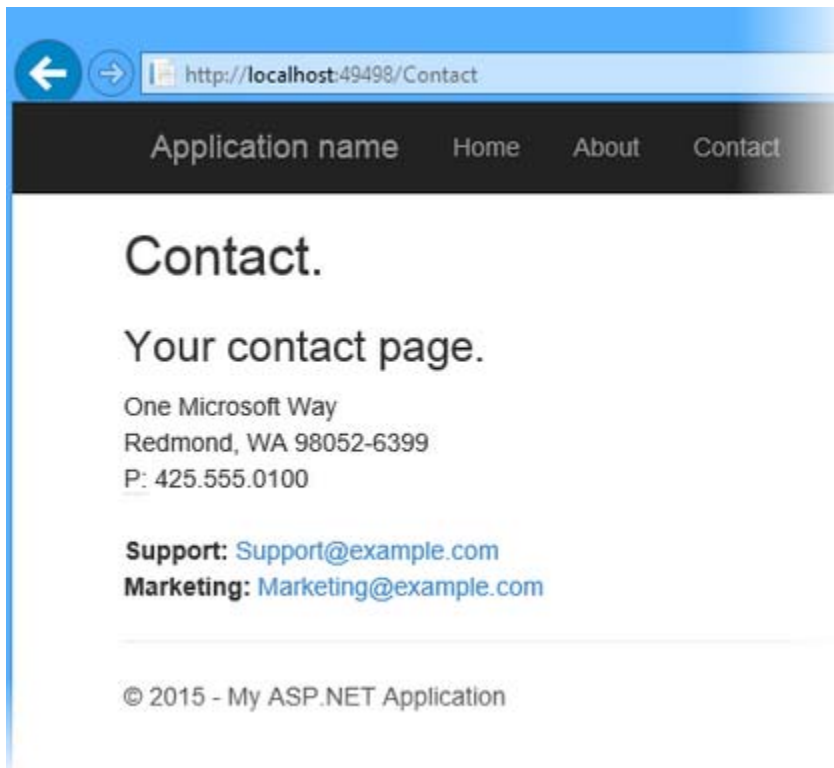
Task 2: Running the Solution

In this task you will run the generated solution, explore the app and some of its features, like URL rewriting and built-in authentication.

Step 1: To run the solution, press **F5** or click the **Start** button located on the toolbar. The application home page should open in the browser.



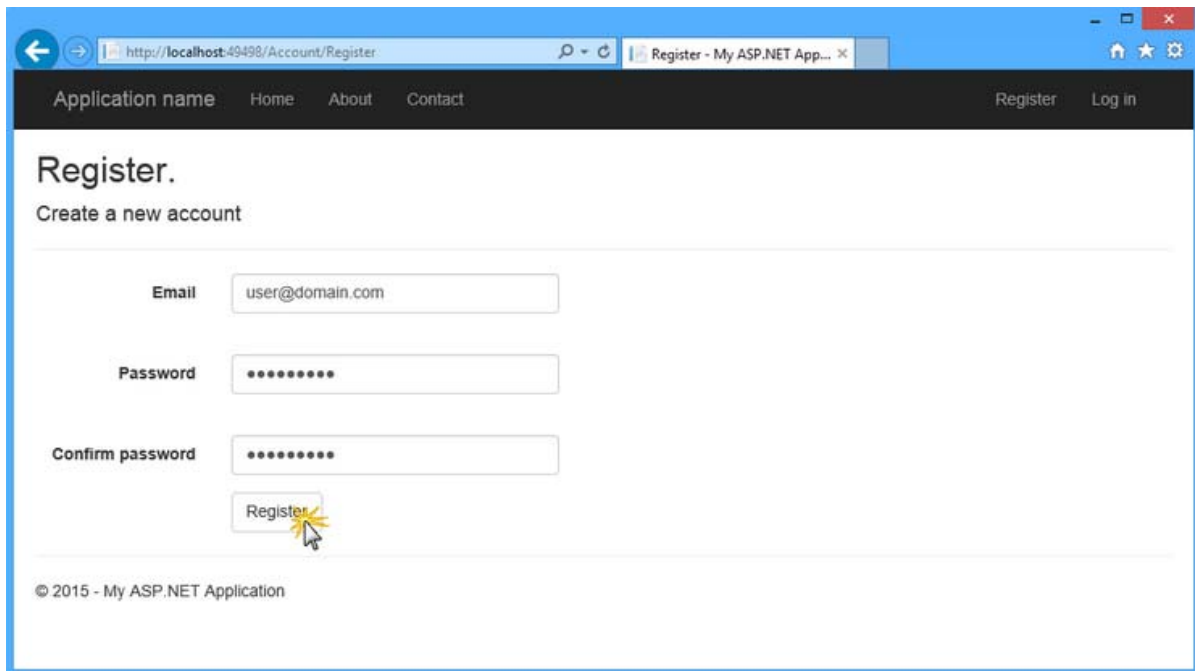
Step 2: Verify that the Web Forms pages are being invoked. To do this, append **/contact.aspx** to the URL in the address bar and press **Enter**.



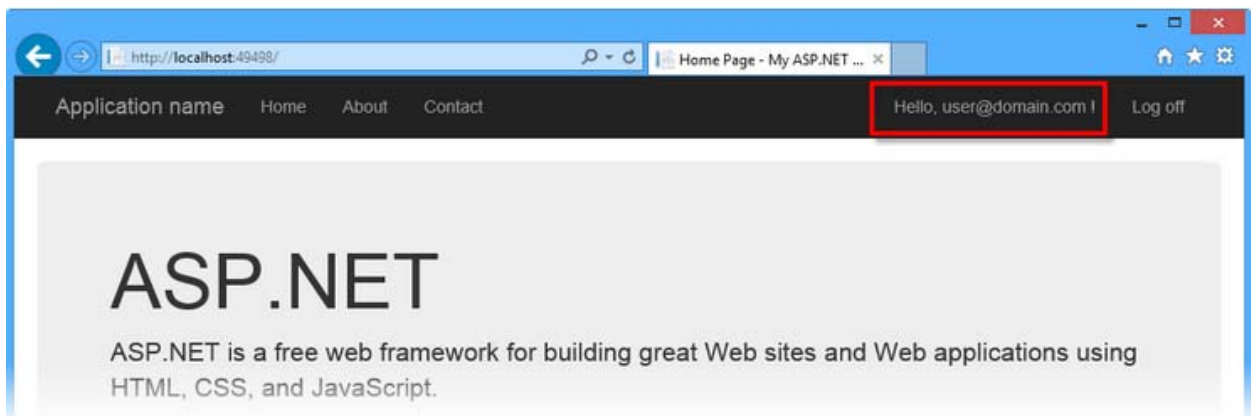
Step 3: You will now explore the authentication flow integrated into the application. To do this, click **Register** in the upper-right corner of the page.



Step 4: In the **Register** page, enter a **User name** and **Password**, and then click **Register**.



Step 5: The application registers the new account, and the user is authenticated.



Step 6: Go back to Visual Studio and press **SHIFT + F5** to stop debugging.

Stretched Assignments:

Lab 2. Creating an MVC Controller Using Scaffolding

Goals	<ul style="list-style-type: none">• Explain what is Scaffolding• Understand and implement CRUD operations using scaffolding feature
Time	25 minutes

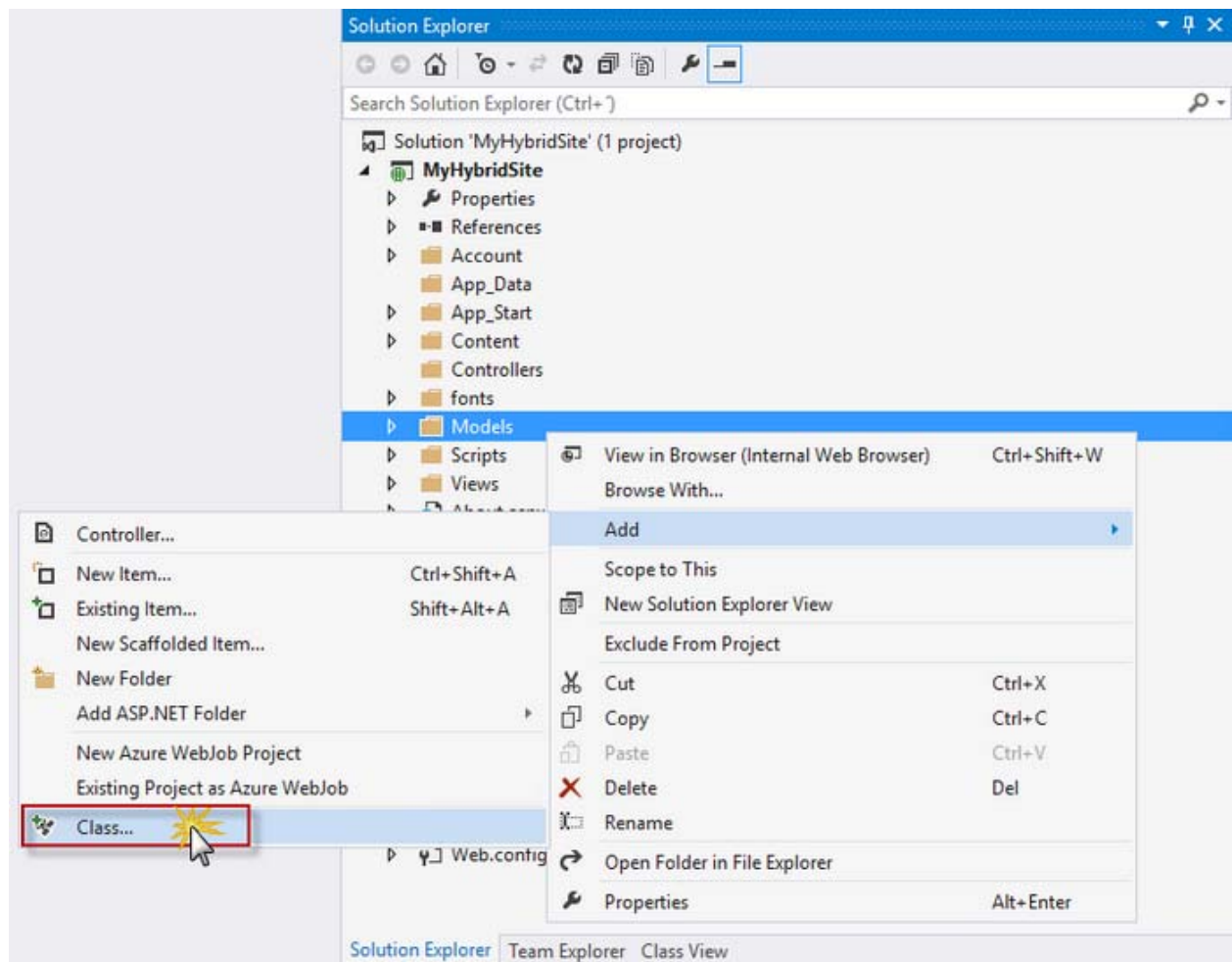
In this exercise you will take advantage of the ASP.NET Scaffolding framework provided by Visual Studio to create an ASP.NET MVC 5 controller with actions and Razor views to perform CRUD operations, without writing a single line of code. The scaffolding process will use Entity Framework Code First to generate the data context and the database schema in the SQL database.

Task 1: Creating a new Model

You will now define a **Person** class, which will be the model used by the scaffolding process to create the MVC controller and the views. You will start by creating a **Person** model class, and the CRUD operations in the controller will be automatically created using scaffolding features.

Step 1: Open **Visual Studio Community 2013** and the **MyHybridSite.sln** solution which was created previously in Lab 1.

Step 2: In **Solution Explorer**, right-click the **Models** folder of the **MyHybridSite** project and select **Add | Class....**



Step 3: In the **Add New Item** dialog box, name the file *Person.cs* and click **Add**.

Step 4: Replace the content of the **Person.cs** file with the following code. Press **CTRL + S** to save the changes

```
namespace MyHybridSite.Models
{
    public class Person
    {
        public int Id { get; set; }

        public string Name { get; set; }

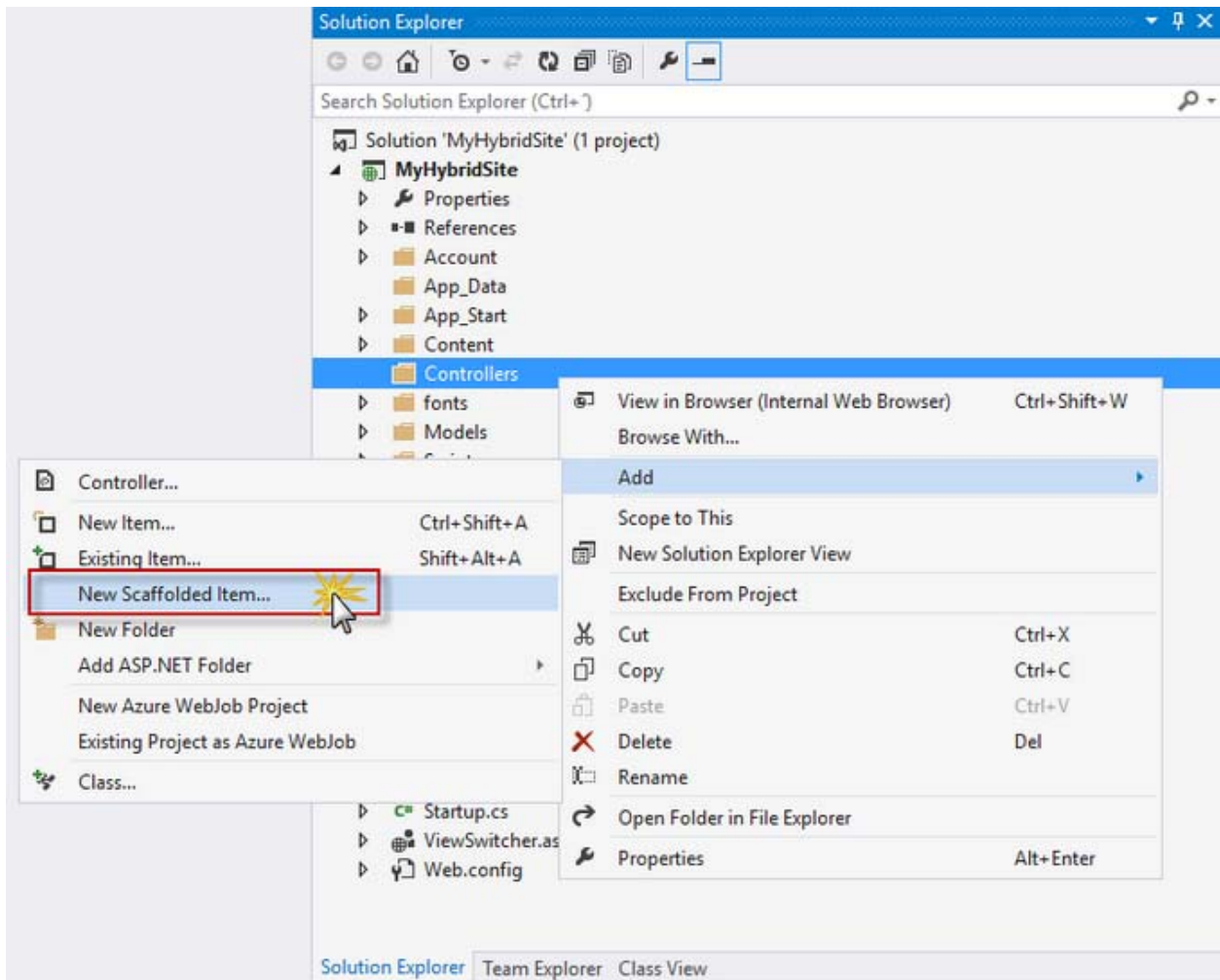
        public int Age { get; set; }
    }
}
```

Step 5: In **Solution Explorer**, right-click the **MyHybridSite** project and select **Build**, or press **CTRL + SHIFT + B** to build the project.

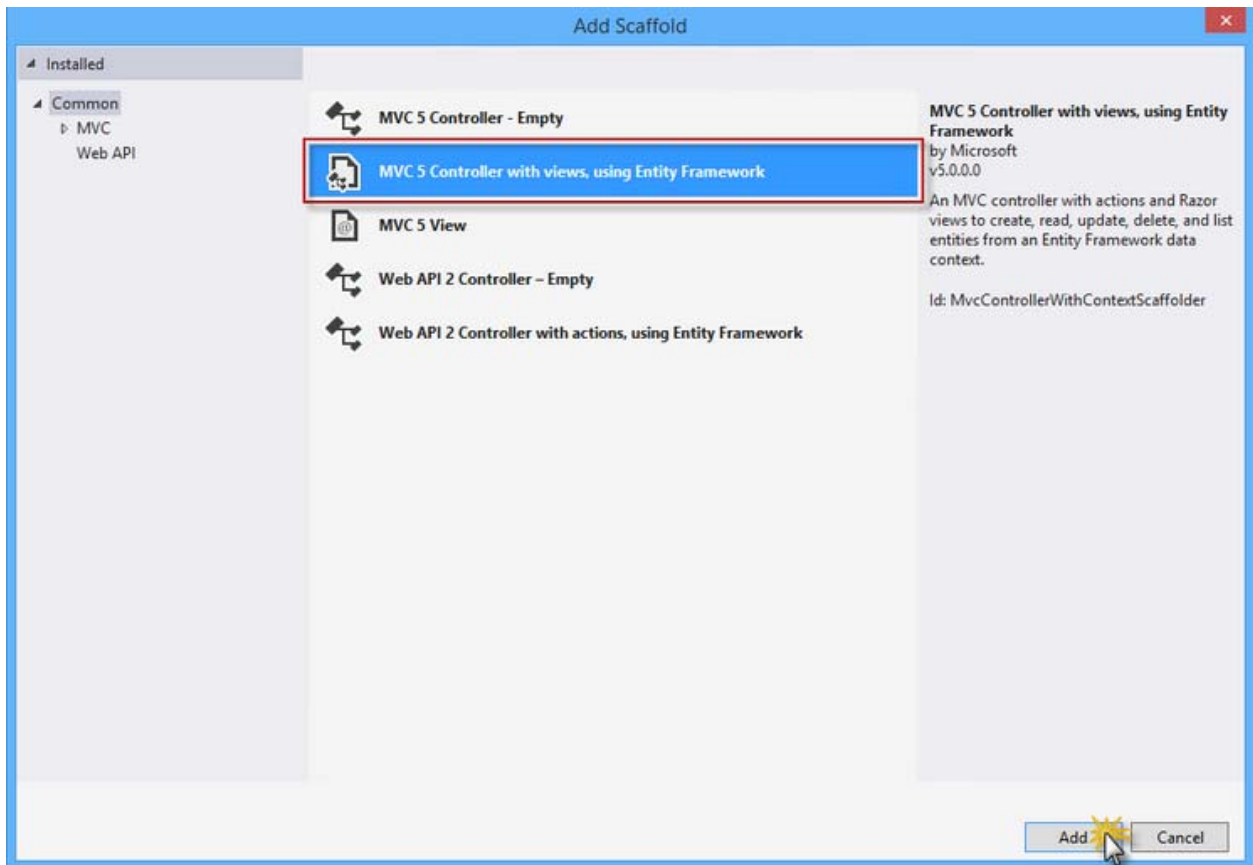
Task 2: Creating an MVC Controller

Now that the Person model is created, you will use ASP.NET MVC scaffolding with Entity Framework to create the CRUD controller actions and views for Person

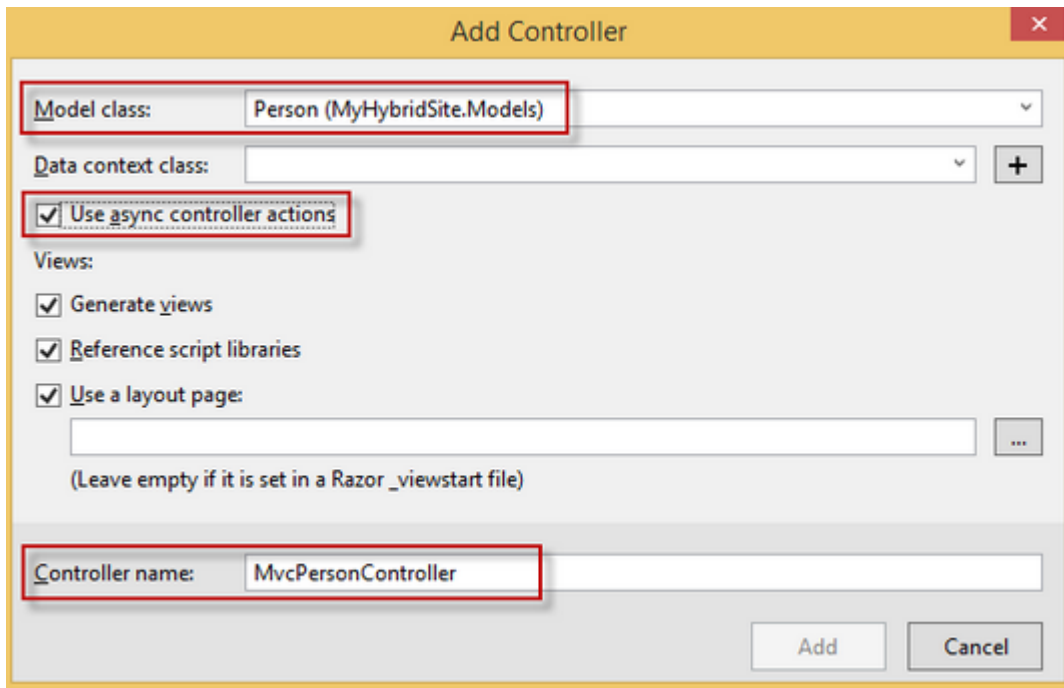
Step 1: In Solution Explorer, **right-click the Controllers folder of the MyHybridSite project and select Add | New Scaffolded Item....**



Step 2: In the Add Scaffold dialog box, select MVC 5 Controller with views, using Entity Framework and then click Add.



Step 3: Set *MvcPersonController* as the Controller name, **select the** Use async controller actions **option and select** Person (MyHybridSite.Models) **as the** Model class.

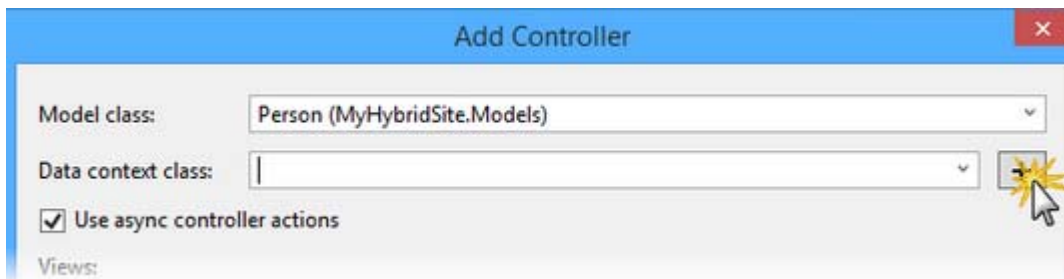


The 'Add Controller' dialog box is shown with the following settings:

- Model class:** Person (MyHybridSite.Models)
- Data context class:** (empty)
- ☒ **Use async controller actions**
- Views:**
 - ☒ Generate views
 - ☒ Reference script libraries
 - ☒ Use a layout page: (empty)
- Controller name:** MvcPersonController

Buttons: Add, Cancel

Step 4: Under Data context class, click New data context button.

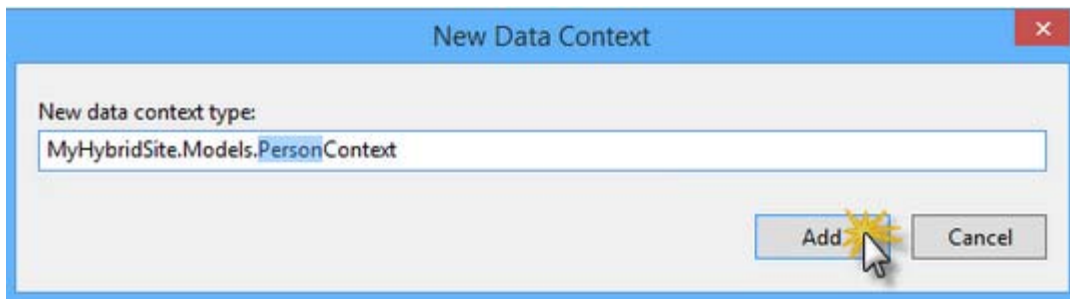


The 'Add Controller' dialog box is shown with the following settings:

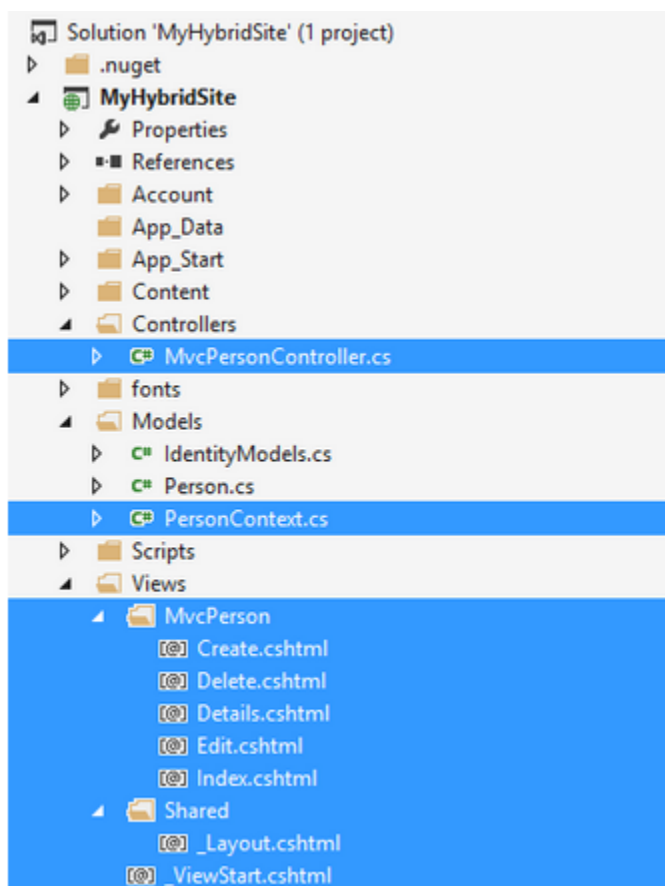
- Model class:** Person (MyHybridSite.Models)
- Data context class:** (empty)
- ☒ **Use async controller actions**
- Views:** (empty)

A yellow starburst icon and a mouse cursor are pointing to the '+' button next to the Data context class field.

Step 5: In the New Data Context dialog box, name the new data context PersonContext and click Add.



Step 6: Click Add to create the new controller for Person with scaffolding. Visual Studio will then generate the controller actions, the Person data context and the Razor views.



Step 7: Open the **MvcPersonController.cs** file in the **Controllers** folder. Notice that the CRUD action methods have been generated automatically.

```
...

// POST: /MvcPerson/Create
// To protect from overposting attacks, please enable the specific properties you want to bind to, for
// more details see http://go.microsoft.com/fwlink/?LinkId=317598.
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<ActionResult> Create([Bind(Include="Id,Name,Age")] Person person)
{
    if (ModelState.IsValid)
    {
        db.People.Add(person);
        await db.SaveChangesAsync();
        return RedirectToAction("Index");
    }

    return View(person);
}

// GET: /MvcPerson/Edit/5
public async Task<ActionResult> Edit(int? id)
{
    if (id == null)
    {
        return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
    }
    Person person = await db.People.FindAsync(id);
    if (person == null)
    {
        return HttpNotFound();
    }
    return View(person);
}

...
```

Note: By selecting the **Use async controller actions** check box from the scaffolding options in the previous steps, Visual Studio generates asynchronous action methods for all actions that involve access to the Person data context. It is recommended that you use asynchronous action methods for long-running, non-CPU bound requests to avoid blocking the Web server from performing work while the request is being processed.

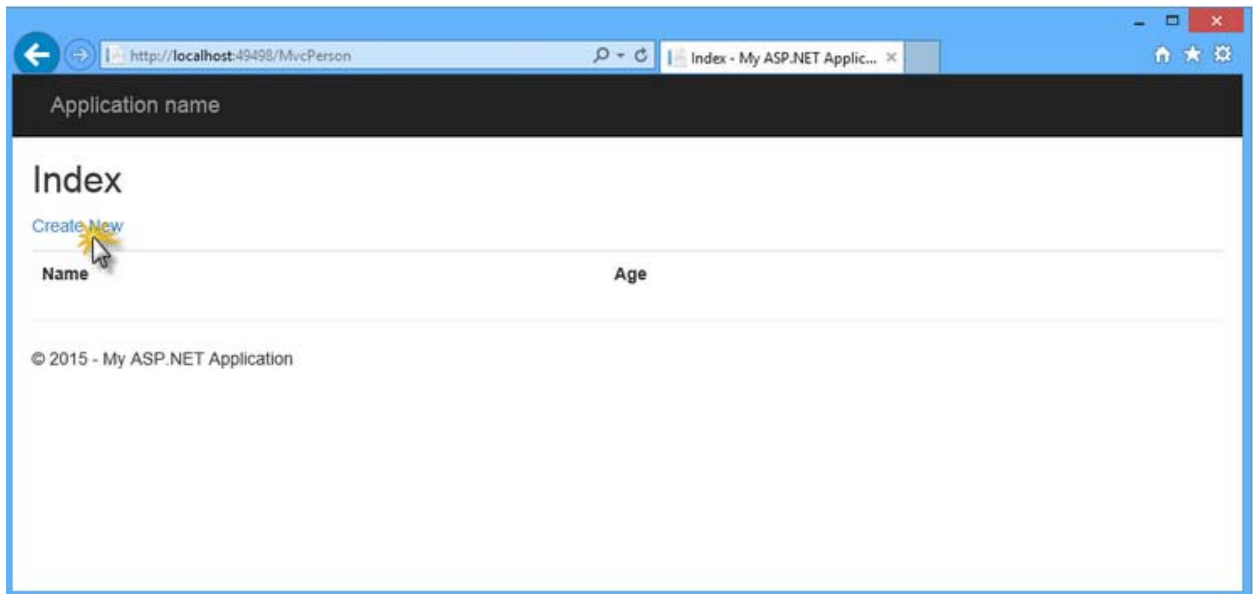
Task 3 – Running the Solution

In this task, you will run the solution again to verify that the views for **Person** are working as expected. You will add a new person to verify that it is successfully saved to the database.

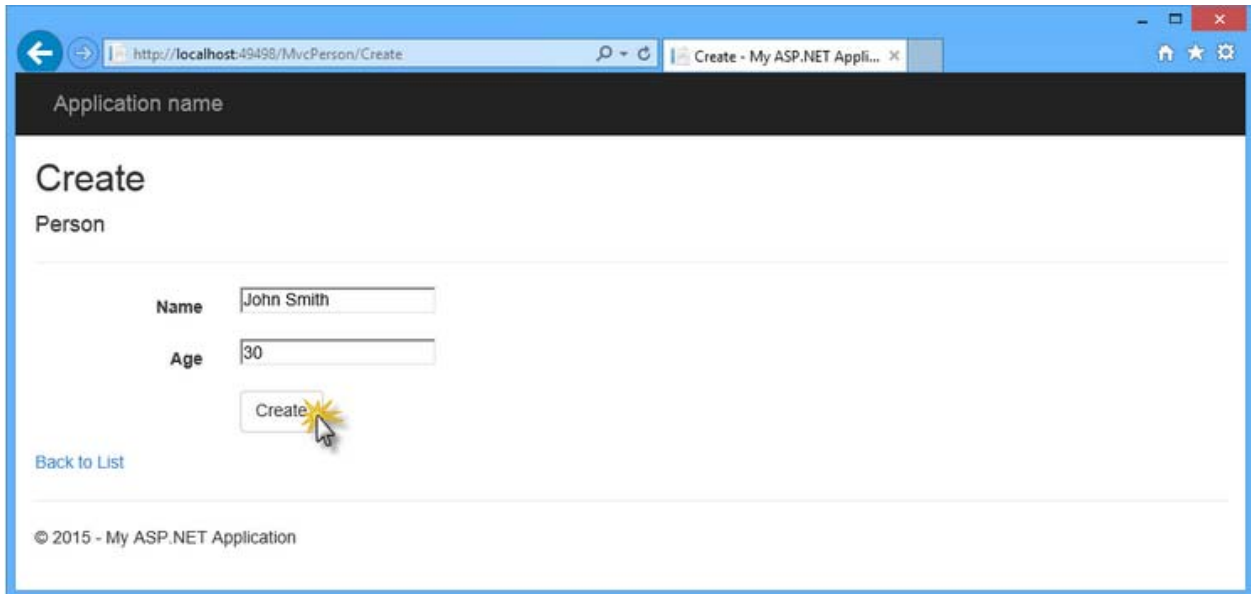
Step 1: Press **F5** to run the solution.

Step 2: Navigate to **/MvcPerson**. The scaffolded view that shows the list of people should appear

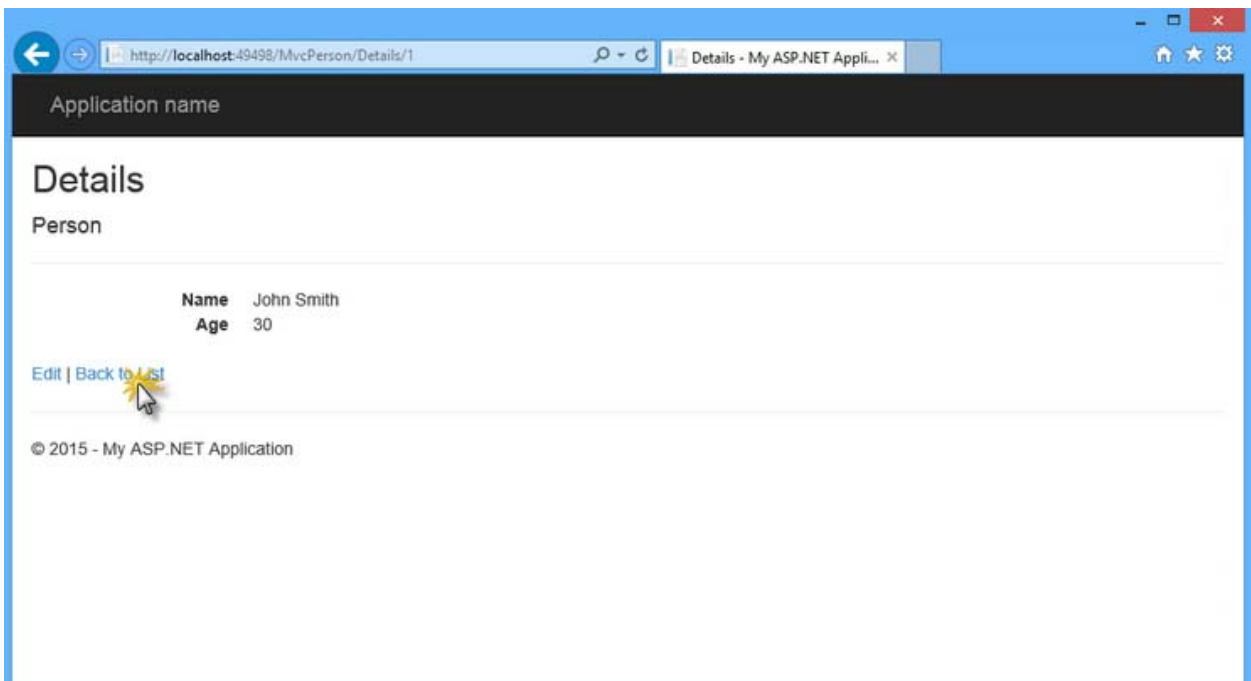
Step 3: Click **Create New** to add a new person.



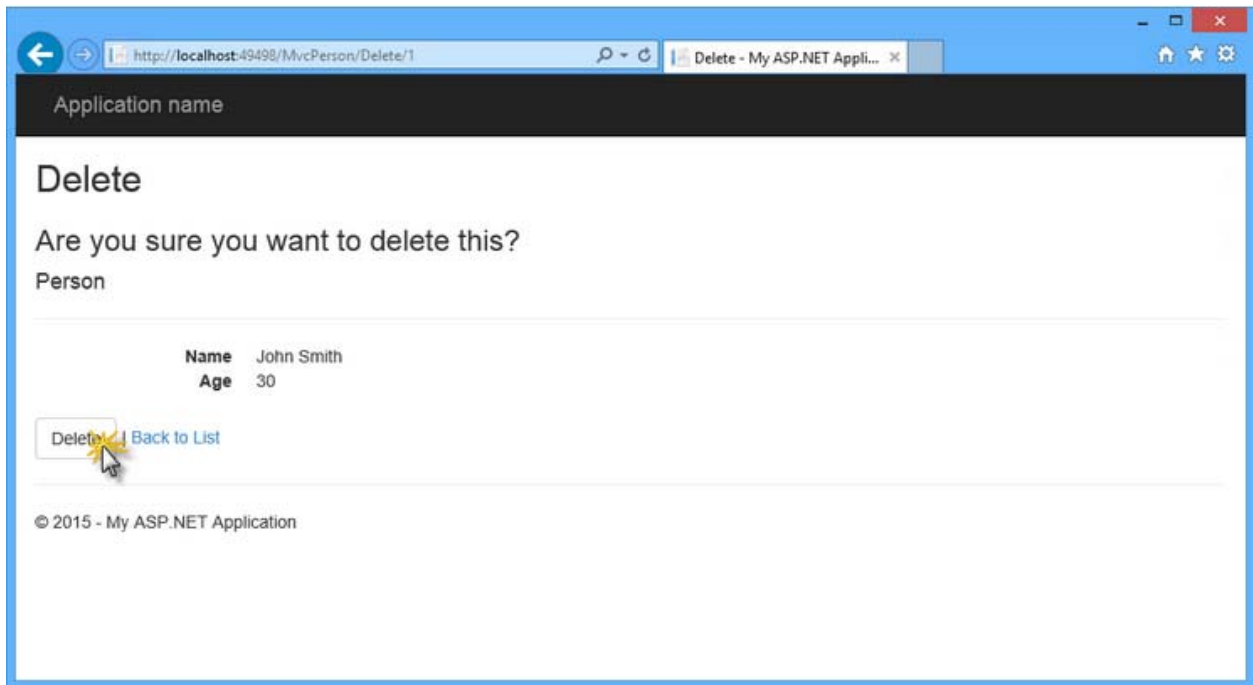
Step 4: In the **Create** view, provide a **Name** and an **Age** for the person, and click **Create**.



Step 5: The new person is added to the list. In the element list, click **Details** to display the person's details view. Then, in the **Details** view, click **Back to List** to go back to the list view.



Step 6: Click the **Delete** link to delete the person. In the **Delete** view, click **Delete** to confirm the operation.



Step 7: Go back to Visual Studio and press **SHIFT + F5** to stop debugging.