

EVALUATION COPY

Unauthorized reproduction or distribution is prohibited.

ASP.NET MVC USING C#

ASP.NET MVC Using C#

Rev. 4.8

Student Guide

Information in this document is subject to change without notice. Companies, names and data used in examples herein are fictitious unless otherwise noted. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of Object Innovations.

Product and company names mentioned herein are the trademarks or registered trademarks of their respective owners.



TM is a trademark of Object Innovations.

Authors: Robert Hurlbut and Robert J. Oberg

Copyright ©2017 Object Innovations Enterprises, LLC All rights reserved.

Object Innovations
877-558-7246
www.objectinnovations.com

Printed in the United States of America.

EVALUATION COPY

Unauthorized reproduction or distribution is prohibited

Table of Contents (Overview)

Chapter 1	Introduction to ASP.NET MVC
Chapter 2	Getting Started with ASP.NET MVC
Chapter 3	ASP.NET MVC Architecture
Chapter 4	The Model
Chapter 5	The Controller
Chapter 6	The View
Chapter 7	Routing
Chapter 8	Unit Testing
Chapter 9	Security
Chapter 10	Combining ASP.NET MVC and Web Forms
Chapter 11	ASP.NET Web API
Chapter 12	ASP.NET and Azure
Appendix A	Learning Resources
Appendix B	Deployment in IIS 7.5

Directory Structure

- **The course software installs to the root directory C:\OIC\MvcCs.**
 - Example programs for each chapter are in named subdirectories of chapter directories **Chap02**, **Chap03**, and so on.
 - The **Labs** directory contains one subdirectory for each lab, named after the lab number. Starter code is frequently supplied, and answers are provided in the chapter directories.
 - The **Demos** directory is provided for doing in-class demonstrations led by the instructor.
- **Data files install to the directory C:\OIC\Data.**

Table of Contents (Detailed)

Chapter 1: Introduction to ASP.NET MVC.....	1
Review of ASP.NET Web Forms	3
Advantages of ASP.NET Web Forms	4
Disadvantages of ASP.NET Web Forms	5
Model-View-Controller Pattern.....	6
ASP.NET MVC	7
Advantages of ASP.NET MVC	8
Disadvantages of ASP.NET MVC.....	9
Goals of ASP.NET MVC.....	10
Unit Testing	11
Summary	12
Chapter 2: Getting Started with ASP.NET MVC.....	13
An ASP.NET MVC 5 Testbed.....	15
Visual Studio ASP.NET MVC Demo.....	16
ASP.NET Documentation Page	18
Starter Application.....	19
Simple App with Controller Only.....	21
Action Methods and Routing	27
Action Method Return Type	28
Rendering a View	29
Creating a View in Visual Studio	30
The View Web Page	31
Dynamic Output.....	32
Razor View Engine	33
Embedded Scripts	34
Embedded Script Example.....	35
Using a Model with ViewBag	36
Controller Using Model and ViewBag	37
View Using Model and ViewBag	38
Using Model Directly	39
A View Using Model in Visual Studio	40
View Created by Visual Studio	41
Passing Parameters in Query String.....	42
Lab 2	43
Summary	44
Chapter 3: ASP.NET MVC Architecture.....	51
The Controller in ASP.NET MVC.....	53
The View in ASP.NET MVC	54
The Model in ASP.NET MVC.....	55
How MVC Works	56
Using Forms.....	57

HTML Helper Functions	58
Handling Form Submission	59
Model Binding	60
Greet View	61
Input Validation	62
Nullable Type	63
Checking Model Validity.....	64
Validation Summary	65
Lab 3	66
Summary.....	67
Chapter 4: The Model	75
Microsoft Technologies for the Model	77
SmallPub Database	78
ADO.NET	80
Language Integrated Query (LINQ)	81
ADO.NET Entity Framework.....	82
EDM Example	83
SmallPub Tables	86
SmallPub Entity Data Model	87
Entity Data Model Concepts.....	88
Querying the EDM.....	89
Class Diagram.....	90
Context Class	91
List of Categories.....	92
List of Books.....	94
LINQ to Entities Demo.....	96
A LINQ Query	97
IntelliSense.....	98
Controller	99
Using a Parameter	100
Modifying a Data Source.....	101
LINQ to Entities Insert Example	103
LINQ to Entities Delete Example	104
Entity Framework in a Class Library.....	105
Data Access Class Library	106
Client Code	107
Configuration in Entity Framework 6.....	108
ASP.NET MVC Database Clients	109
ASP.NET MVC CRUD Demo.....	110
Lab 4	115
NuGet Package Manager	116
Entity Framework 6 via NuGet.....	117
Summary	118

Chapter 5: The Controller	125
IController Interface	127
A Low Level Controller.....	128
Displaying Context	129
Add and Subtract	130
Controller Base Class.....	131
Controller Base Class.....	132
Action Methods.....	133
Action Method Example	134
HomeController	135
MathController.....	136
Invoking MathController	137
Receiving Input.....	138
Binding Example	139
Non-Nullable Parameters.....	140
Nullable Parameters	141
Using a Model.....	142
Action Results.....	143
Action Result Example	144
Output Demo.....	145
JavaScript Object Notation	148
Action Method Attributes	149
HTML Output	150
Filters	151
Asynchronous Controllers	153
Lab 5	154
Summary	155
Chapter 6: The View.....	163
View Responsibility.....	165
A Program without a View	166
A Program with a View	167
View Page	168
Passing Data to the View	169
Dynamic and ExpandoObject	170
Passing Lists to the View.....	171
HTML Helper Methods	172
Link-Building Helpers	173
Form Helpers	174
Html Helper Example	175
Validation Helpers	177
Templated Helpers	178
Validation in Model	180
Validation in Controller.....	181
ValidationMessage Helper.....	182
Running the Example.....	183

Lab 6	184
Summary	185
Chapter 7: Routing	193
ASP.NET Routing.....	195
Routing in ASP.NET MVC	196
RouteCollectionExtensions Class	197
Simple Route Example	198
Default Values for URL Parameters	200
Using a Default Route.....	201
Home Controller	202
Assigning Parameter Values	203
Controller Code.....	204
View Code	205
Running the Example.....	206
Properties of Routes	207
UrlParameter.Optional	208
Matching URLs to Route	209
Demo: Route Order.....	210
Install Route Debugging Utility.....	214
Multiple Matches	216
Fixing the Bug	217
Debugging Routes: Summary	218
Areas	219
Demo: Areas	220
Summary	223
Chapter 8: Unit Testing.....	225
Test-Driven Development.....	227
Functional Tests.....	228
Unit Tests	229
Test Automation.....	230
Refactoring.....	231
Regression Testing.....	232
Unit Testing in ASP.NET MVC	233
Creating a Test Project.....	234
Adding a Unit Test Project	235
MVC Unit Test Example	236
A Test Class Library	237
The Model.....	238
Testing the Model	239
Structure of Unit Tests	240
Assertions.....	241
Visual Studio Unit Framework	242
Assert Class.....	243
Assert.AreEqual()	244

More Assert Methods.....	245
CollectionAssert Class	246
StringAssert Class	247
Test Case	248
Test Methods.....	249
Test Class	250
Test Runner.....	251
More Tests	252
Ignoring Tests	253
Fixing the Bug	254
Testing Controllers	255
Lab 8A	256
Classes with External Dependencies	257
Dependency Injection	258
Mocking Frameworks	259
Using Moq	260
Installing Moq Using NuGet.....	261
MvcMortgage Example	262
Models	263
Controller	264
Index View.....	265
Monthly Payment View.....	266
Unit Tests	267
Inversion of Control (IoC) Containers.....	268
Lab 8B.....	269
Summary	270
Chapter 9: Security.....	283
Web Security.....	285
Input Forgery	286
Cross-Site Scripting	288
Cross-Site Scripting Example.....	290
XSS Example	291
Entering JavaScript.....	293
The Attack.....	294
What Allowed the Attack.....	295
Blocking the Attack	296
Using Razor	297
Session Hijacking	298
Cross-Site Request Forgery	300
XSRF Example	302
Controller and Model Code	304
View Code	305
Attacker Code	306
The Defense	307
SQL Injection.....	308

Using the MVC Framework Securely.....	312
Authorize Attribute	313
ChildActionOnly Attribute	314
RequireHttps Attribute.....	315
ValidateInput Attribute	316
Summary	317
Chapter 10: Combining ASP.NET MVC and Web Forms.....	319
Using Web Forms in an MVC Application	321
Linking and Redirecting from Web Forms Pages to MVC Actions	322
Default.aspx	323
Web Forms in MVC Example	324
Sharing Data between ASP.NET MVC and ASP.NET Web Forms	326
Using Web Form Controls in MVC Views.....	327
Using MVC in a Web Forms Application	328
Steps to Create Hybrid MVC - Web Forms Application.....	329
Web Forms MVC Interop in VS 2017	331
Summary	332
Chapter 11: ASP.NET Web API.....	333
ASP.NET Web API.....	335
REST	336
Representation, State and Transfer	337
Collections and Elements.....	338
Web API Demo.....	339
Specifying a Start Page	343
Implementing PUT Verb.....	344
Using Fiddler	345
Composing a Request	347
ASP.NET MVC and Web API.....	349
String API Demo.....	350
Route Registration	352
Lab 11A	355
HTTP Response Codes	356
POST Response Code	357
HttpResponseException.....	358
Web API Clients	359
HttpClient.....	360
Initializing HttpClient	361
Issuing a GET Request	362
Issuing a POST Request	363
Lab 11B.....	364
Summary	365
Chapter 12: ASP.NET and Azure	375
What Is Windows Azure?	377
A Windows Azure Testbed.....	378

Windows Azure Demo.....	379
Publish to Azure.....	380
Web Deployment Completed.....	381
Modifying a Web Application	382
Deploy to Original Site	383
Lab 12	384
Summary	385
Appendix A: Learning Resources.....	389
Appendix B: Deployment in IIS 7.5	393
Internet Information Services	394
Installing IIS 7.5	395
MVC with IIS 7.5	396
.NET Framework Version.....	397
Registering ASP.NET	400
Demo: Running an MVC App on IIS	401
Convert to an Application.....	402
XCOPY Deployment.....	403
Lab B	404

EVALUATION COPY

Unauthorized reproduction or distribution is prohibited

Chapter 1

Introduction to ASP.NET MVC

Introduction to ASP.NET MVC

Objectives

After completing this unit you will be able to:

- **Describe advantages and disadvantages of ASP.NET Web Forms.**
- **Understand the Model-View-Controller (MVC) pattern**
- **Outline the parts of an ASP.NET MVC application.**
- **Describe advantages and disadvantages of ASP.NET MVC.**
- **Understand the use of unit testing in creating ASP.NET MVC applications.**

Review of ASP.NET Web Forms

- **ASP.NET Web Forms provide a way to build web applications.**
- **You can use compiled, object-oriented languages with ASP.NET, including C# and Visual Basic.**
 - All the power of the .NET Framework is available to you, including the extensive class library.
- **Code and presentation elements can be cleanly separated.**
 - Code can be provided in a separate section of a Web page from user interface elements.
 - The separation can be carried a step further by use of separate “code behind” files.
- **ASP.NET Web Forms comes with an extensive set of server controls that provide significant functionality out of the box.**
- **Server controls transparently handle browser compatibility issues.**
 - A special set of Mobile Controls can emit either HTML or WML, depending on the characteristics of the client device.
- **Configuration is handled by XML files without need of any registry settings, and deployment can be done simply by copying files.**

Advantages of ASP.NET Web Forms

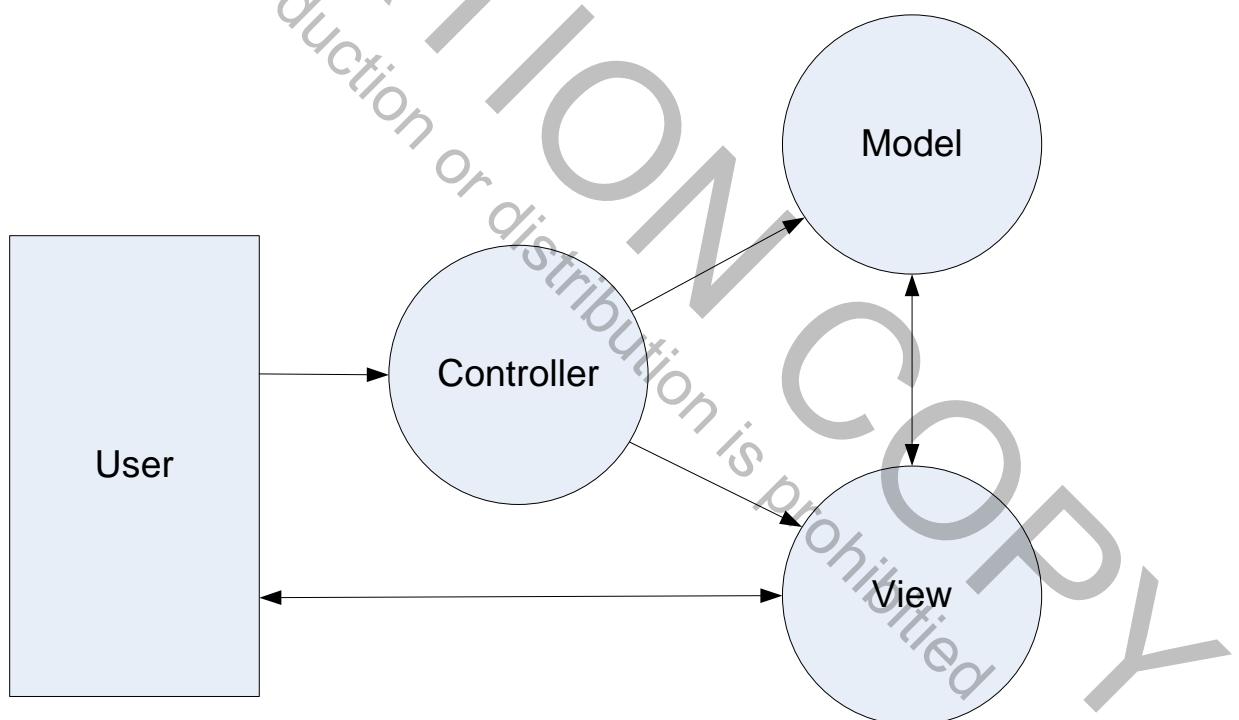
- **ASP.NET Web Forms continue to be supported and have their own advantages:**
 - A rich event model supported in hundreds of server controls facilitates easy development of Web server applications, following a familiar GUI development paradigm.
 - View state makes it easy to manage state information.
 - The model works well for individuals and small teams doing rapid application development.
 - The large number of built-in and third-party components also facilitates rapid application development.
- **In general, Web Forms are quite easy to work with and generally require less code.**

Disadvantages of ASP.NET Web Forms

- Key disadvantages of ASP.NET Web Forms include
 - ViewState tends to be large depending on the number of server controls contained on the page, thus increasing the size of the page and the length of the response time from server to browser
 - ASP.NET Web Forms provide tight coupling with the code-behind classes which make automated testing of the back-end code apart from the web pages more difficult
 - Because the code-behind classes are so tightly coupled to the web forms, developers are encouraged to mix presentation code with application logic in the same code-behind classes which can lead to fragile and unintelligible code
 - Limited control of HTML rendered through use of server controls

Model-View-Controller Pattern

- The Model-View-Controller (MVC) design pattern divides an application into three conceptual components:
 - A **model** represents the data and operations that are meaningful to the domain of the application. It implements the application logic for the domain.
 - **Views** display a user interface for portions of the model. Typically the UI is created from model data.
 - **Controllers** handle incoming requests, work with the model, and select a view to render a UI back to the user.



ASP.NET MVC

- **ASP.NET MVC is a framework based on ASP.NET for creating Web applications.**
 - It is an alternative to Web Forms.
- **ASP.NET MVC 1.0 was installed on top of .NET 3.5 SP1 and Visual Studio 2008 SP.**
- **ASP.NET MVC 2.0 is integrated into .NET 4.0 and Visual Studio 2010.**
- **ASP.NET MVC 3.0 is a separate download and adds important new features, such as the Razor view engine.**
- **ASP.NET MVC 4.0 is integrated into .NET 4.5 and Visual Studio 2012.**
- **ASP.NET MVC 5.0 is integrated into .NET 4.5.x/4.6/4.7 and Visual Studio 2013/2015/2017.**
- **ASP.NET MVC 6.0, part of ASP.NET Core, unifies MVC with Web API.**
 - ASP.NET Core is covered in OI course 4043 .
- **ASP.NET MVC does not replace Web Forms but is an alternative approach to creating Web applications.**
 - It relies on the same ASP.NET infrastructure as does Web Forms and is integrated with ASP.NET features such as master pages and membership-based authentication.

Advantages of ASP.NET MVC

- **Key advantages of ASP.NET MVC include:**
 - The MVC pattern promotes **separation of concerns** into input logic (controller), business logic (model) and UI (view). This aids in managing complexity.
 - These components are loosely coupled, promoting parallel development.
 - This loose coupling also facilitates automated testing.
 - Views are created using standard HTML and cascading style sheets, giving the developer a high degree of control over the user interface.
 - There is no view state, reducing the load on the browser in rendering a page.
- **Separation of Concerns:**
 - Each component has one responsibility
 - SRP – Single Responsibility Principle
 - DRY – Don't Repeat Yourself
 - More easily testable
 - Helps with concurrent development

Disadvantages of ASP.NET MVC

- Key disadvantages of ASP.NET MVC include:
 - Writing View contents the old ASP-like way (though this is now easier with the newer Razor view syntax).
 - Unit testing and Test Driven Development (TDD) are encouraged and used more but also bring a steep learning curve.
 - Need to understand HTML controls and style sheets, but at the same time this allows a designer to work independently of the coders.

Goals of ASP.NET MVC

- **The ASP.NET MVC Framework has the following goals:**
 - Frictionless Testability
 - Tight control over markup
 - User/Search Engine friendly URLs
 - Leverage the benefits of ASP.NET
 - Conventions and Guidance
- **Extensibility**
 - Replace any component of the system
 - Interface-based architecture
 - Very few sealed methods/classes

Unit Testing

- Unit testing lets you specify the expected behavior of individual classes or other small code units in isolation.
- ASP.NET MVC encourages unit testing of the Models and the Controllers of the application to verify expected behaviors.
- Separation of concerns makes unit testing of individual components feasible.
- You may use Microsoft's Visual Studio Unit Test Framework for unit testing or other unit test framework such as NUnit.
 - NUnit is free open source software.
 - Visual Studio Unit Test Framework is now built into the free Community version of Visual Studio 2017 and is used in this course.

Summary

- ASP.NET Web Forms is still used and has both advantages and disadvantages compared to MVC.
- The Model-View-Controller (MVC) pattern is useful in creating applications that have separation of concerns.
- Unit testing is helpful and encouraged in developing ASP.NET MVC applications

Chapter 2

Getting Started with ASP.NET MVC

Getting Started with ASP.NET MVC

Objectives

After completing this unit you will be able to:

- Understand how ASP.NET MVC is used within Visual Studio.
- Create several versions of a simple ASP.NET MVC application.
- Understand how Views are rendered.
- Use the Razor view engine in ASP.NET MVC 5.
- Understand how dynamic output works.
- Pass input data to an MVC application in a query string.

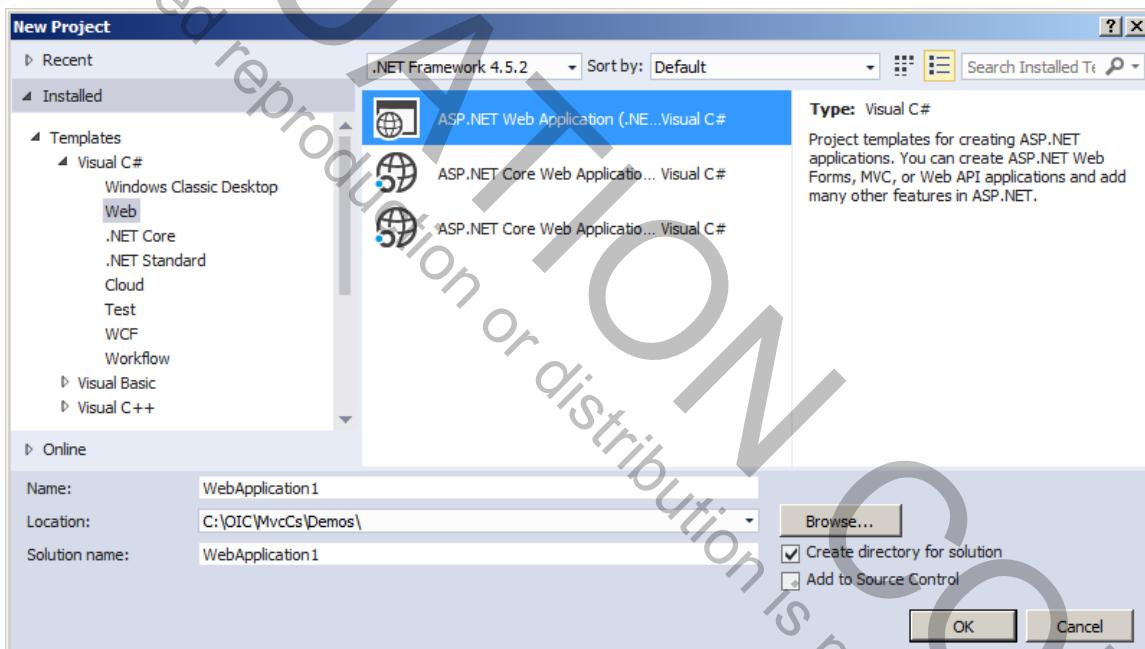
An ASP.NET MVC 5 Testbed

- **This course uses the following software:**
 - Visual Studio 2017. The course was tested using the free Visual Studio Community 2017.
 - This includes bundled ASP.NET MVC 5.
 - SQL Server 2016 LocalDB, which also comes bundled with Visual Studio 2017.
- **Recommended operating system is Windows 7 SP1, which is what was used in developing this course.**
- **If you want to practice deployment on IIS, you should also have IIS installed.**
 - See Appendix B.

Visual Studio ASP.NET MVC Demo

- Let's use Visual Studio to create an ASP.NET MVC 5 Web Application project.

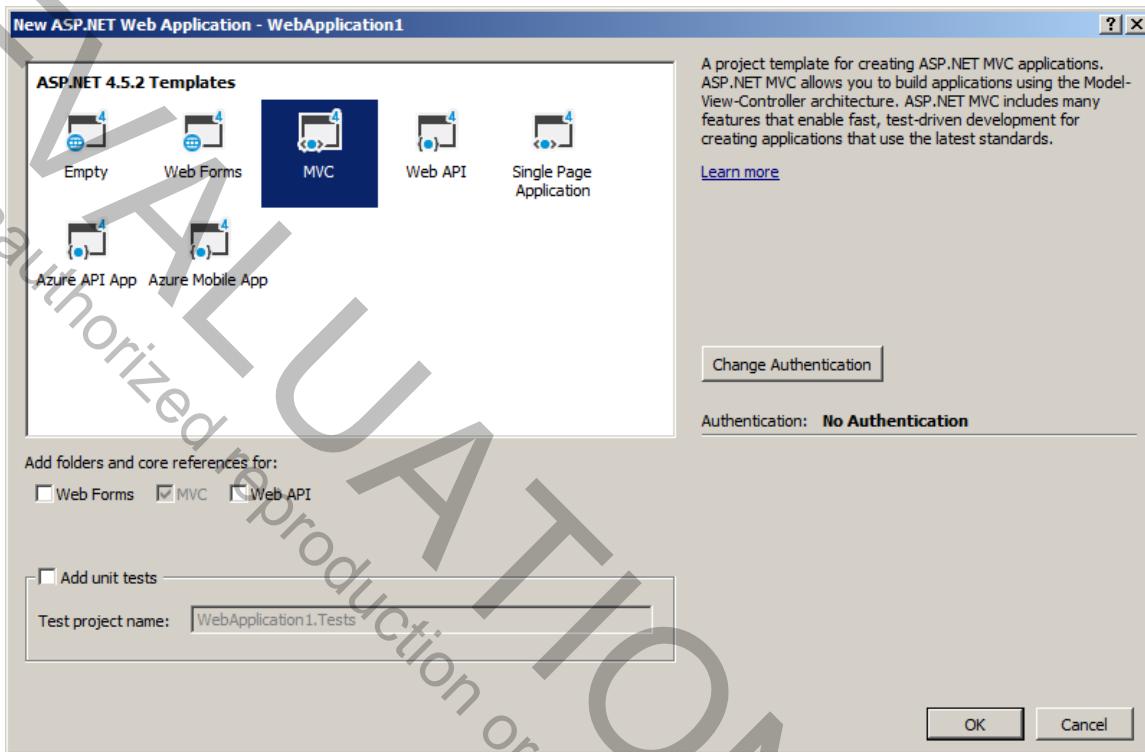
- From File | New Project choose ASP.NET Web Application.
- Browse to the C:\OIC\MvCs\Demos folder, and leave the name as WebApplication1.
 - You will be able to choose on the next screen whether to create a Web Forms, MVC, or Web API project.



- Click OK.

ASP.NET MVC Demo (Cont'd)

4. Choose the MVC template.



5. Click OK.

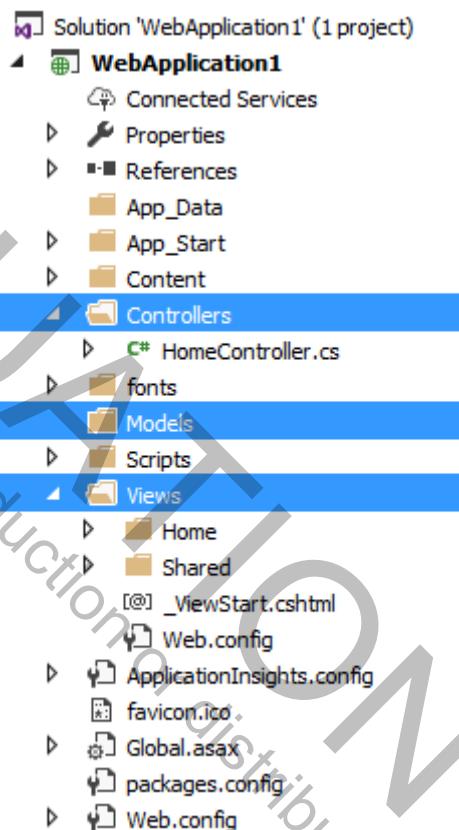
ASP.NET Documentation Page

- You will see an ASP.NET documentation page displayed.



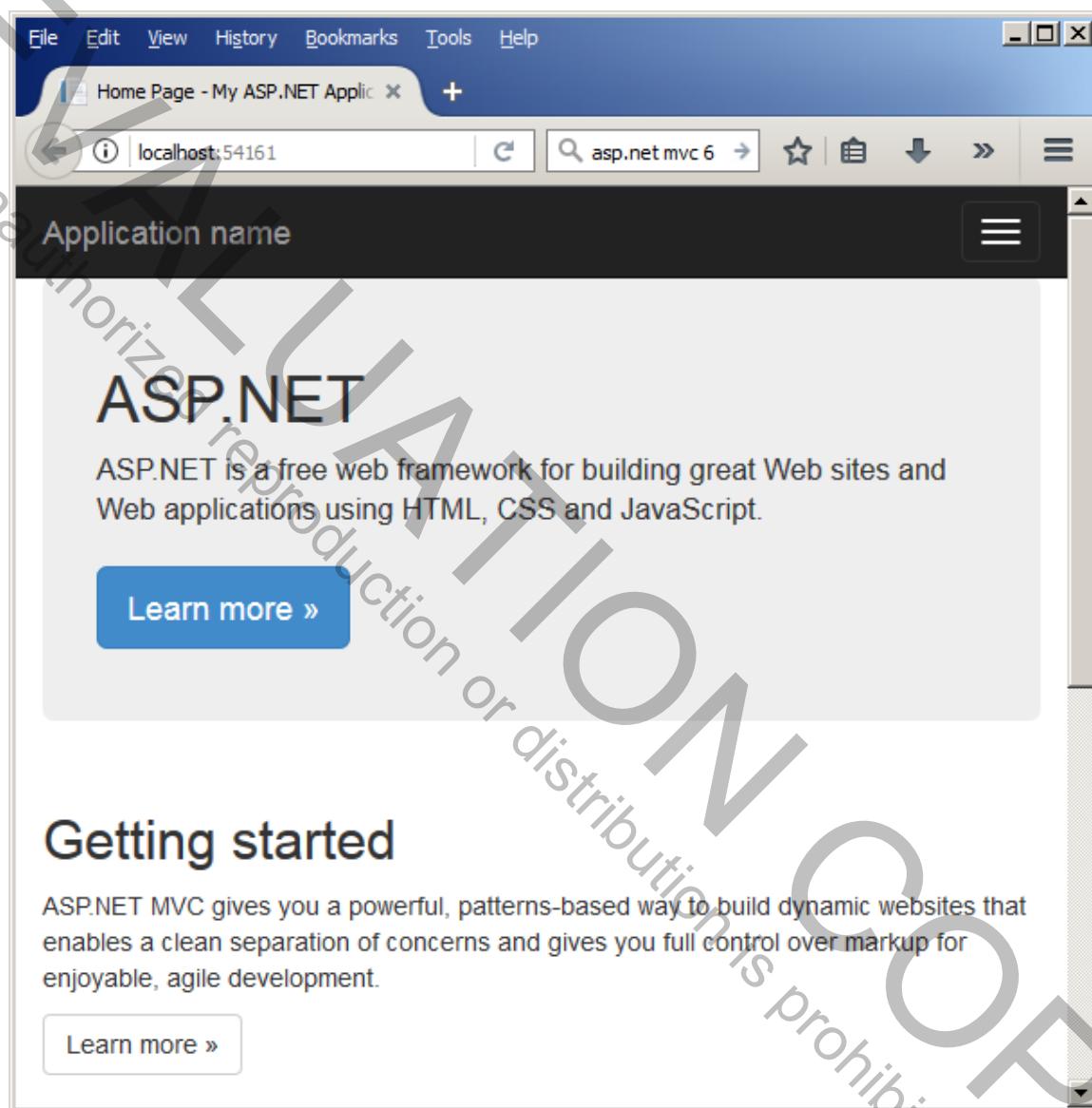
Starter Application

- Notice that there are separate folders for Controllers, Models and Views.



Starter Application (Cont'd)

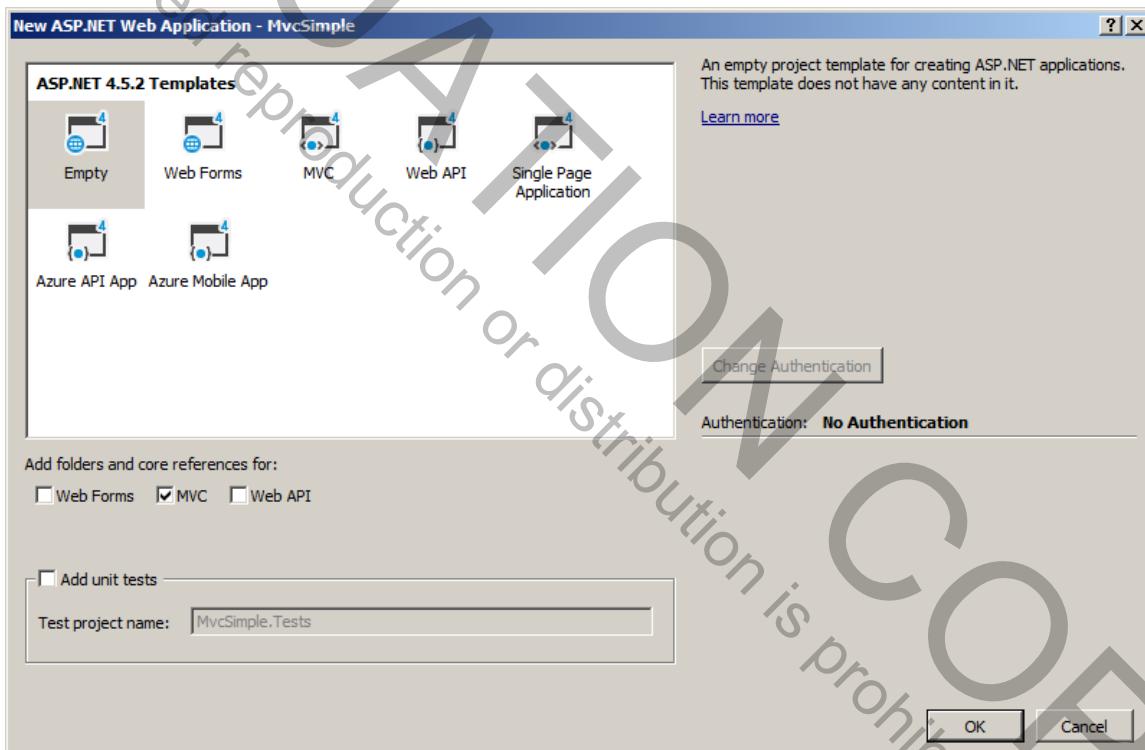
- Build and run this starter application¹:



¹ Visual Studio will automatically start your default browser to run the application. In our screenshots you will sometimes see Firefox and sometimes Internet Explorer.

Simple App with Controller Only

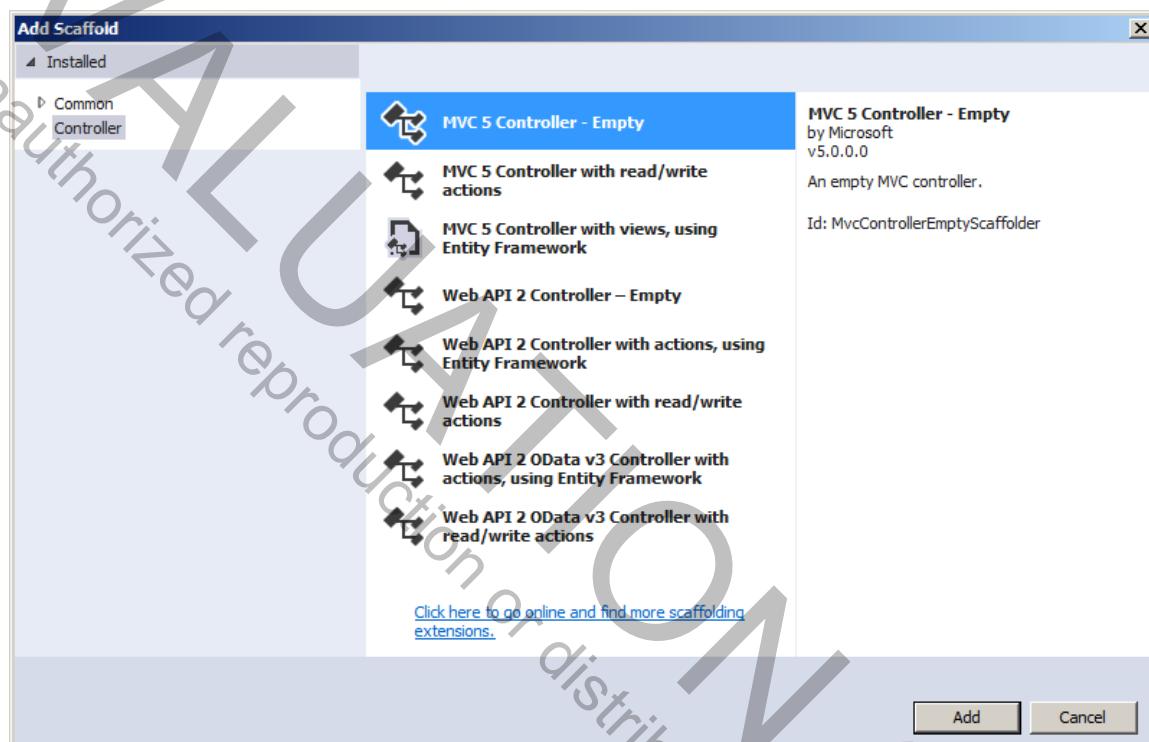
- To start learning how ASP.NET MVC works, let's create a simple app with only a controller.
1. Create a new ASP.NET Web Application project with the name **MvcSimple** in the **Demos** folder.
 2. This time choose the Empty project template.
 3. Check MVC. Note that the same project can include any combination of Web Forms, MVC and Web API.



4. Click OK.

Demo: Controller Only (Cont'd)

5. Right-click over the Controllers folder and choose Add | Controller from the context menu.
6. Choose MVC 5 Controller - Empty for the scaffold.



7. Click Add.
8. Provide the name **HomeController**



9. Click Add.

Demo: Controller Only (Cont'd)

10. Examine the generated code **HomeController.cs**.

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

namespace MvcSimple.Controllers
{
    public class HomeController : Controller
    {
        // GET: /Home/
        public ActionResult Index()
        {
            return View();
        }
    }
}
```

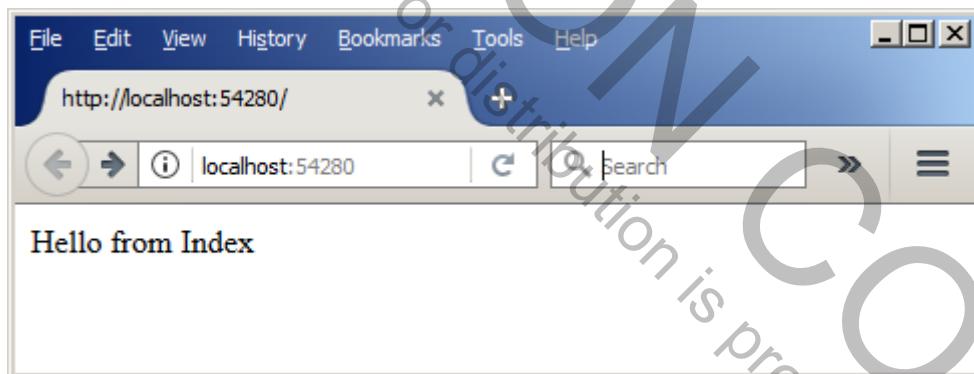
Demo: Controller Only (Cont'd)

11. Replace the code for the **Index()** method by the following. Also, provide a similar **Foo()** method.

```
public class HomeController : Controller
{
    // GET: /Home/
    public string Index()
    {
        return "Hello from Index";
    }

    public string Foo()
    {
        return "Hello from Foo";
    }
}
```

12. Build and run.



13. Examine the URL Visual Studio used to invoke the application. (The port number varies.)

`http://localhost:54280/`

Demo: Controller Only (Cont'd)

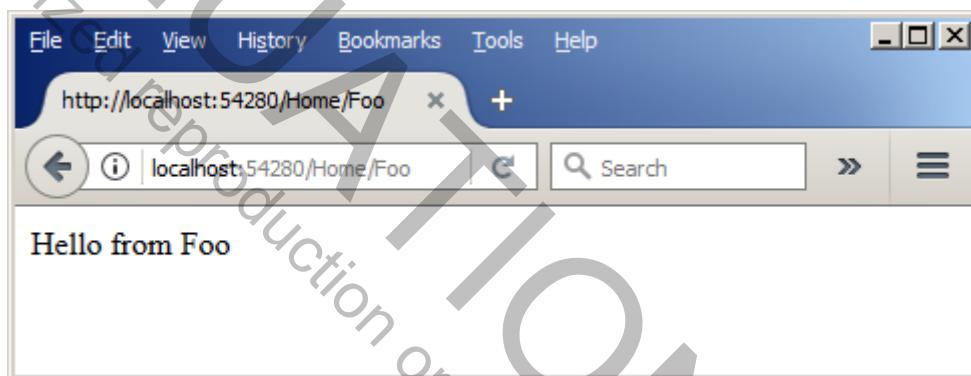
14. Now try using these URLs². You should get the same result.

`http://localhost:54280/Home/`
`http://localhost:54280/Home/Index/`

15. Now try this URL.

`http://localhost:54280/Home/Foo/`

You will see the second method **Foo()** invoked:



16. Finally, let's add a second controller **SecondController.cs**.
17. Provide the following code for the **Index()** method of the second controller.

```
public class SecondController : Controller
{
    // GET: /Second/
    public string Index()
    {
        return "Hello from second controller";
    }
}
```

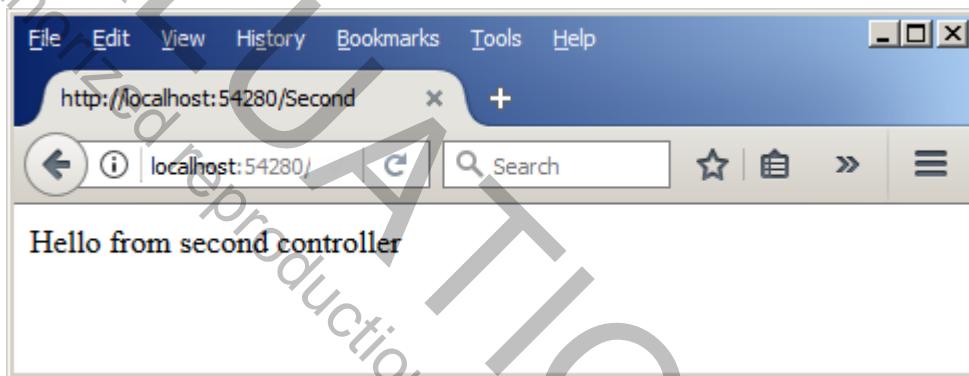
² The trailing forward slash in these URLs is optional.

Demo: Controller Only (Cont'd)

18. You can invoke this second controller using either of these URLs:

`http://localhost:54280/Second/`
`http://localhost:54280/Second/Index/`

In either case we get the following result. The program at this point is saved in **MvcSimple\Controller** in the chapter folder³.



³ You should open all the ASP.NET MVC examples as projects, not web sites.

Action Methods and Routing

- Every public method in a controller is an *action method*.
 - This means that the method can be invoked by some URL.
- The ASP.NET MVC routing mechanism determines how each URL is mapped onto particular controllers and actions.
- The default routing is specified in the file *RouteConfig.cs*, contained in the *App_Start* folder.

```
public static void RegisterRoutes(RouteCollection routes)
{
    routes.IgnoreRoute(
        "{resource}.axd/{*pathInfo}");

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

- If desired, additional route maps can be set up here.

Action Method Return Type

- An action method normally returns a result of type **ActionResult**.
 - An action method can return any type, such as **string**, **int**, and so on, but then the return value is wrapped in an **ActionResult**.
- The most common action of an action method is to call the **View()** helper method, which returns a result of type **ViewResult**, which derives from **ActionResult**.
- The table shows some of the important action result types, which all derive from **ActionResult**.

Action Result	Helper Method	Description
ViewResult	View()	Renders a view as a Web page, typically HTML
RedirectResult	Redirect()	Redirects to another action method using its URL
JsonResult	Json()	Returns a serialized Json object
FileResult	File()	Returns binary data to write to the response

Rendering a View

- Our primitive controllers simply returned a text string to the browser.
- Normally, you will want an HTML page returned. This is done by rendering a *view*.
 - The controller will return a **ViewResult** using the helper method **View()**.

```
public ViewResult Index()
{
    return View();
}
```

- Try doing this in the *MvcSimple* program. Build and run. It compiles but you get a runtime error.

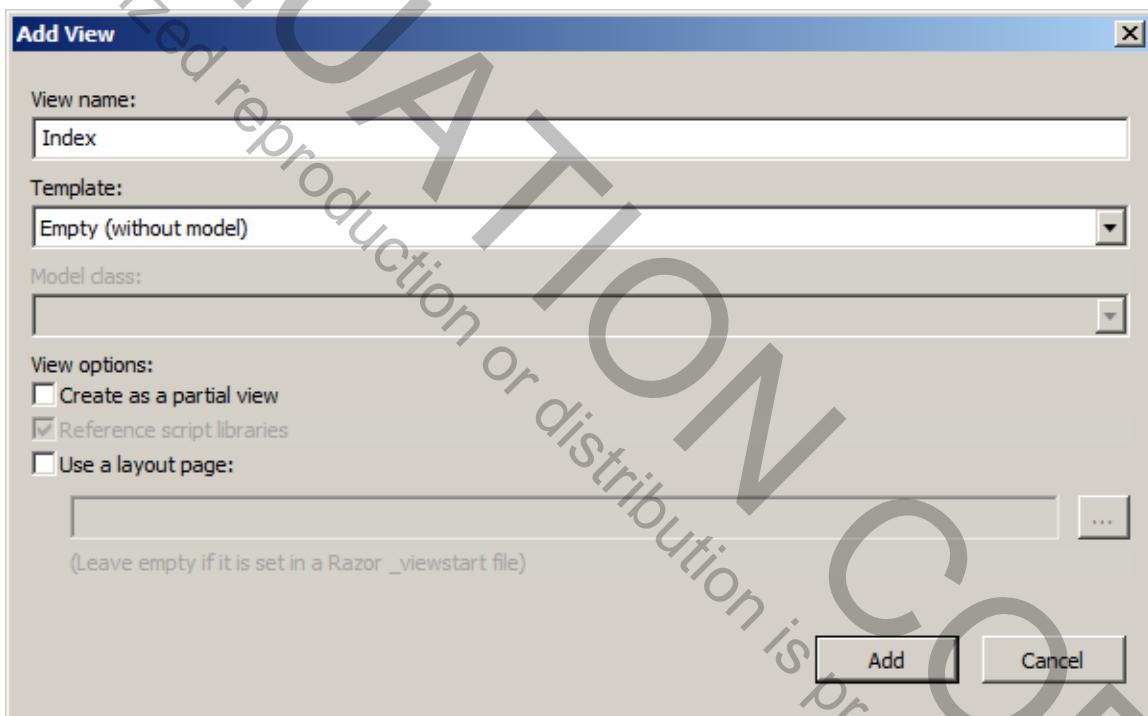
Server Error in '/' Application.

The view 'Index' or its master was not found or no view engine supports the searched locations. The following locations were searched:

*~/Views/Home/Index.aspx
~/Views/Home/Index.ascx
~/Views/Shared/Index.aspx
~/Views/Shared/Index.ascx
~/Views/Home/Index.cshtml
~/Views/Home/Index.vbhtml
~/Views/Shared/Index.cshtml
~/Views/Shared/Index.vbhtml*

Creating a View in Visual Studio

- **The error message is quite informative!**
 - Let us create an appropriate file **Index.cshtml** in the folder **Views/Home**.
- **In Visual Studio you can create a view by right-clicking in the action method. Choose Add View.**
 - Clear the check box for layout page and click Add

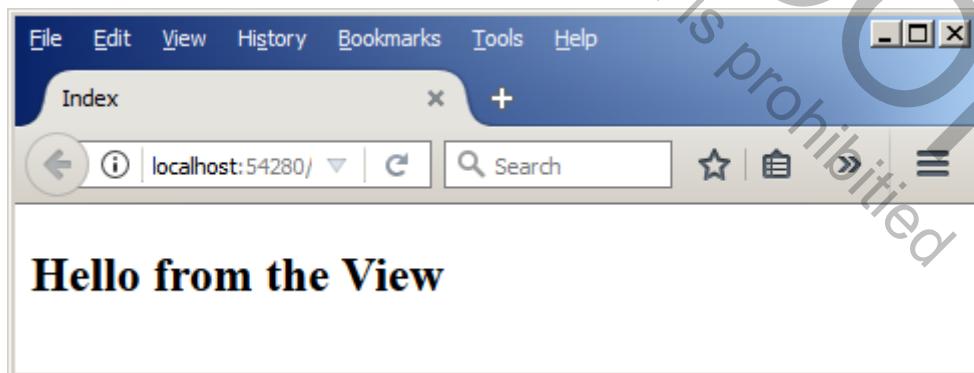


The View Web Page

- A file **Index.cshtml** is created in the **Views\Home** folder.
 - Edit this file to display a welcome message from the view.
To make it stand out, use H2 format.

```
@{  
    Layout = null;  
}  
  
<!DOCTYPE html>  
  
<html>  
<head>  
    <meta name="viewport"  
          content="width=device-width" />  
    <title>Index</title>  
</head>  
<body>  
    <h2>Hello from the View</h2>  
</body>  
</html>
```

- Build and run.



Dynamic Output

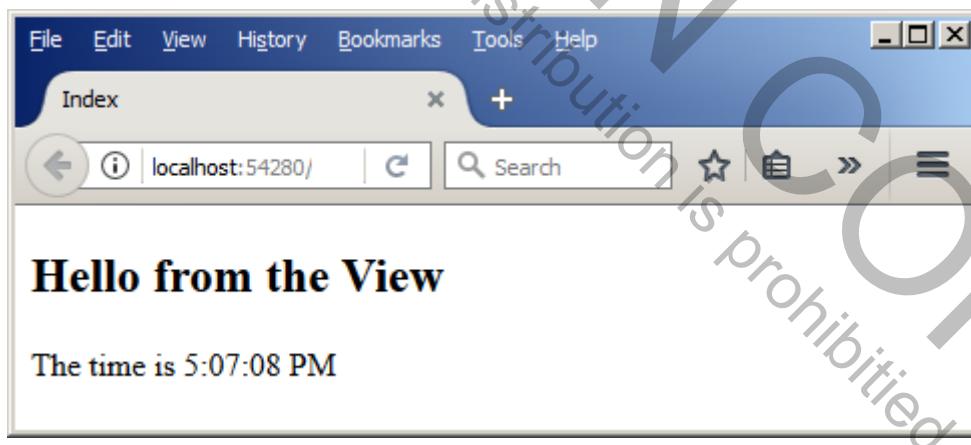
- ***ViewBag* is a dynamic type that can be used for passing data from the controller to the view, enabling the rendering of dynamic output.**
- **This code in the controller stores the current time.**

```
public ViewResult Index()
{
    ViewBag.Time =
        DateTime.Now.ToString();
    return View();
}
```

- **This markup in the view page displays the data.**

```
<h2>Hello from the View</h2>
The time is @ViewBag.Time
```

- **Here is a run:**



- The program is saved in **MvcSimple\View**.

Razor View Engine

- From the beginning ASP.NET MVC has supported “view engines”, which are pluggable components that implement different syntax options for view templates.
- In ASP.NET MVC 1.0 and 2.0 the default view engine is the Web Forms (or ASPX) view engine.
- In ASP.NET MVC 3.0 and 4.0 the default view engine is Razor.
 - In creating a view, Visual Studio allowed you to choose whether to use ASPX or Razor.
- Razor template syntax is much more concise than ASPX template syntax.
 - You use @ in place of <%=...%>
 - The Razor parser makes use of syntactic knowledge of C# code (in a .cshtml file) or of VB code (in a .vbhtml file).
- In ASP.NET MVC 5.0 the Razor view engine is used automatically, and we will employ it in our examples.

Embedded Scripts

- **Razor makes it easy to use embedded C# script in an HTML page. Simply enclose it with @{}.**

```
@{  
    int day = 0;  
    int gifts = 0;  
    int total = 0;  
    while (day < 12)  
    {  
        day += 1;  
        gifts += day;  
        total += gifts;  
    }  
}
```

- **You can convert an object to a string and display it in HTML simply by using the @ symbol in front of it.**

```
<p>Total number of gifts = @total</p>
```

- **Inside an embedded script you can simply use HTML elements, giving you great flexibility in output.**

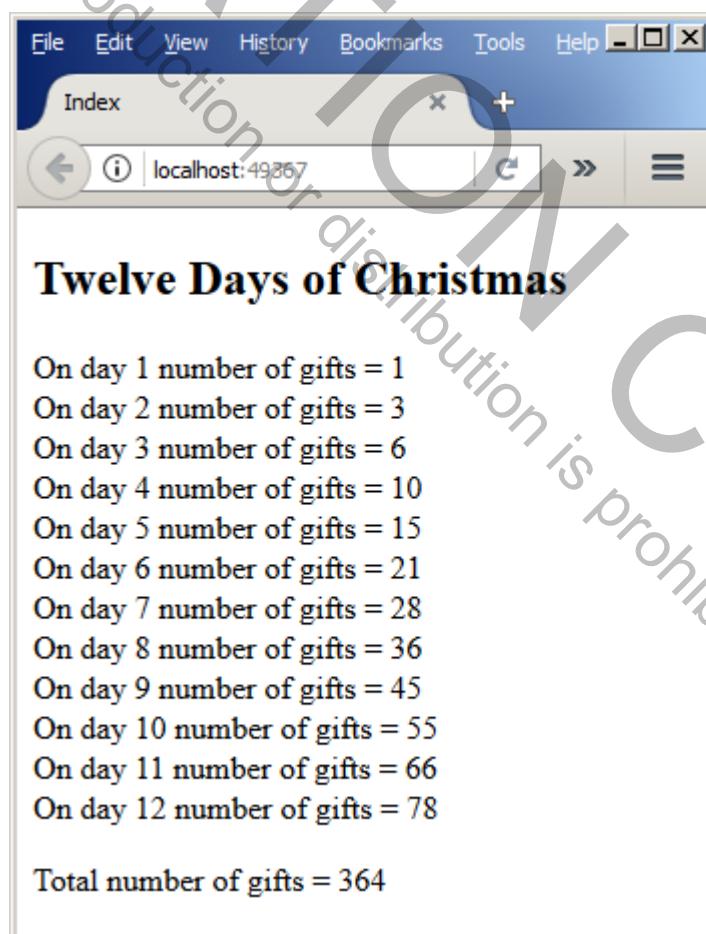
- You can use literal text by prefacing it with @::

```
@{  
    ...  
    while (day < 12)  
    {  
        day += 1;  
        gifts += day;  
        total += gifts;  
        @::On day @day number of gifts = @gifts <br />  
    }  
}
```

Embedded Script Example

- See *MvcSimple\Script*.

```
@{  
    int day = 0;  
    int gifts = 0;  
    int total = 0;  
    while (day < 12)  
    {  
        day += 1;  
        gifts += day;  
        total += gifts;  
        @:On day @day number of gifts = @gifts <br />  
    }  
}
```



Using a Model with ViewBag

- Our next version of the program uses a model along with the ViewBag.
 - See **MvcSimple\ModelViewBag** in the chapter folder.
- The model contains a class defining a *Person*.
 - See the file **Person.cs** in the **Models** folder of the project.
 - There are public properties **Name** and **Age**.
 - Unless otherwise assigned, **Name** is “John” and **Age** is 33.

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

namespace MvcSimple.Models
{
    public class Person
    {
        public string Name { get; set; }
        public int Age { get; set; }
        public Person()
        {
            Name = "John";
            Age = 33;
        }
    }
}
```

Controller Using Model and ViewBag

- The controller instantiates a *Person* object and passes it in *ViewBag*.
 - Note that we need to import the **MvcSimple.Models** namespace.

```
using MvcSimple.Models;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

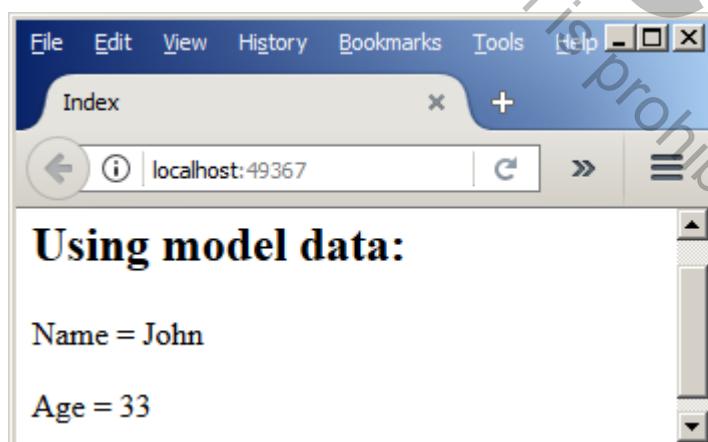
namespace MvcSimple.Controllers
{
    public class HomeController : Controller
    {
        // GET: /Home/
        public ViewResult Index()
        {
            ViewBag.person = new Person();
            return View();
        }
    }
}
```

View Using Model and ViewBag

- The view displays the output using appropriate script.
 - Again we need to import the **MvcSimple.Models** namespace.

```
@{ Layout = null; }  
@using MvcSimple.Models;  
<!DOCTYPE html>  
<html>  
<head>  
    <meta name="viewport" content="width=device-width" />  
    <title>Index</title>  
</head>  
<body>  
    @{ Person p = ViewBag.person; }  
    <h2>Using model data:</h2>  
    <p>Name = @p.Name</p>  
    <p>Age = @p.Age </p>  
</body>
```

- The output:



Using Model Directly

- You may pass a single model object to a view through the use of an overloaded constructor of the View() method.
 - For an example see **MvcSimple\Model**.
- To see how this works, first rewrite the controller.

```
public ViewResult Index()
{
    return View(new Person());
}
```

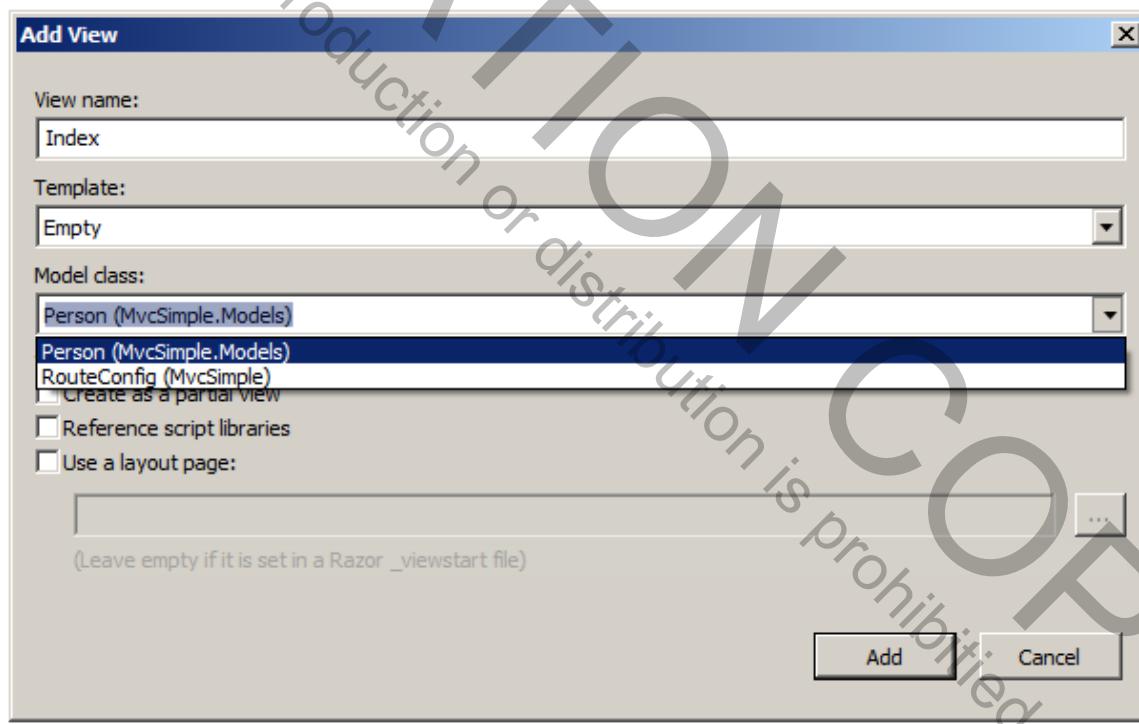
- The parameter to the overload of the View() method is a model object.
- Next, rewrite the view page.

```
@model MvcSimple.Models.Person
...
<body>
    <h2>Using model data:</h2>
    <p>Name = @Model.Name</p>
    <p>Age = @Model.Age </p>
</body>
</html>
```

- The **Person** object is passed as a parameter to the view, and the model object can be accessed through the variable **Model**.
- We no longer need the script code.

A View Using Model in Visual Studio

- To create a view using a Model in Visual Studio, right-click inside an action method and choose Add View from the context menu.
- You may create a view tied to the model by selecting a model from the dropdown.
 - You should build the application first in order that the dropdown be populated.
 - Select the Empty template, rather than the Empty (without model) template.



- You can demonstrate this for yourself by deleting the view in the **MvcSimple\Model** example.

View Created by Visual Studio

- Here is the view page created by Visual Studio:

```
@model MvcSimple.Models.Person  
  
{@  
    Layout = null;  
}  
  
<!DOCTYPE html>  
  
<html>  
<head>  
    <meta name="viewport"  
          content="width=device-width" />  
    <title>Index</title>  
</head>  
<body>  
    <div>  
    </div>  
</body>  
</html>
```

Passing Parameters in Query String

- In MVC applications you will typically need to handle input data in one manner or another.
- A simple way to pass input data is through the query string on the URL that invokes the application.
- For an example, see the *MvcHello* application in the chapter folder.
 - Pass the name in the query string, for example:

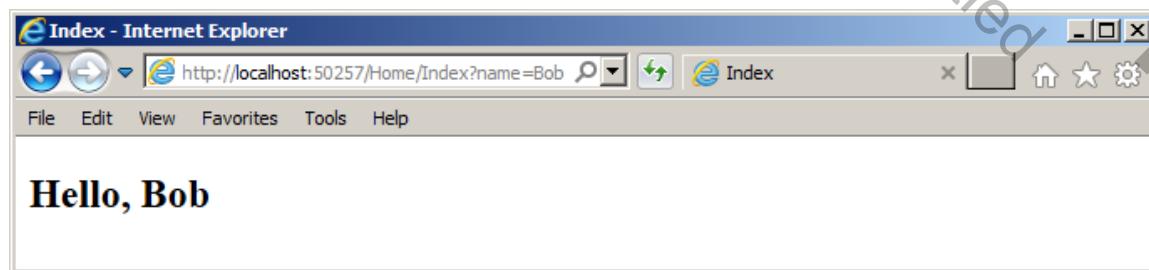
/Home/Index?name=Bob

- The Index action method in the home controller takes name as a parameter, which is stored in the ViewBag.

```
// GET: /Home/Index?name=x
public ActionResult Index(string name)
{
    ViewBag.Name = name;
    return View();
}
```

- The view displays a greeting using the name.

```
<body>
    <h2>Hello, @ViewBag.Name</h2>
</body>
```



Lab 2

Contact Manager Application

In this lab you will implement an ASP.NET MVC application that creates a contact and displays it on the page. The contact can be changed by passing the first and last names in the query string. The model persists the contact in a flat file.

Detailed instructions are contained in the Lab 2 write-up at the end of the chapter.

Suggested time: 30 minutes

Summary

- You can begin creating an ASP.NET MVC application with the controller, which handles various URL requests.
- From an action method of a controller you can create a view using Visual Studio.
- ASP.NET MVC 5 uses the Razor view engine.
- You can pass data from the controller to the view by using the *ViewBag*.
- By creating a model you can encapsulate the business data and logic.
- You can pass data to an MVC application in query string.

Lab 2

Contact Manager Application

Introduction

In this lab you will implement an ASP.NET MVC application that creates a contact and displays it on the page. The contact can be changed by passing the first and last names in the query string. The model persists the contact in a flat file.

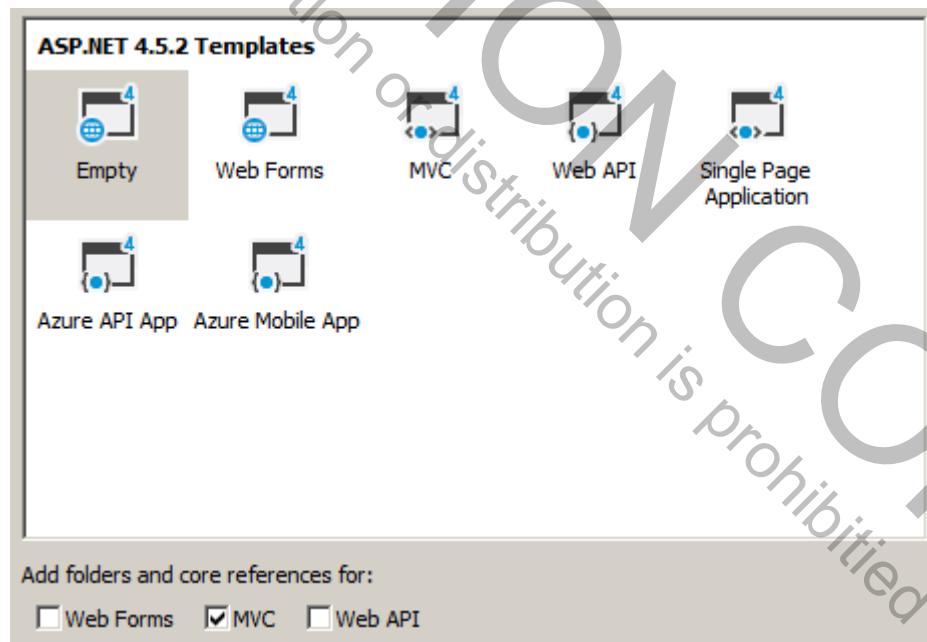
Suggested Time: 30 minutes

Root Directory: C:\OIC\MvcCs

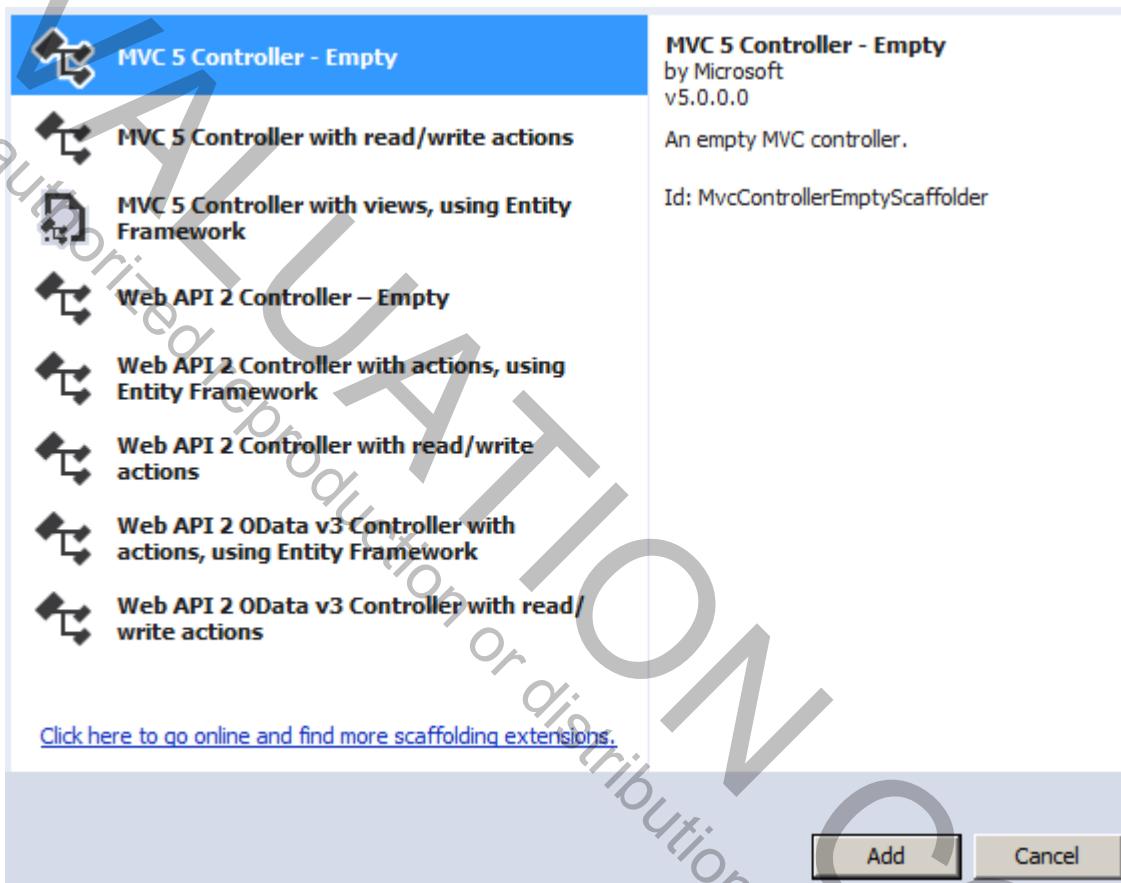
Directories: **Labs\Lab2** (do your work here)
Labs\Lab2>Contact.cs (starter code for model)
Chap02\MvcContact (solution)

Instructions

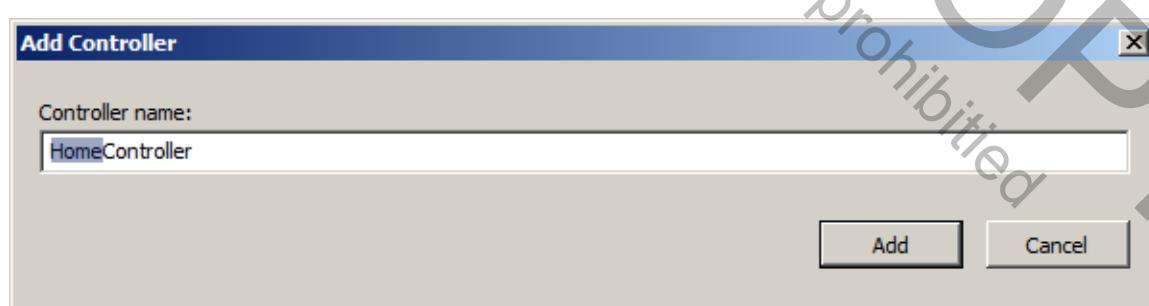
1. Create a new ASP.NET Empty Web Application **MvcContact** in the working directory. Add folders and core references for MVC.



2. Copy the file **Contact.cs** defining a model class to the **Models** folder and add it to your new project. Examine the code. There are public properties **FirstName** and **LastName** and public static methods to read and write the contact to the flat file **contact.txt** in the **\OIC\Data** folder. A constructor initializes the contact to what is read in from the file.
3. Right-click over the Controllers folder and choose Add | Controller from the context menu. Select the MVC 5 Controller – Empty template and click Add.



4. Assign name **HomeController** and click Add.

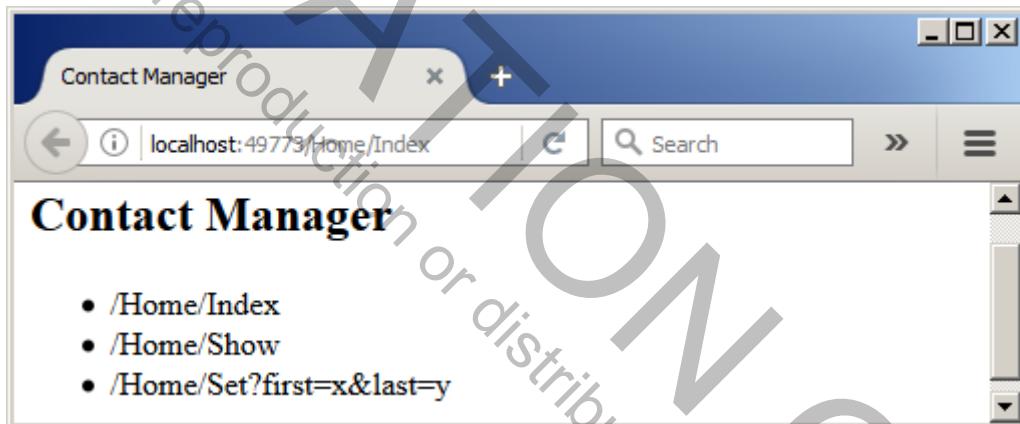


5. Add a view corresponding to the **Index()** action method. Use the suggested name **Index** and the Empty (without model) template. Do not use a layout page.

6. Make the title of the view “Contact Manager”. Provide HTML for a little help page consisting of an unordered list showing three URLs for invoking the application, corresponding to action methods Index, Show and Set. The latter takes a query string specifying first and last names.

```
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Contact Manager</title>
</head>
<body>
    <h2>Contact Manager</h2>
    <ul>
        <li>/Home/Index</li>
        <li>/Home>Show</li>
        <li>/Home/Set?first=x, last=y</li>
    </ul>
</body>
</html>
```

7. Build and run the application. You should see the help page displayed.

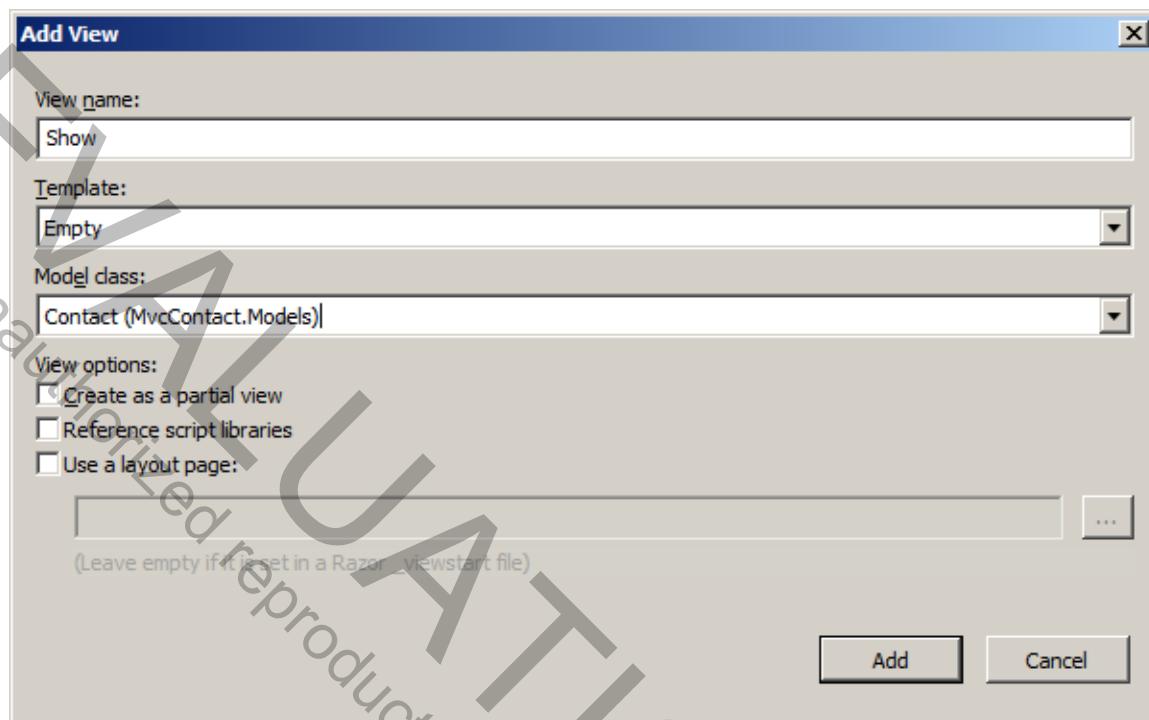


8. Provide a **Show()** action method. Use an override of **View()** that takes the name of the view as the first argument and an object as the second object. Use a new **Contact** as the object.

```
// GET: /Home>Show
public ActionResult Show()
{
    return View("Show", new Contact());
}
```

9. Import the namespace **MvcContact.Models** so that you can access the **Contact** class.
10. Build the project to make sure you get a clean compile and so that you can use the model when you create the view.

11. Right-click inside this new action method to add a view. Accept the suggested name Show. From the dropdown for Model class select the Contact class. See screen capture on the following page.



12. The scaffolding will have placed a `@model` directive at the top of the **.cshtml** file. Provide Razor HTML code to display the first and last names separated by a space.

```
@model MvcContact.Models.Contact

 @{
    Layout = null;
}

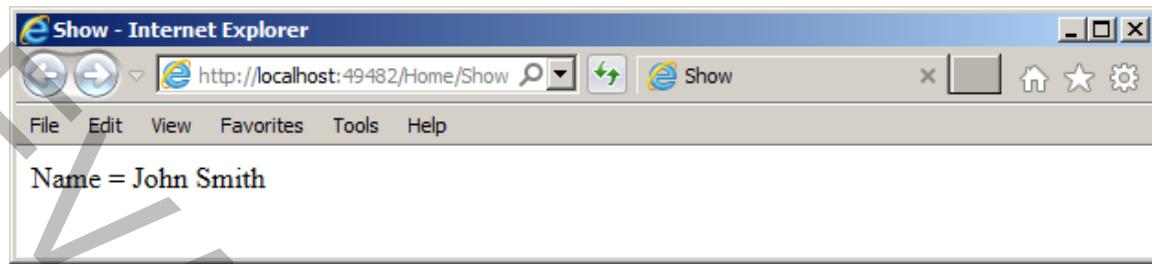
<!DOCTYPE html>

<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Show</title>
</head>
<body>
    <div>
        Name = @Model.FirstName @Model.LastName
    </div>
</body>
</html>
```

13. Build and run. You should initially see the help page. Then modify the URL in the browser to invoke Show:

/Home/Show

You should see the default name that is stored in the file.



14. Next provide a third controller action method **Set()** which takes as parameters strings for the first and last name. As a comment, show the query string by which the parameters will be passed in the URL. The code should write the contact to the flat file and store the first and last names in the ViewBag.

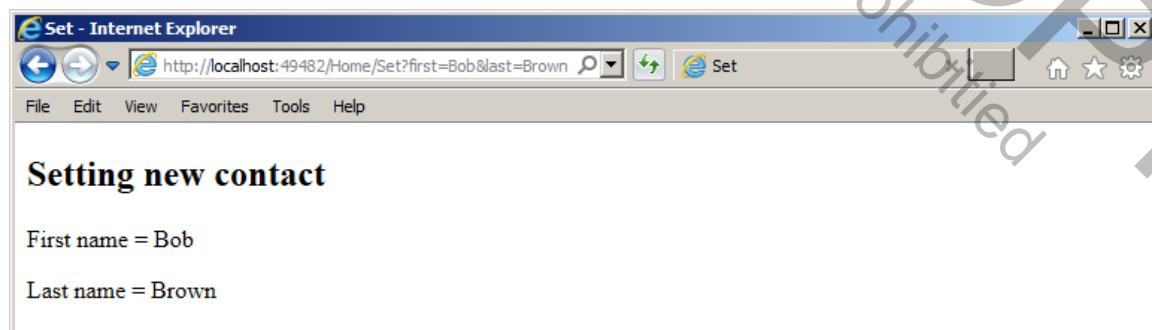
```
// GET: /Home/Set?first=x,last=y
public ActionResult Set(string first, string last)
{
    Contact.WriteContact(first, last);
    ViewBag.First = first;
    ViewBag.Last = last;
    return View();
}
```

15. Add a corresponding view, in which you display the parameters as stored in the ViewBag.

```
<body>
    <h2>Setting new contact</h2>
    <p>First name = @ViewBag.First</p>
    <p>Last name = @ViewBag.Last</p>
</body>
```

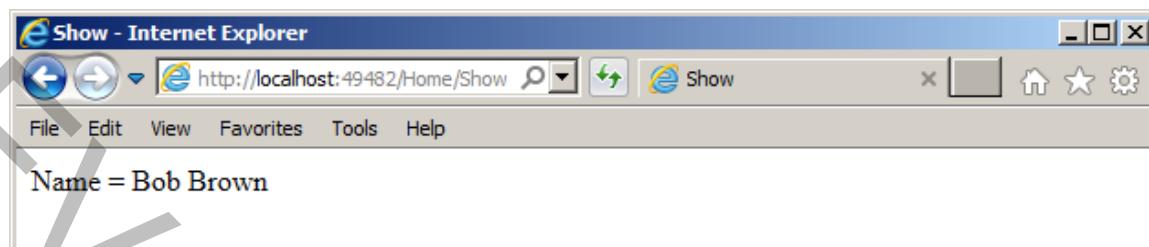
16. Build and run. Invoke Set, providing first and last names in the query string, for example:

/Home/Set?first=Bob&last=Brown



17. Finally, invoke Show again. You should see the new contact displayed.

/Home / Show



18. As a final adjustment to the project, set the project properties so that you will always display the home page when you run the application, not whatever view happens to be open in the editor. You can do this by setting the Start Action to a specific page, which will be Home/Index.

