TRAINING @ SILVER TOUCH TECHNOLOGIES LTD.

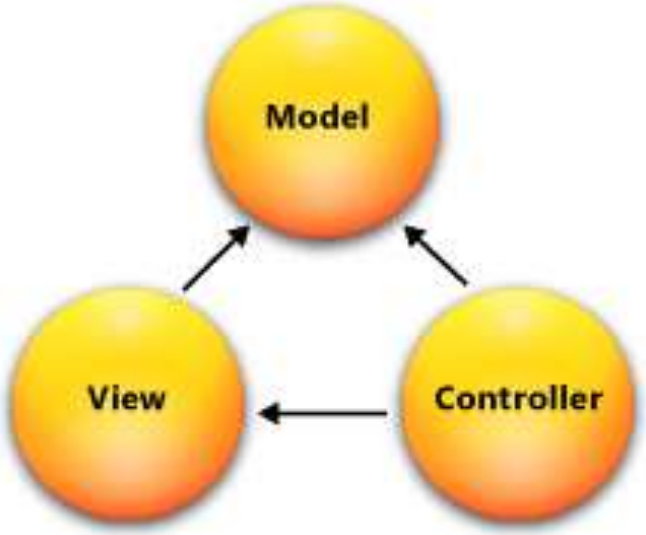
A LEADER IN SYSTEMS INTEGRATION &
DIGITAL TRANSFORMATION

SINCE 1995

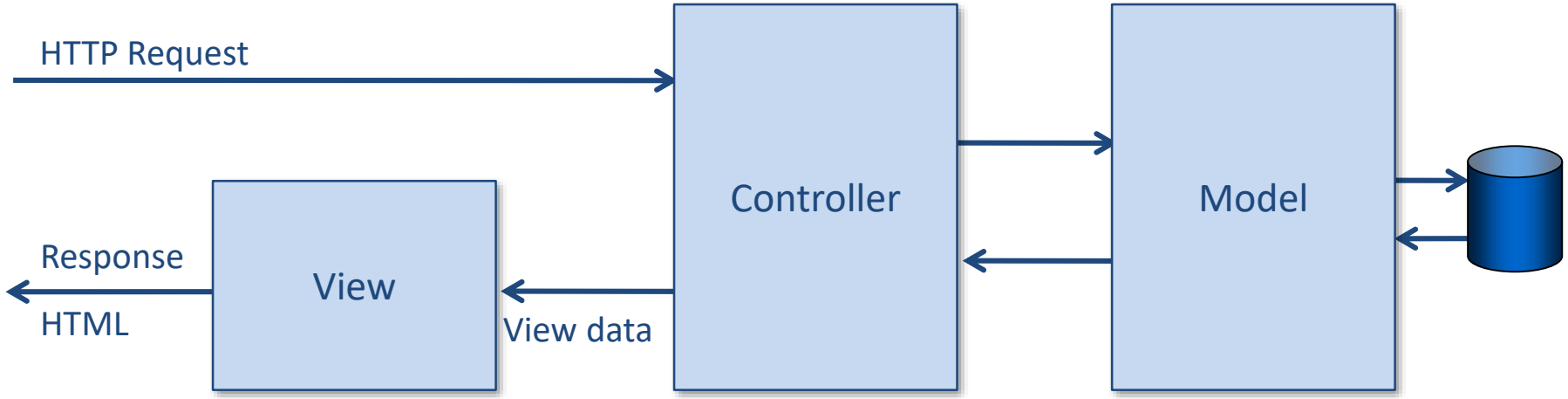
ASP .NET MVC 5

What is MVC?

- Model-View-Controller (MVC)
- Standard Architectural Pattern
- Separation of concerns:
  model, view, controller



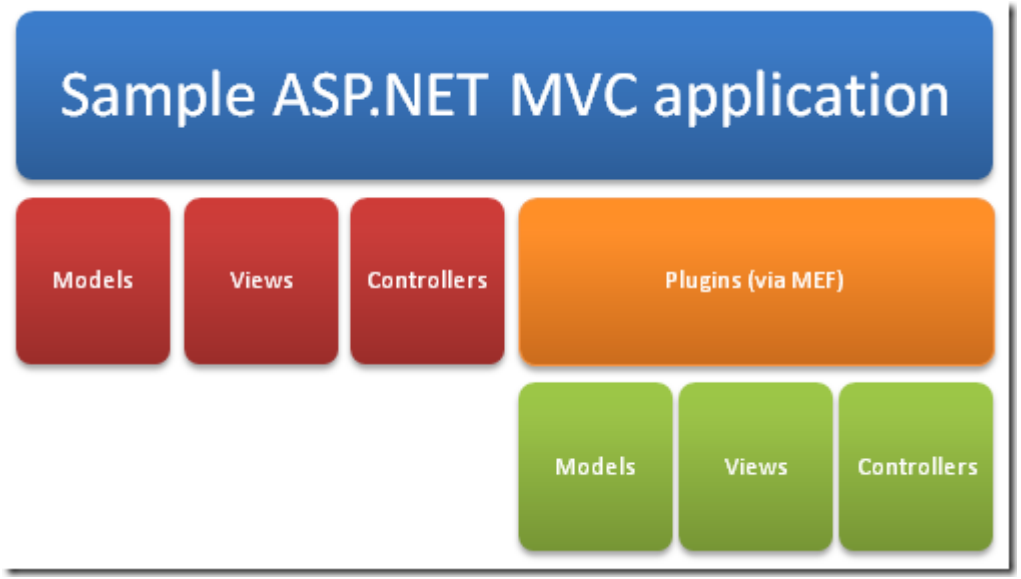
ASP.NET MVC Architecture



- **The controller is probably the most interesting player**
  - Processes user input
    - Works with the model to handle the request
  - Manages application logic
    - E.g. navigating a multi-step process, authentication, etc.
  - Prepares the data to be displayed
    - This is known as "view data" in ASP.NET MVC

ASP .NET MVC Framework

- An alternative to ASP .NET Web Forms
- Presentation framework
  - Lightweight
  - Highly testable
  - Integrated with the existing ASP .NET features:
    - Master pages
    - Membership-Based Authentication
    - …



MVC Flow

Step 1

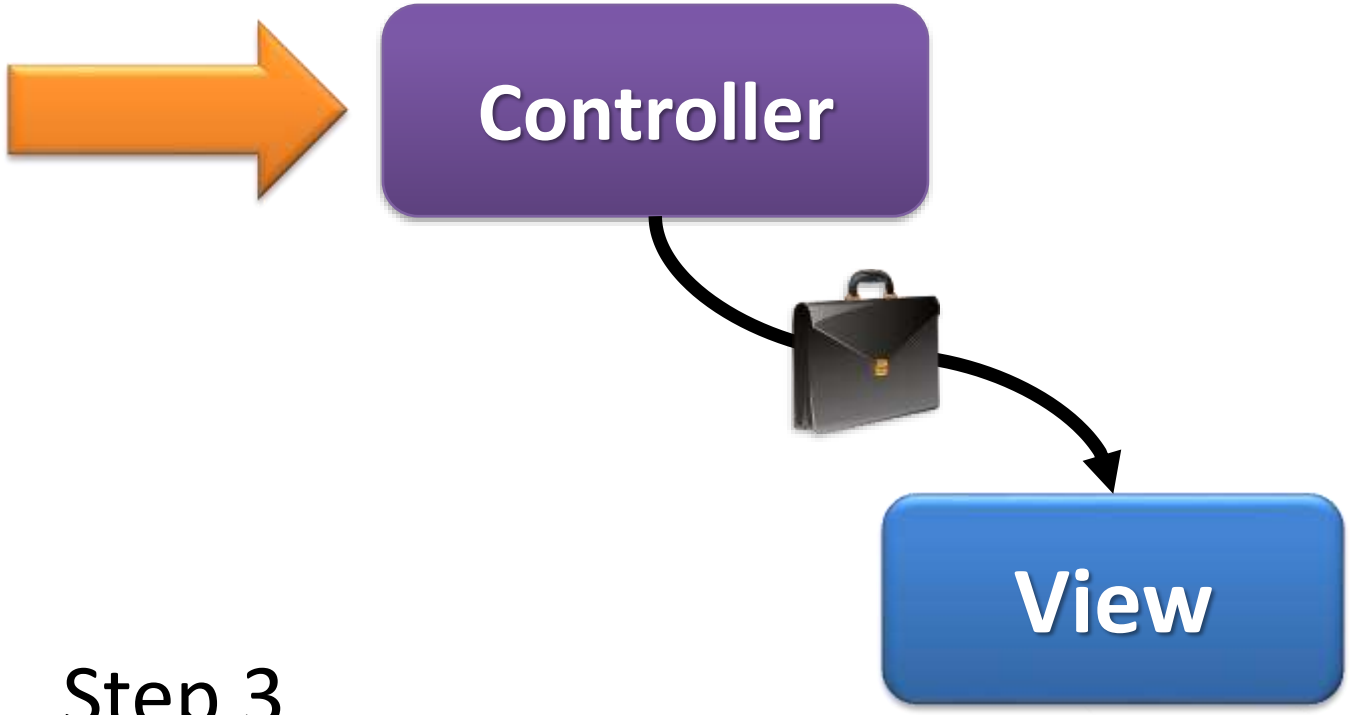
Incoming request directed to **Controller**

MVC Flow

Step 2

Controller processes request and forms a data Model

MVC Flow



Step 3

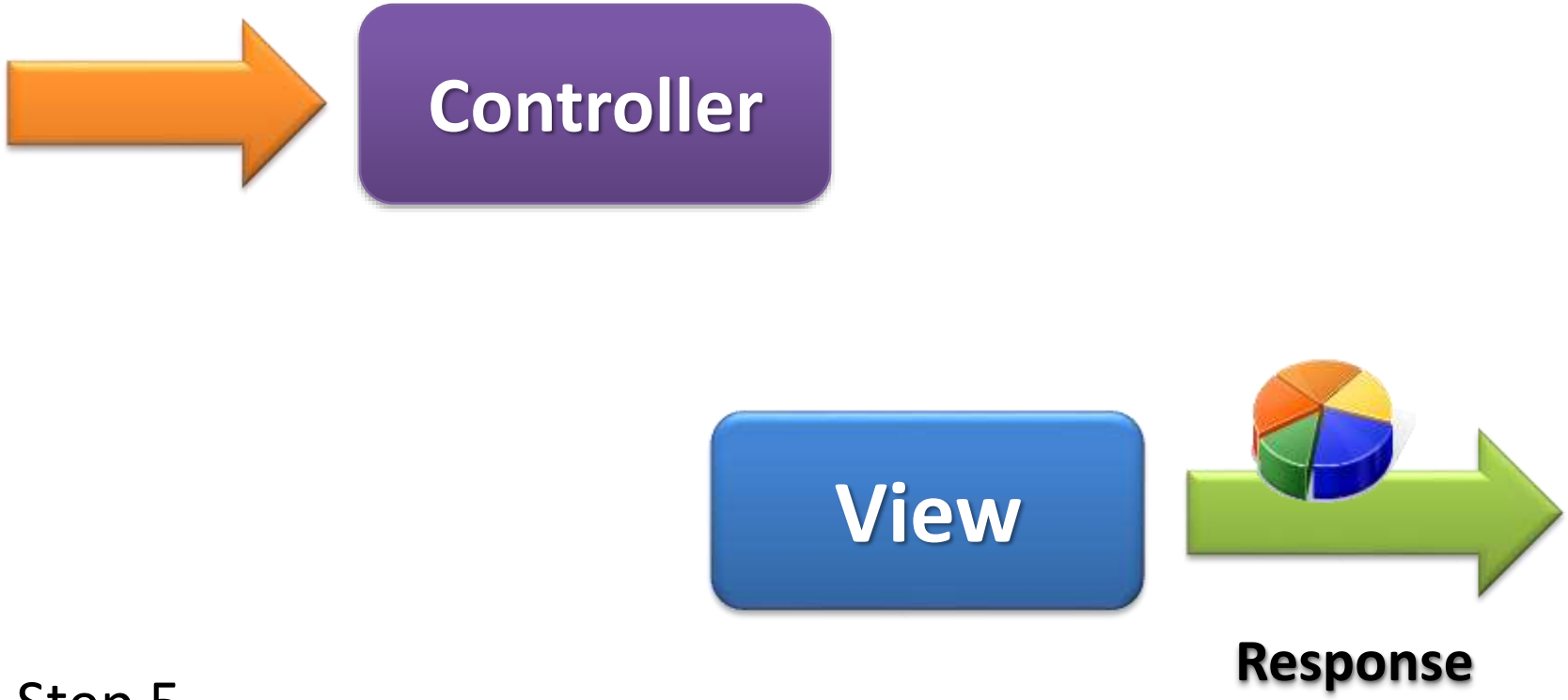
Model is passed to **View**

MVC Flow

Step 4

View transforms Model into appropriate output format

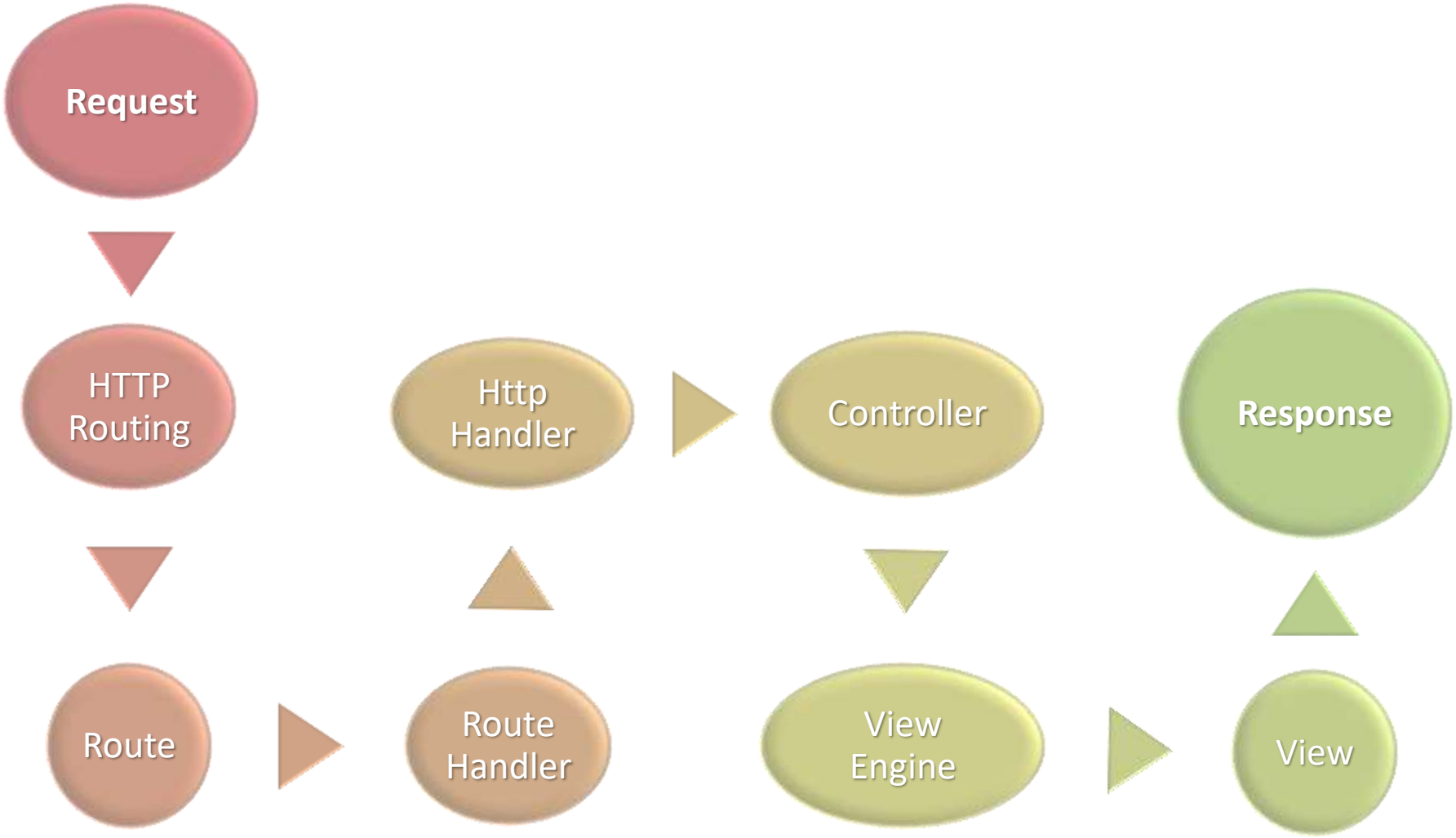
MVC Flow



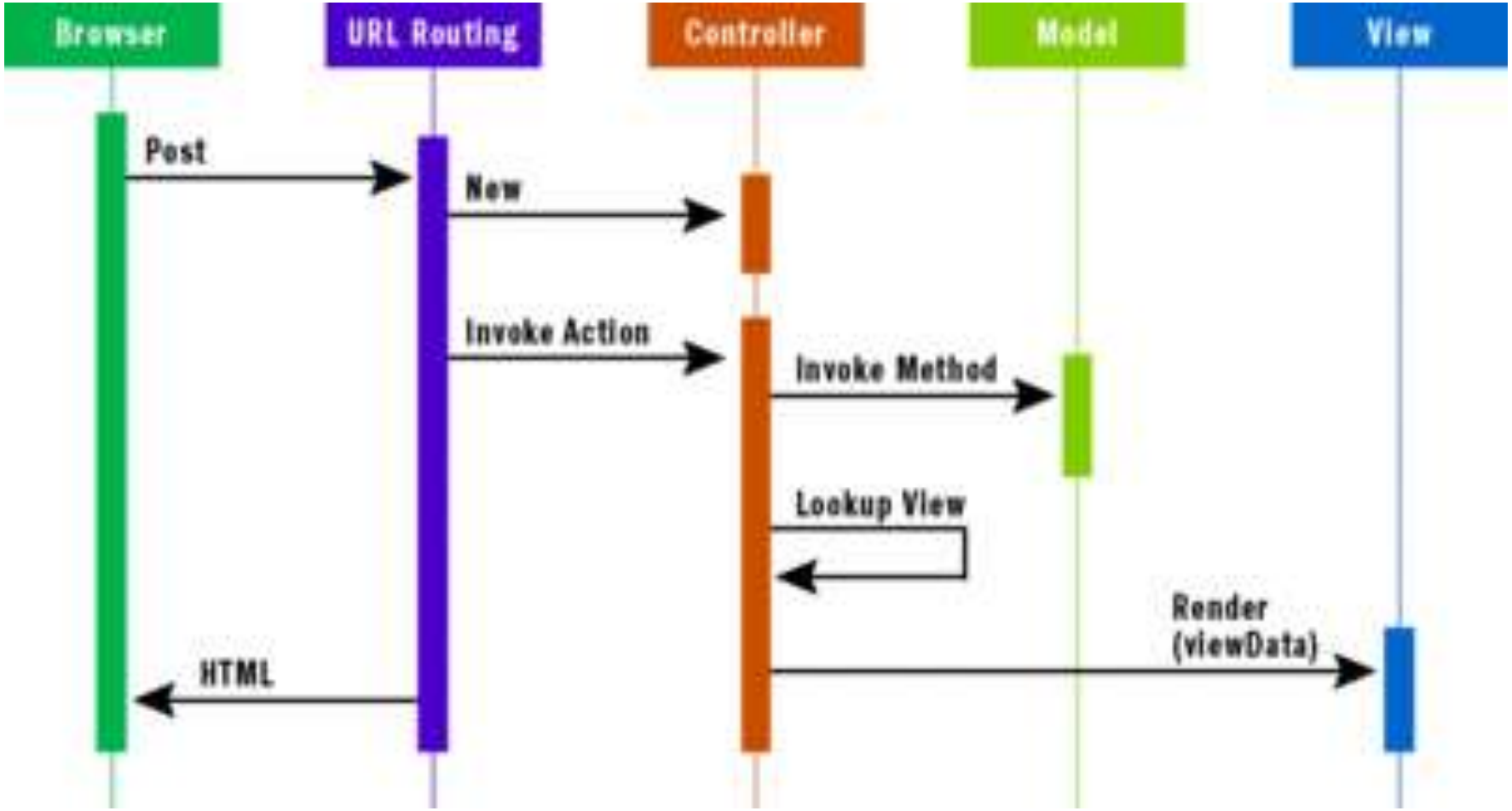
Step 5

Response is rendered

Request Flow – in more detail



MVC App Execution

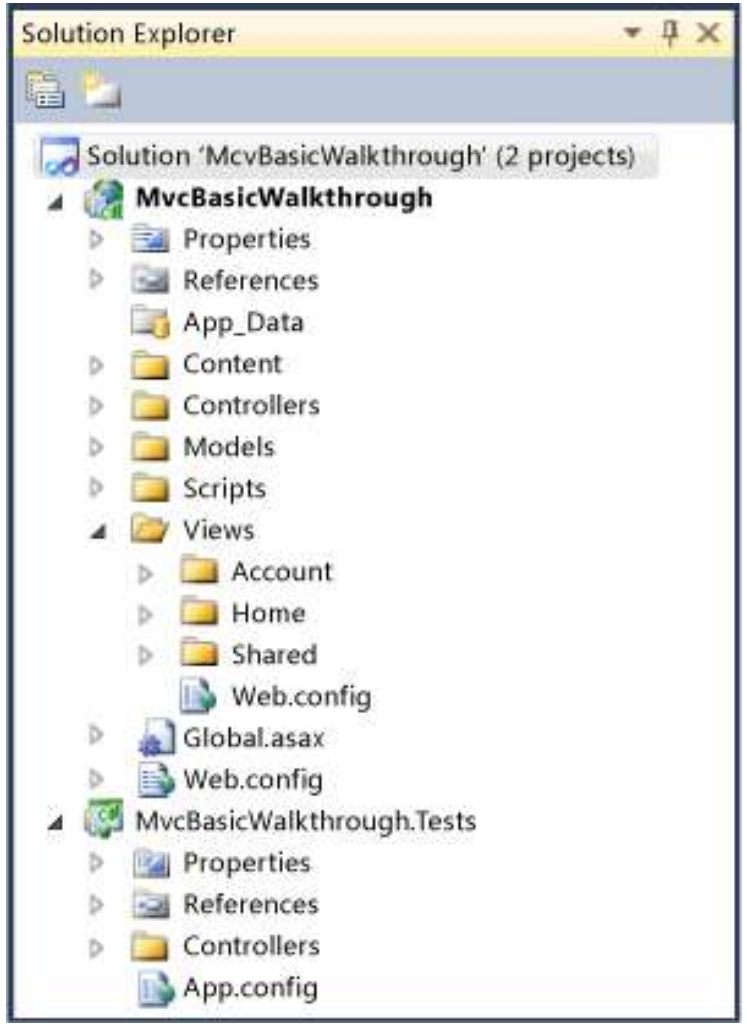


ASP .NET MVC Framework Components

- Models
  - Business/domain logic
  - Model objects, retrieve and store model state in a persistent storage (database).
- Views
  - Display application's UI
  - UI created from the model data
- Controllers
  - Handle user input and interaction
  - Work with model
  - Select a view for rendering UI

ASP .NET App Structure

- No Postback interaction!
- All user interactions routed to a controller
- No view state and page lifecycle events



BUILDING VIEW PAGES USING RAZOR LANGUAGE

RAZOR ENGINE

Razor Engine

- A new view-engine
- Optimized around HTML generation
- Code-focused templating approach

Razor Engine – Design goals

- Compact, Expressive and Fluid
- Easy to learn
- It is not a new language
- Works with any text editor
- Great Intellisense
- Unit-testable
  - Testing views without server, controllers…

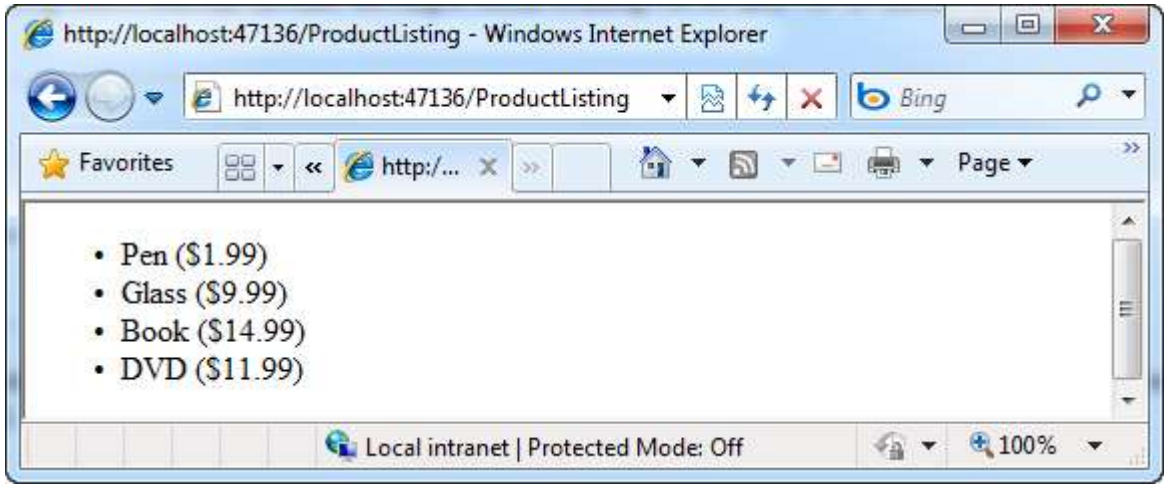
Loops and Nested HTML

Razor syntax

```
<ul id="products">

    @foreach(var p in products) {
        <li>@p.Name ($@p.Price)</li>
    }

</ul>
```

.aspx syntax

```
<ul id="products">

    <% foreach(var p in products) { %>
        <li><%=p.Name%> ($<%=p.Price%>)</li>
    <% } %>

</ul>
```



If Blocks and Multi-line Statements

IF statement

```
@if(products.Count == 0) {
    <p>Sorry - no products in this category</p>
} else {
    <p>We have a products for you!</p>
}
```

Multi-Token statement

```
<p>Your Message: @("Number is: " + number)
```

Multi-line statement

```
@{
    int number = 1;
    string message = "Number is" " + number;
}

<p>Your Message: @message</p>
```

Variables

Variables can span multiple server code blocks!

Layout/Master page

```
<!DOCTYPE html>
<html>
    <head>
        <title>Simple Site</title>
    </head>
    <body>

        <div id="header">
            <a href="/">Home</a>
            <a href="/About>About</a>
        </div>

        <div id="body">
            @RenderBody()
        </div>

    </body>
</html>
```

SiteLayout.cshtml

@RenderBody()
Including specific body content.

Content page

```
@{
    LayoutPage = "SiteLayout.cshtml";
}

<h1>About This Site</h1>

<p>
    This is some content that will make up the "about"
    page of our web-site.  We'll use this in conjunction
    with a layout template.  The content you are seeing here
    comes from the Home.cshtml file.
</p>
<p>
    And obviously I can have code in here too. Here is the
    current date/time: @DateTime.Now
</p>
```

Explicitly setting LayoutPage property.

```
<!DOCTYPE html>
<html>
    <head>
        <title>Simple Site</title>
    </head>
    <body>

        <div id="header">
            <a href="/">Home</a>
            <a href="/About">About</a>
        </div>

        <div id="body">

            <h1>About This Site</h1>

            <p>
                This is some content that will make up the "about"
                page of our web-site.  We'll use this in conjunction
                with a layout template.  The content you are seeing here
                comes from the Home.cshtml file.
            </p>
            <p>
                And obviously I can have code in here too. Here is the
                current date/time: 7/2/2010 2:53:24 PM
            </p>

        </div>

    </body>
</html>
```

Complete HTML page.

Master page – section overrides

```
<!DOCTYPE html>
<html>
    <head>
        <title>Simple Site</title>
    </head>
    <body>

        <div id="header">
            <a href="/">Home</a>
            <a href="/About">About</a>
        </div>

        <div id="left-menu">
            @RenderSection("menu", optional:true)
        </div>

        <div id="body">
            @RenderBody()
        </div>

        <div id="footer">
            @RenderSection("footer", optional:true)
        </div>

    </body>
</html>
```

This section is optional.

This section is optional.

Re-usable “HTML Helpers”

- Methods that can be invoked within code-blocks
- Encapsulate generating HTML
- Implemented using pure code
- Work with Razor engine

```
<fieldset>
    <legend>Edit Product</legend>

    <div>
        @Html.LabelFor(m => m.ProductID)
    </div>
    <div>
        @Html.TextBoxFor(m => m.ProductID)
        @Html.ValidationMessageFor(m => m.ProductID)
    </div>
</fieldset>
```

Built-in HTML helper

Define own HTML helpers

@helper declarative syntax

Helper's parameters (full language and ebugging support)

```
@helper ProductListing(List<Product> products) {

    <ul id="products">

        @foreach(var p in products) {
            <li>@p.Name ($@p.Price)</li>
        }

    </ul>
}
```

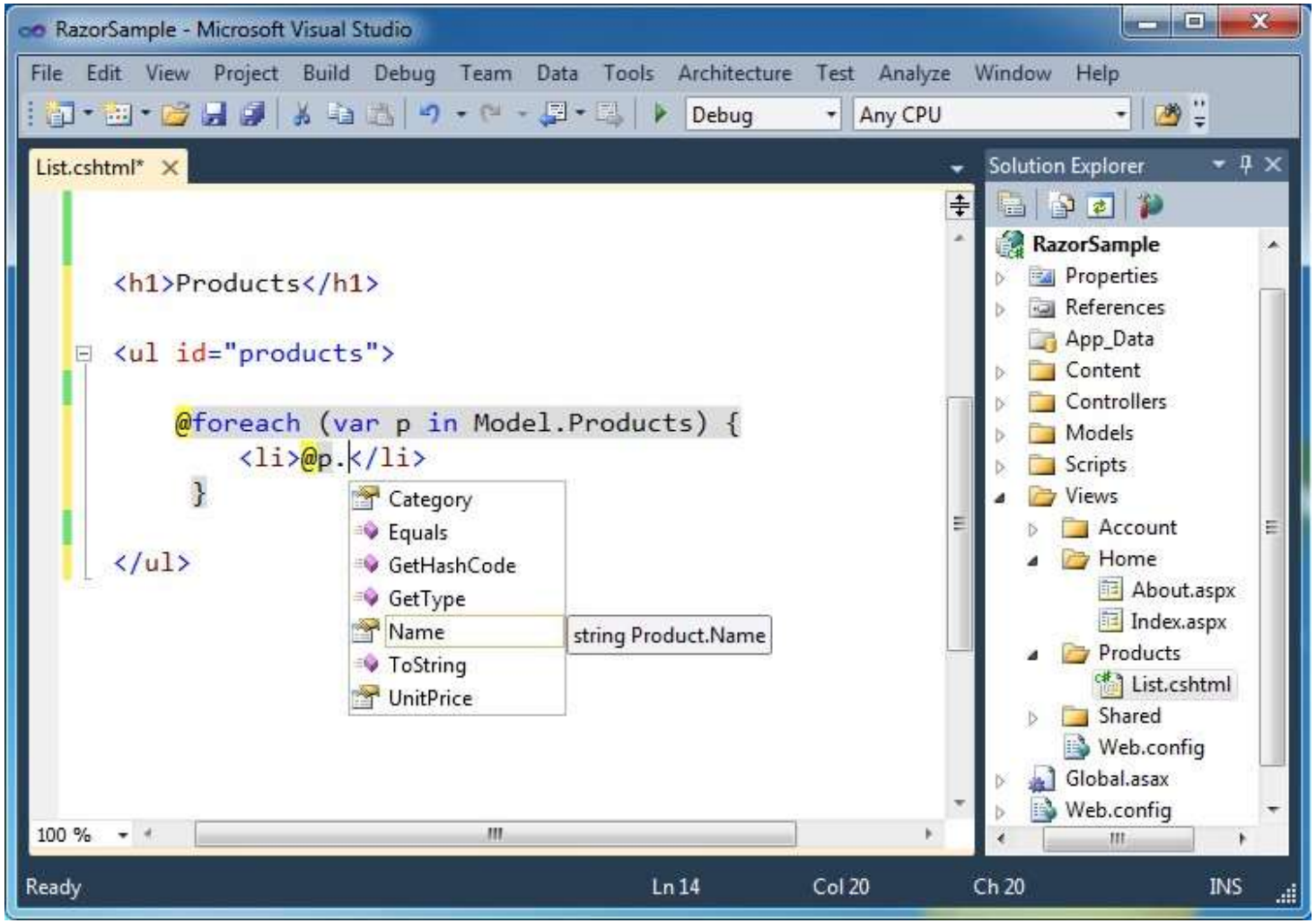
HTML Helper definition

HTML Helper should be placed to Views\Helper directory.

```
<body>

    <h1>Here are My Products</h1>

    <div>
        @ProductListing(Model.Products)
    </div>

</body>
```

HTML Helper Invocation

Visual Studio support

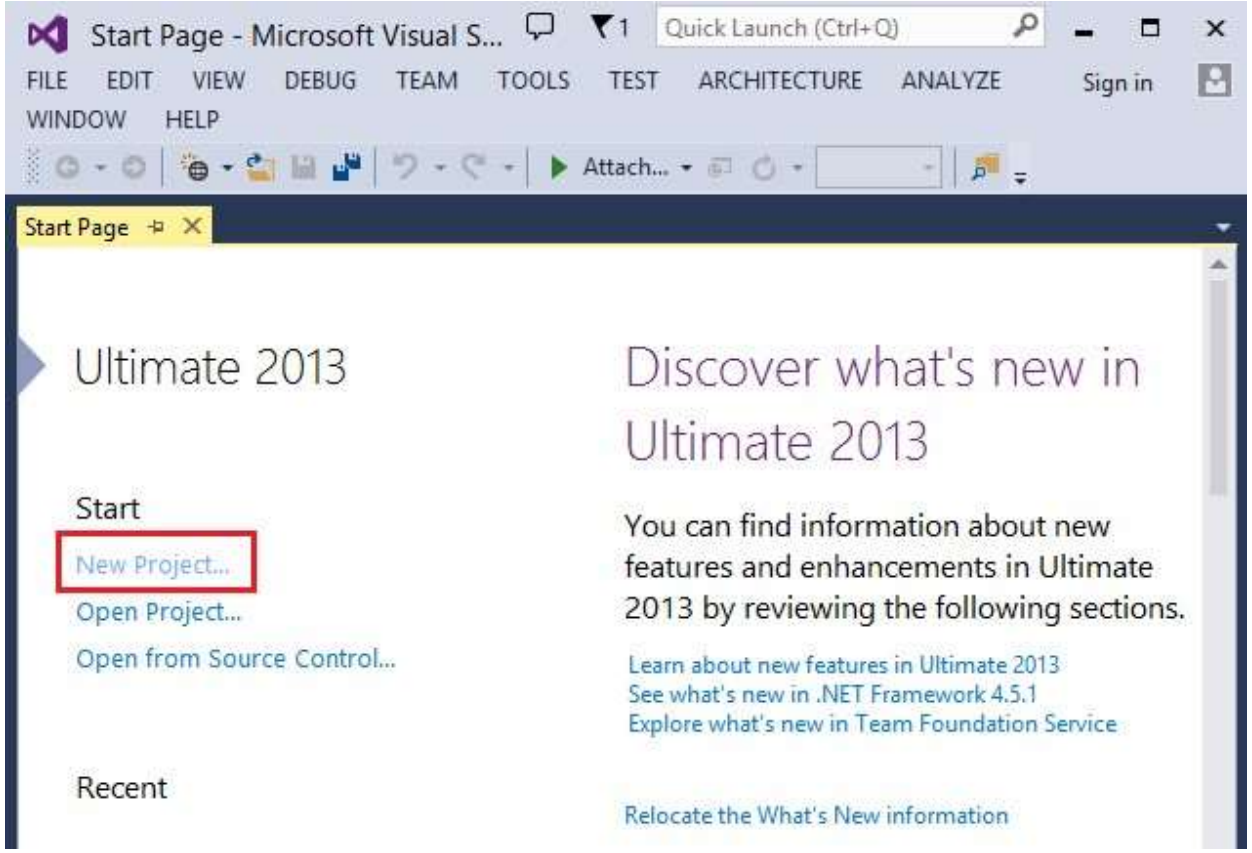


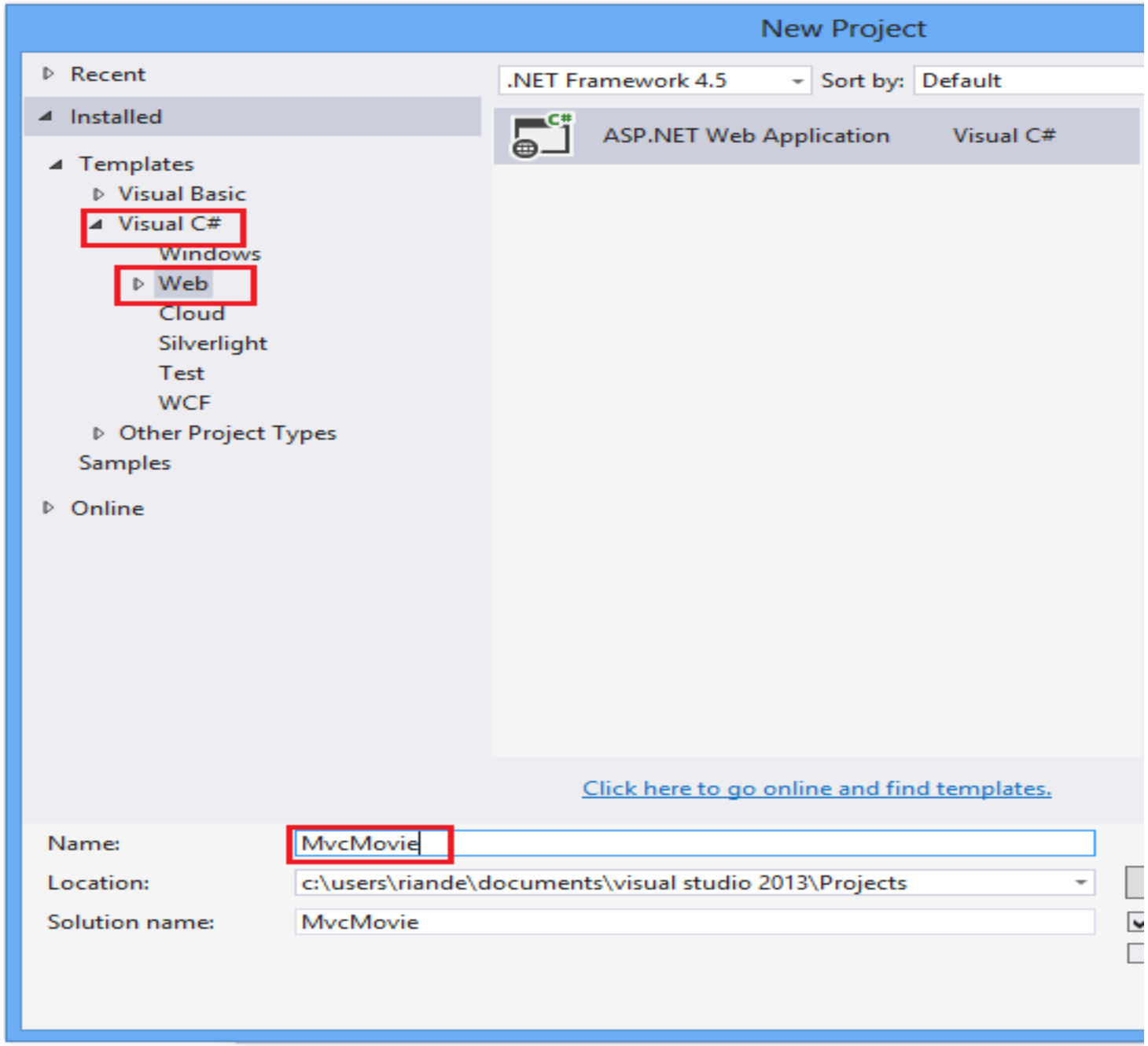
Razor – Summary

- A good new view engine
- Code-focused templating
- Fast and expressive
- Compact syntax
- Integrated with C# and VB

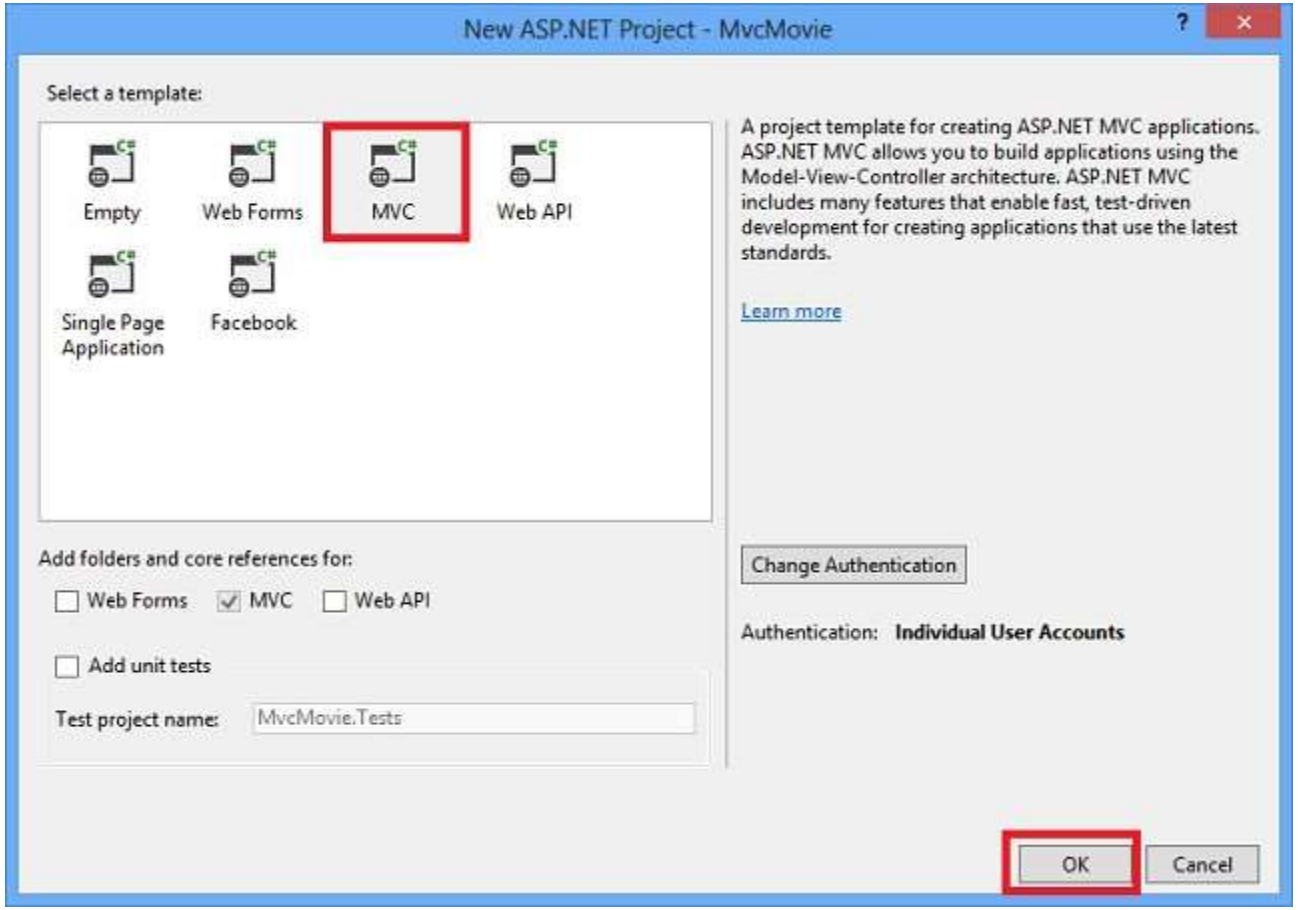
CREATING ASP .NET MVC APPLICATION

New Project …

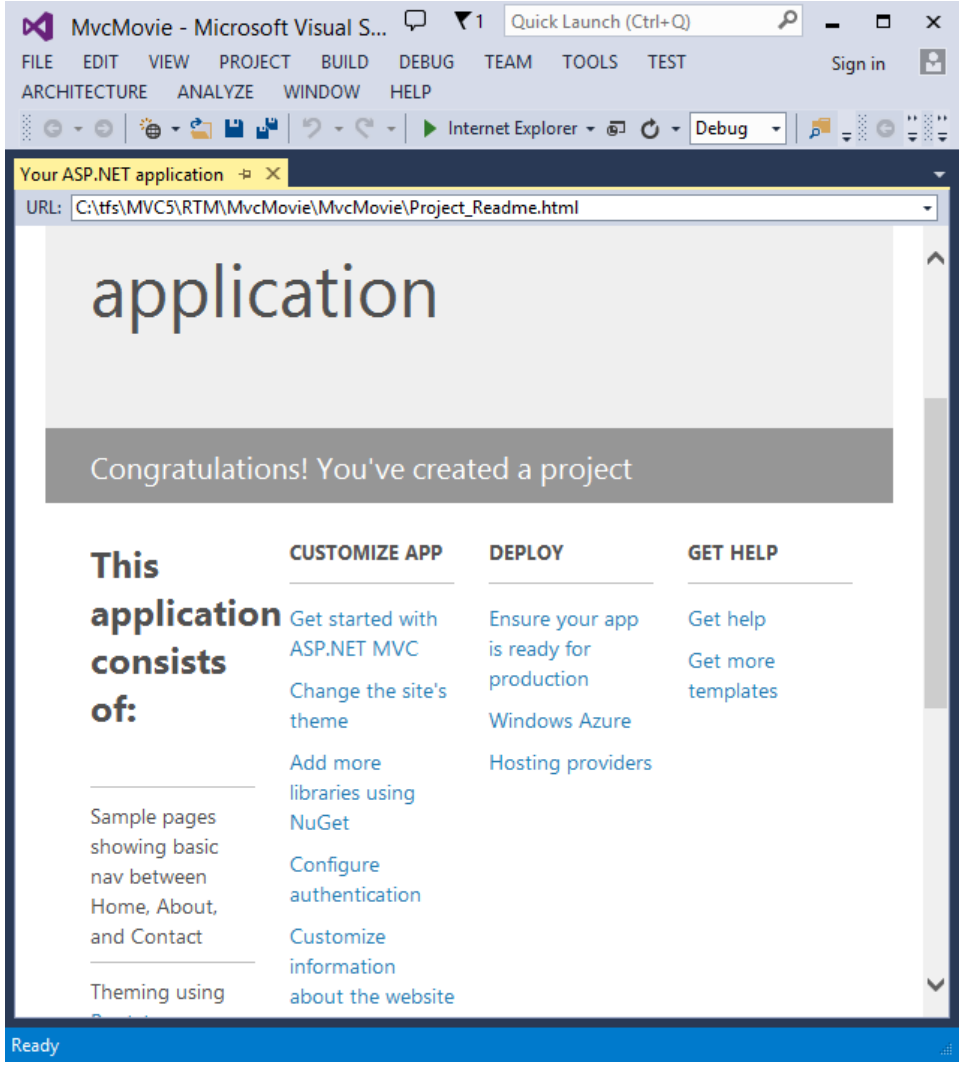

Start Page - Microsoft Visual S...
Quick Launch (Ctrl+Q)
FILE EDIT VIEW DEBUG TEAM TOOLS TEST ARCHITECTURE ANALYZE
Sign in
WINDOW HELP
Attach...
Start Page
Ultimate 2013
Start
New Project...
Open Project...
Open from Source Control...
Recent
Discover what's new in Ultimate 2013
You can find information about new features and enhancements in Ultimate 2013 by reviewing the following sections.
Learn about new features in Ultimate 2013
See what's new in .NET Framework 4.5.1
Explore what's new in Team Foundation Service
Relocate the What's New information


New Project
Recent
Installed
Templates
Visual Basic
Visual C#
Windows
Web
Cloud
Silverlight
Test
WCF
Other Project Types
Samples
Online
.NET Framework 4.5
Sort by:
Default
ASP.NET Web Application
Visual C#
Click here to go online and find templates.
Name:
MvcMovie
Location:
c:\users\riande\documents\visual studio 2013\Projects
Solution name:
MvcMovie

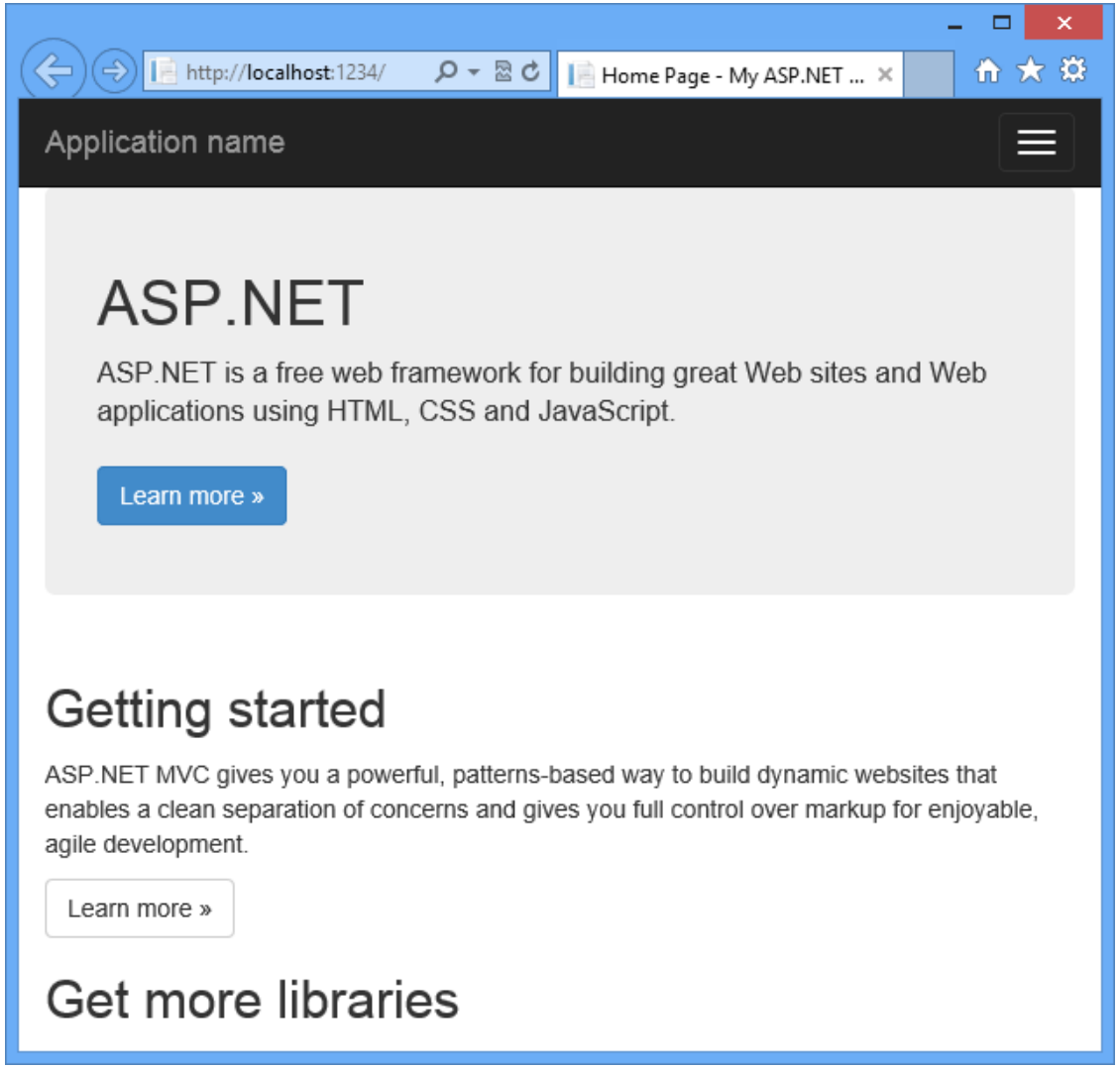
Select the project template



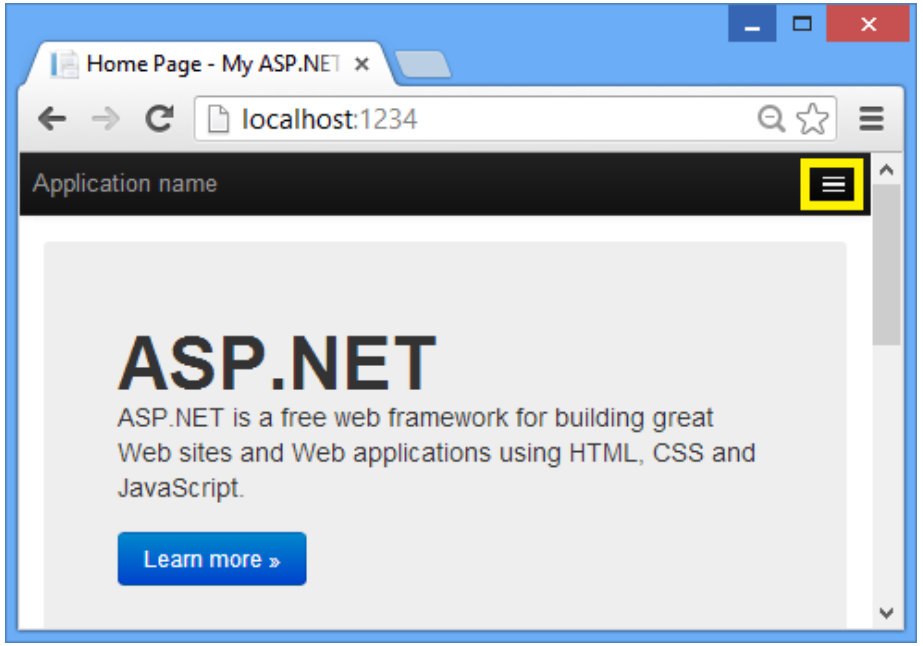
ASP .NET MVC App Home page

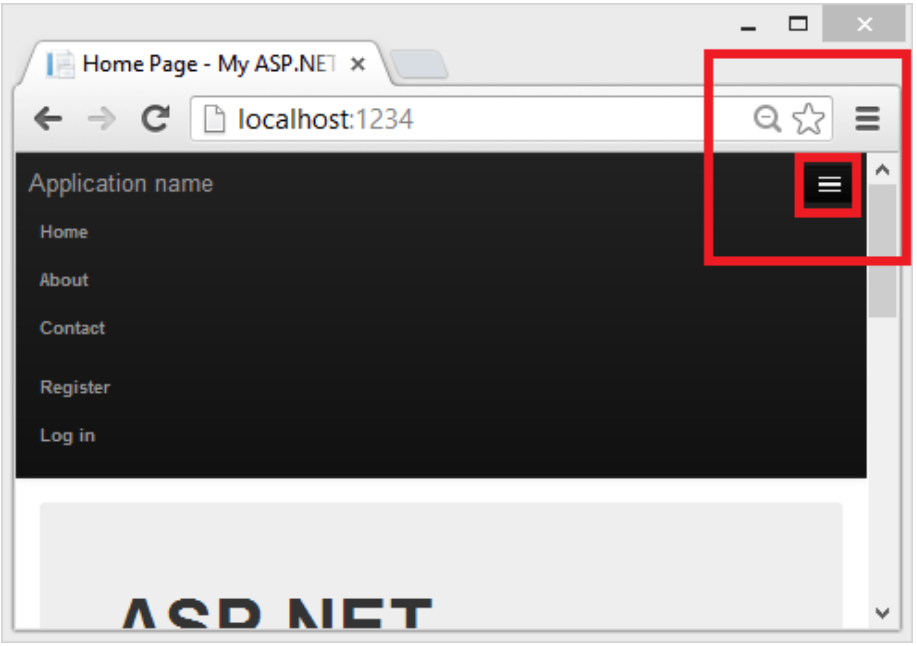


Run the application…



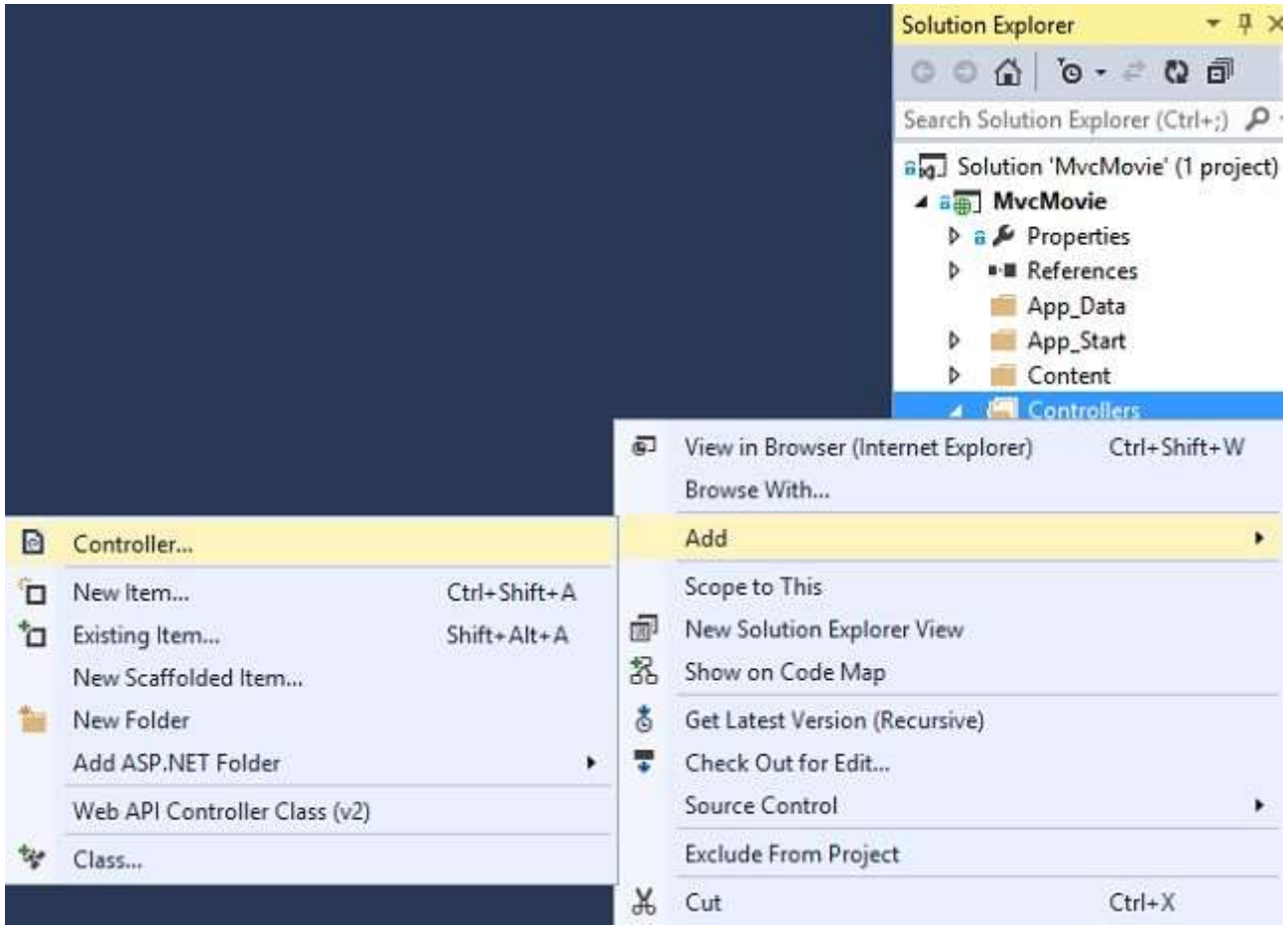
Expand the default App menu



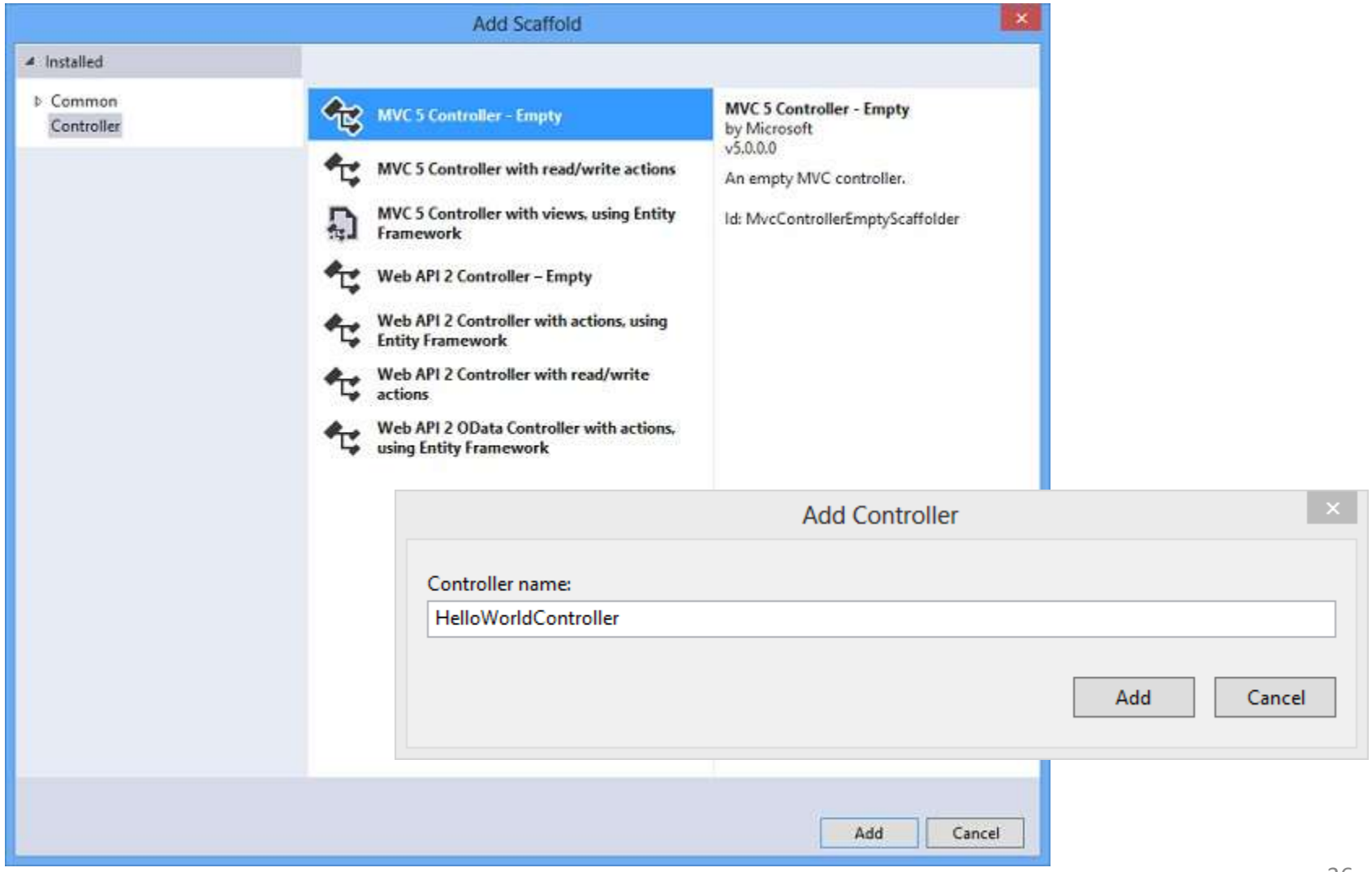


ADDING CONTROLLER

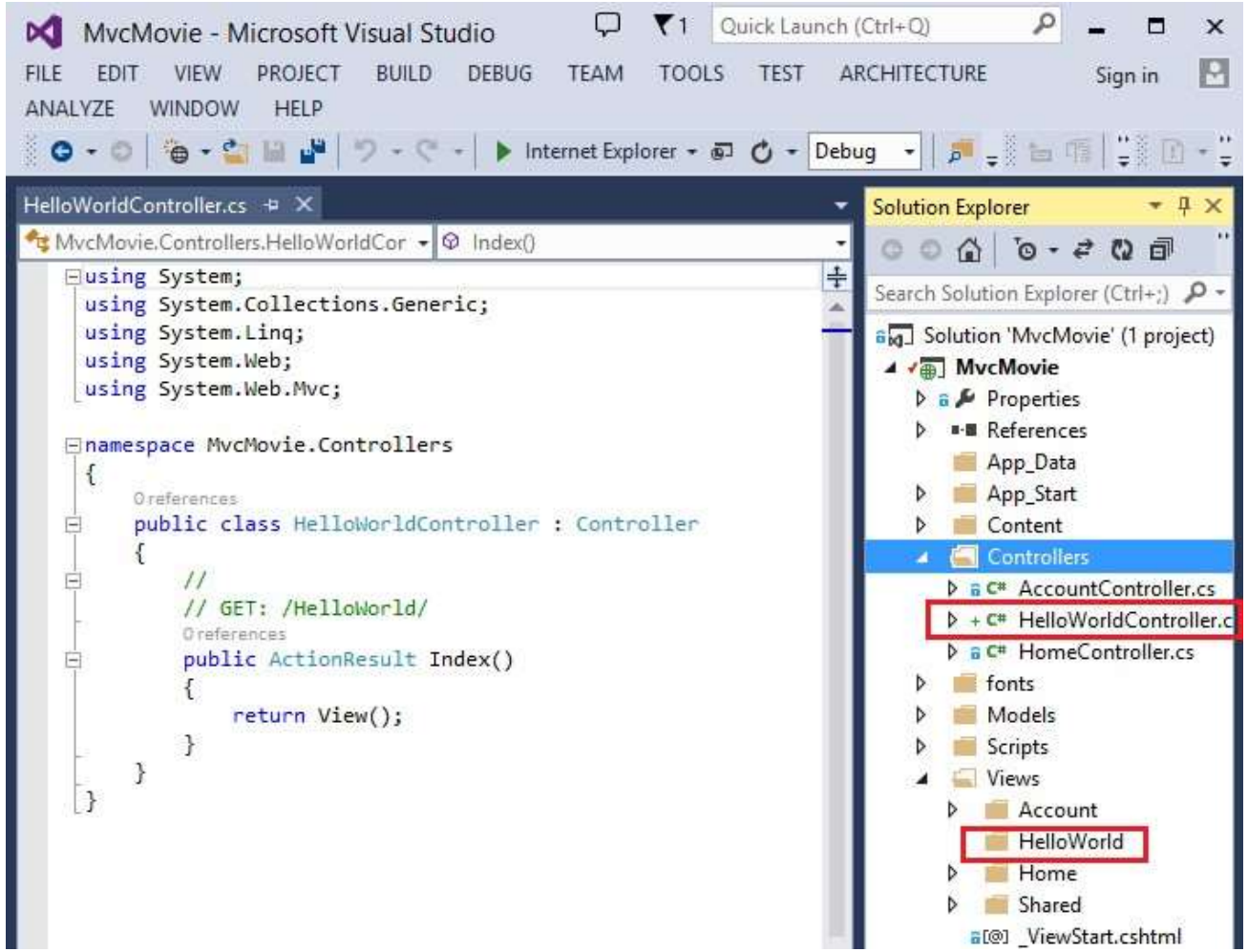
Adding controller



Adding controller (cont.)



Adding a controller (cont.)



Testing the controller

```
using System.Web;
using System.Web.Mvc;

namespace MvcMovie.Controllers
{
    public class HelloWorldController : Controller
    {
        //
        // GET: /HelloWorld/

        public string Index()
        {
            return "This is my <b>default</b> action...";
        }

        //
        // GET: /HelloWorld/Welcome/

        public string Welcome()
        {
            return "This is the Welcome action method...";
        }
    }
}
```

Mapping controller

- Controller selection based on URL
- Default URL routing logic: /[Controller]/[ActionName]/[Parameters]
- Format for routing in *App_Start/RouteConfig.cs*

```
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 }
    );
}
```

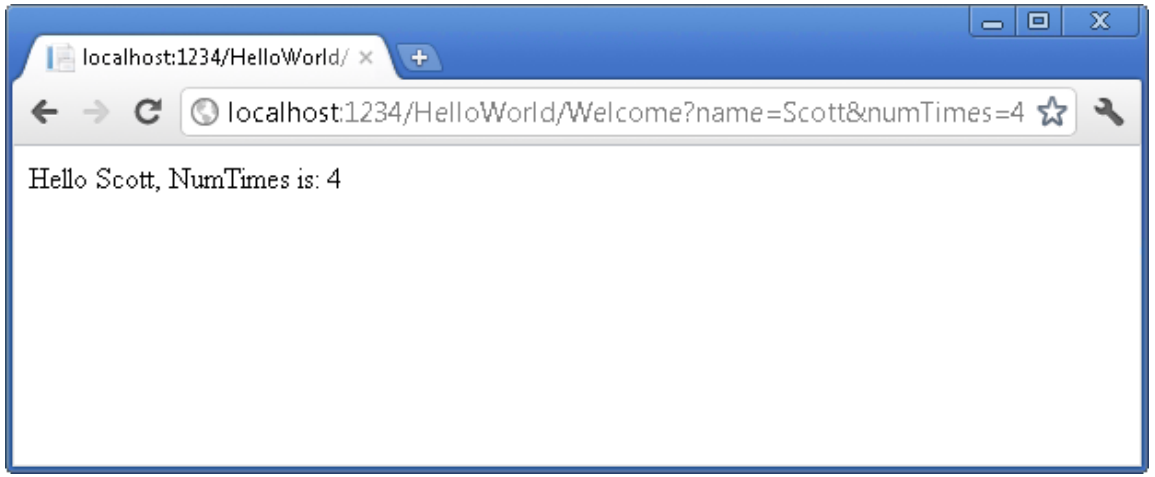
URL routing

- Webapp URL without URL segments => HomeController::Index()
- Index() – default method of a controller
- /HelloWorld => HelloWorldController
- /HelloWorld/Index => HelloWorldController::Index()
- http://webapp:port/HelloWorld/Welcome => HelloWorldController::Welcome()

Parameters

- /HelloWorld/Welcome?name=Scott&numtimes=4
- Introducing 2 parameters to Welcome method
- Parameters passed as query strings!

```
public string Welcome(string name, int numTimes = 1) {
    return HttpUtility.HtmlEncode("Hello " + name + ", NumTimes is: " + numTimes);
}
```

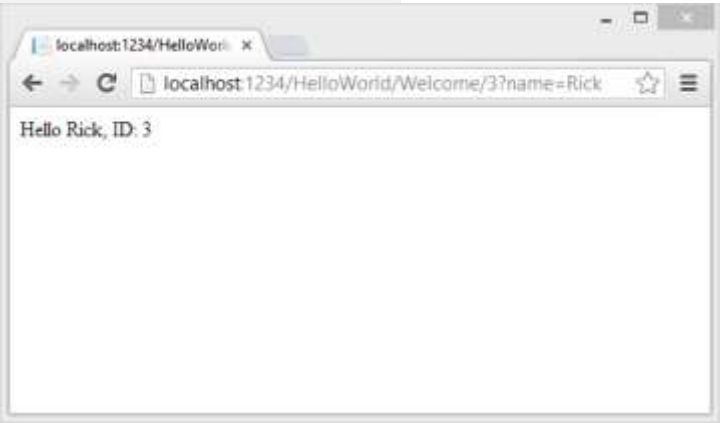


URL Parameters

- http://webapp/HelloWorld/Welcome/3?name=Rick

```
public string Welcome(string name, int ID = 1)
{
    return HttpUtility.HtmlEncode("Hello " + name + ", ID: " + ID);
}
```

Parameter ID matches URL specification in RegisterRoutes method.




```
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 }
    );
}
```

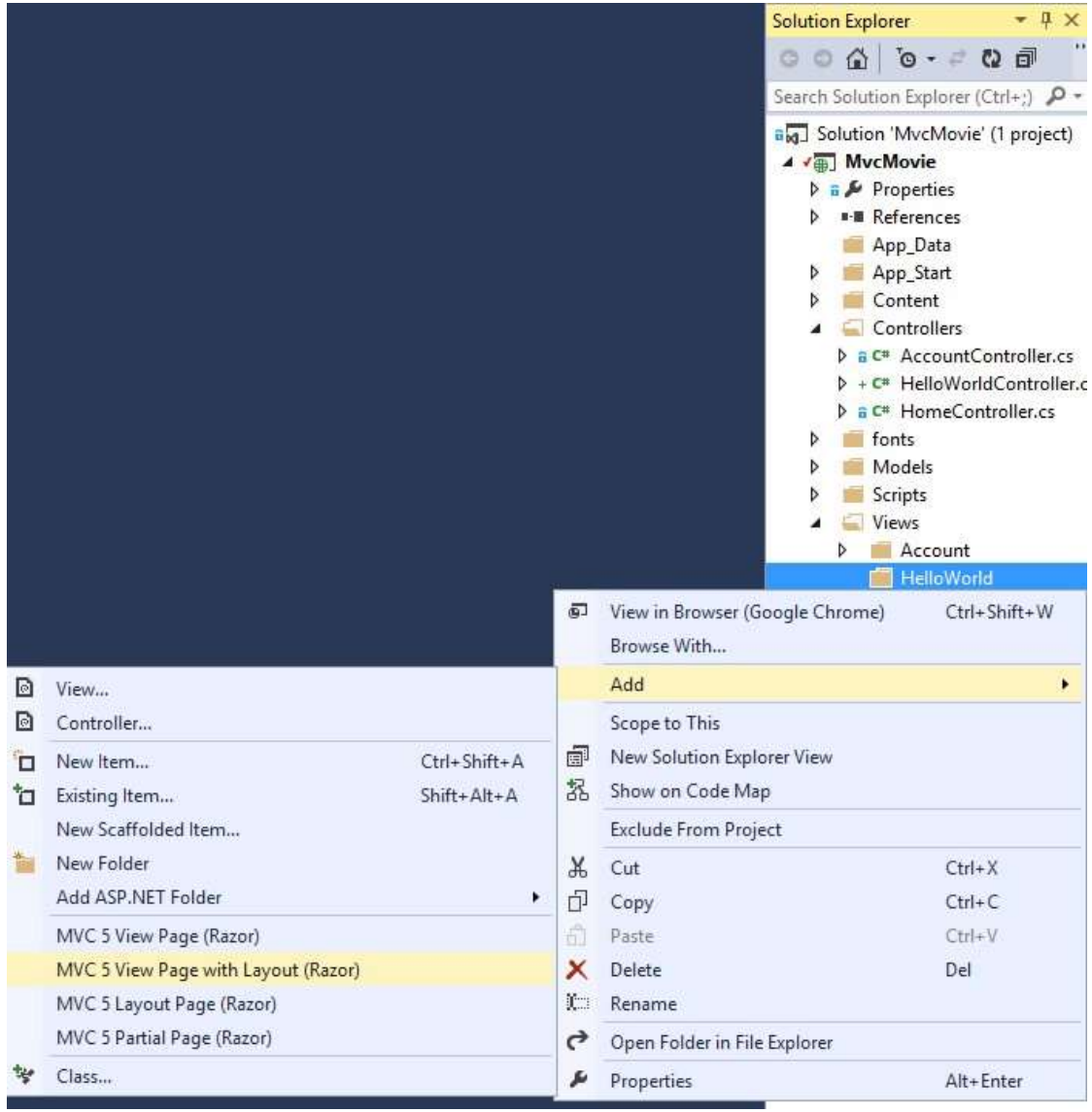
ADDING A VIEW

Views

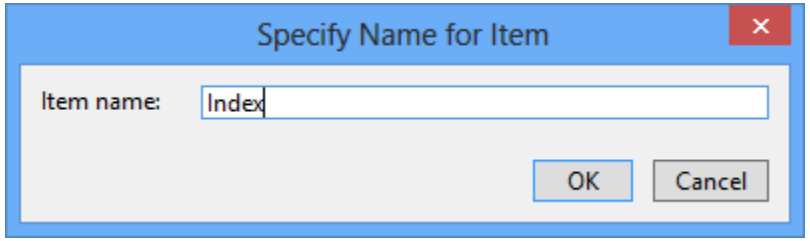
- Views created using Razor view engine
- Controller method returns View object
- Controller method return type is ActionResult
- Common pattern: all view pages share the same master layout page

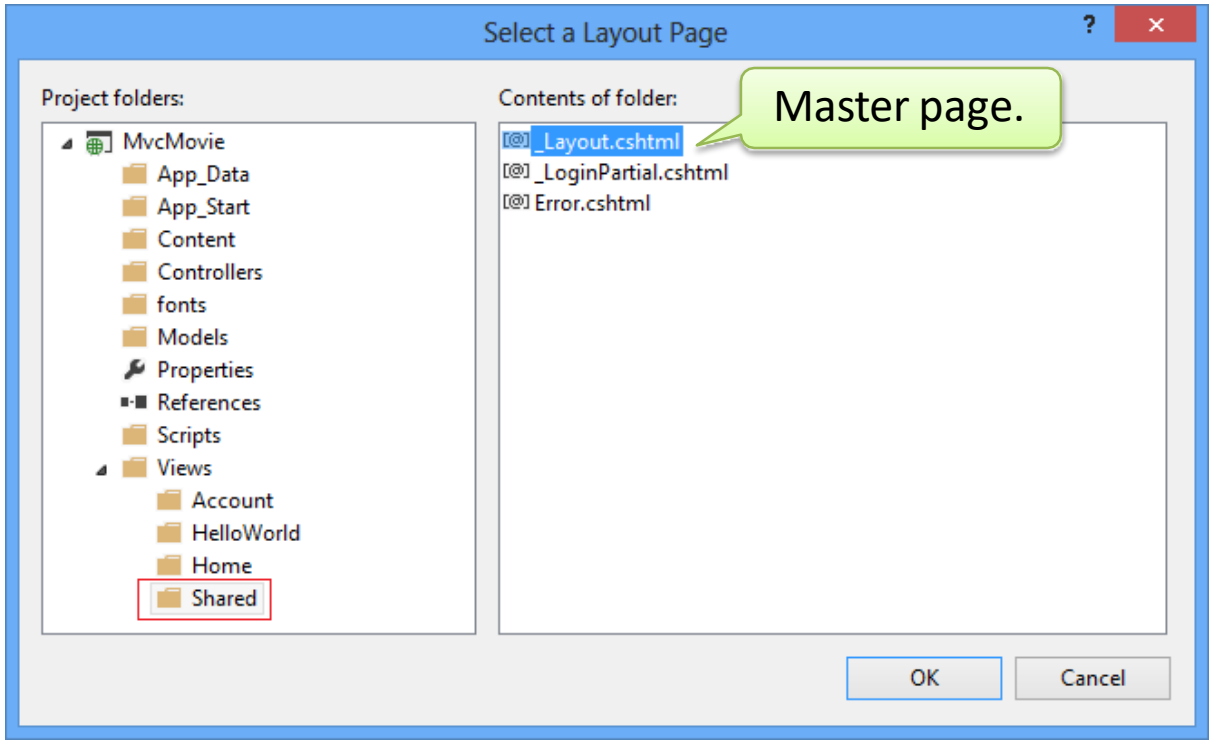
```
public ActionResult Index()
{
    return View();
}
```

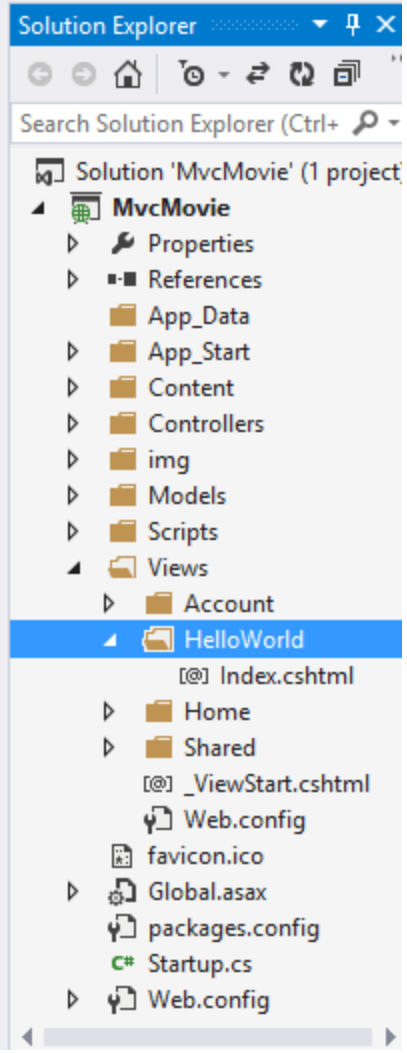
Create View page



Create View page



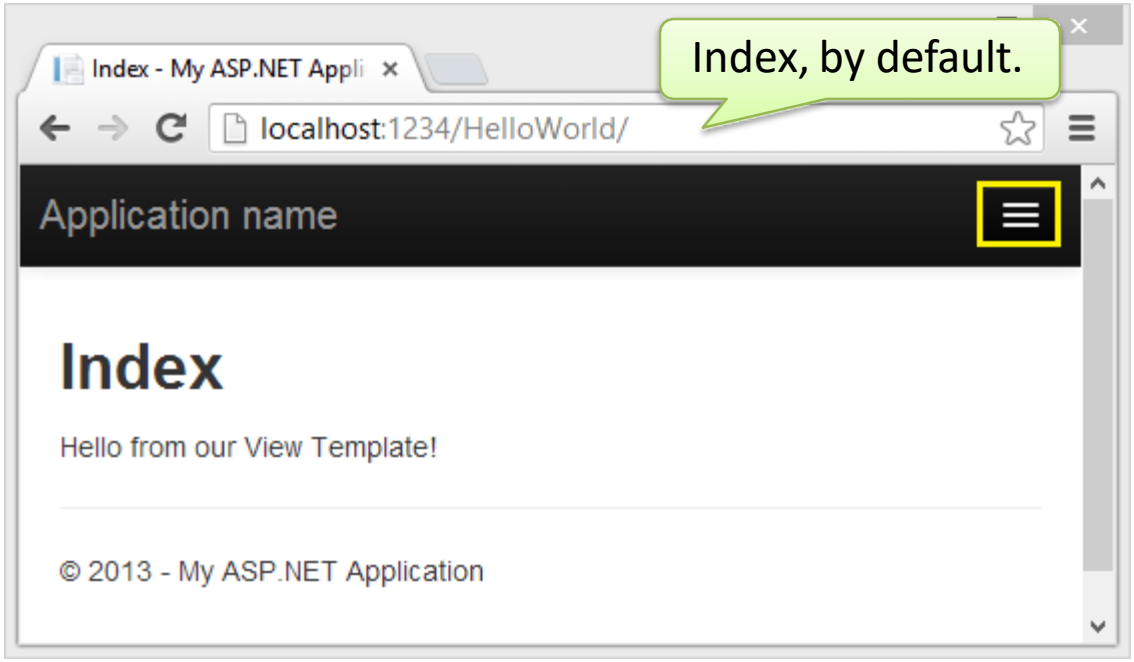




Implementing View page

Selected master page.

```
@{
    Layout = "~/Views/Shared/_Layout.cshtml";
}

@{
    ViewBag.Title = "Index";
}

<h2>Index</h2>

<p>Hello from our View Template!</p>
```



Index, by default.

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

Change controller's method signature. The method retures a view object: searches a view file that is named the same as the method (Index.cshtml).

ViewBag

- Pass data between view template and layout view file
- ViewBag is a dynamic object (has no defined properties)

View template file.

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

Layout view file.

```
@{
    ViewBag.Title = "Movie List";
}

<h2>My Movie List</h2>

<p>Hello from our View Template!</p>
```

Passing data from Controller to View

- View is used for data presentation
- Controller must provide a view with the data
- One approach: using ViewBag
  - Controller puts data to ViewBag,
  - View reads ViewBag and renders the data
  - No data binding!
- Alternative approach: the view model
  - Strongly typed approach

Passing data from Controller to View

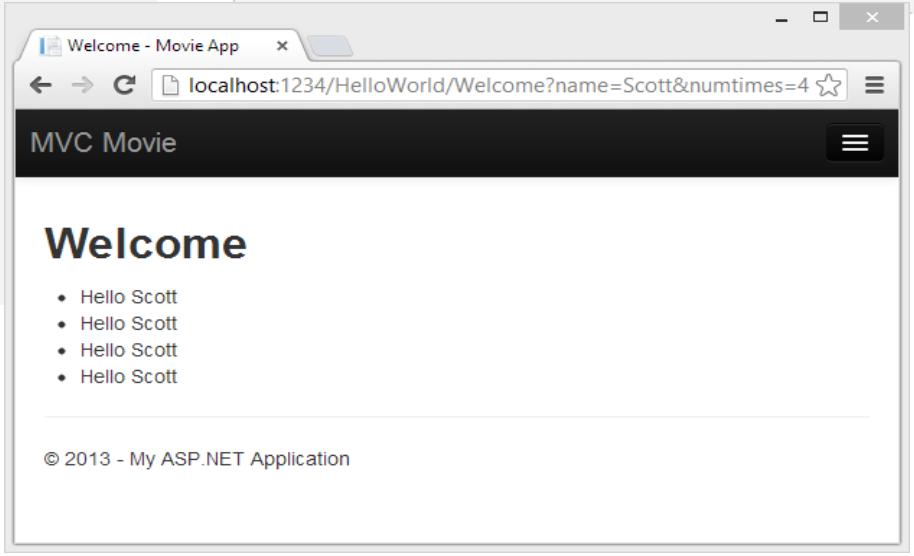
Controller

```
using System.Web;
using System.Web.Mvc;

namespace MvcMovie.Controllers
{
    public class HelloWorldController : Controller
    {
        public ActionResult Index()
        {
            return View();
        }

        public ActionResult Welcome(string name, int numTimes = 1)
        {
            ViewBag.Message = "Hello " + name;
            ViewBag.NumTimes = numTimes;

            return View();
        }
    }
}
```

Returns HelloWorldView object.

View

```
@{
    ViewBag.Title = "Welcome";
}

<h2>Welcome</h2>

<ul>
    @for (int i = 0; i < ViewBag.NumTimes; i++)
    {
        <li>@ViewBag.Message</li>
    }
</ul>
```

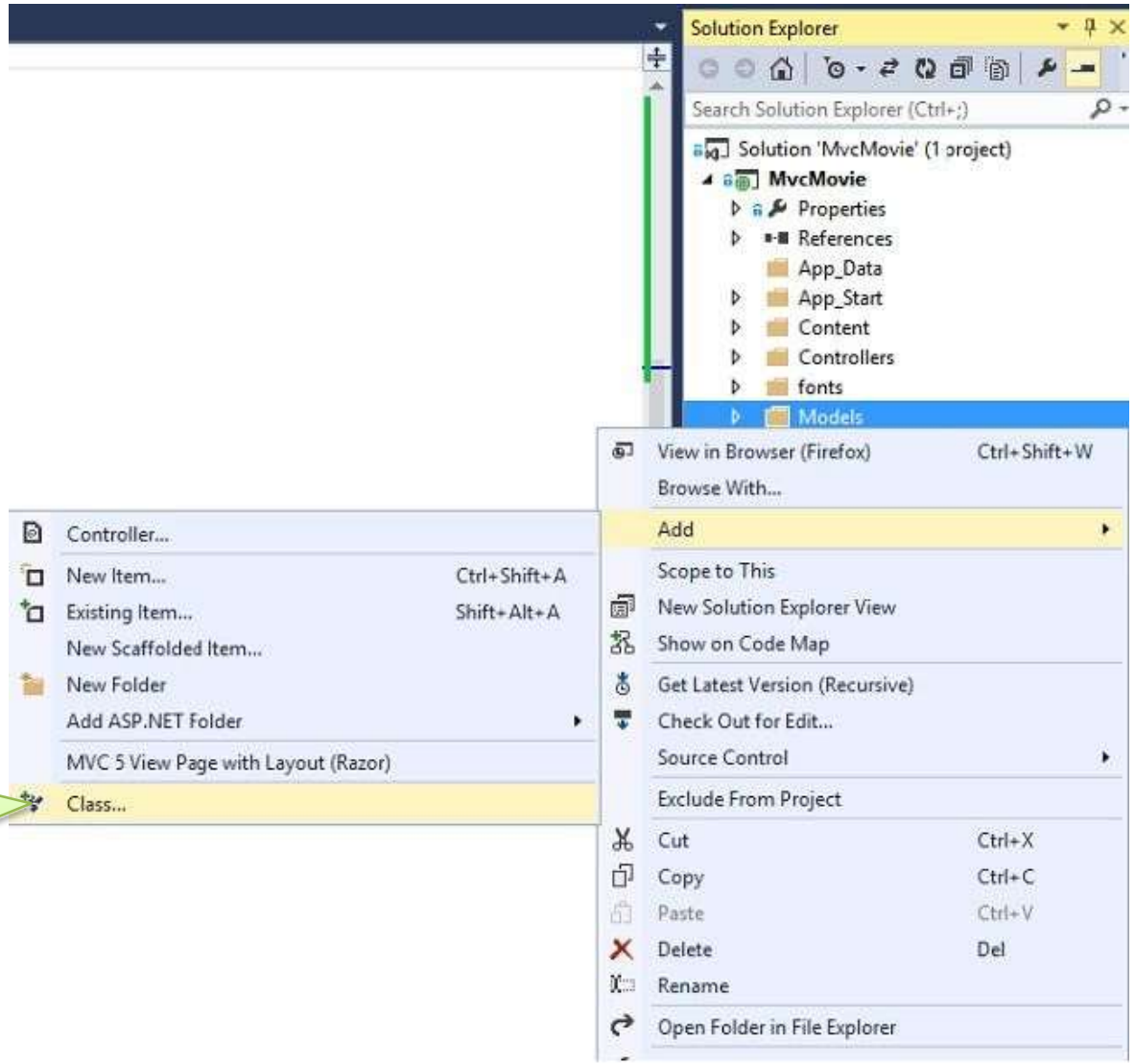


ADDING A MODEL

Model components

- Entity framework - data access technology
- “Code first” development paradigm
  (first code classes, then generate DB schema)
- “Database first” development paradigm
  define db schema first,
  then generate models, controllers and views

Adding a model class



Enter the class name, e.g. Movie.cs

Adding properties to a model class

```
using System;

namespace MvcMovie.Models
{
    public class Movie
    {
        public int ID { get; set; }
        public string Title { get; set; }
        public DateTime ReleaseDate { get; set; }
        public string Genre { get; set; }
        public decimal Price { get; set; }
    }
}
```

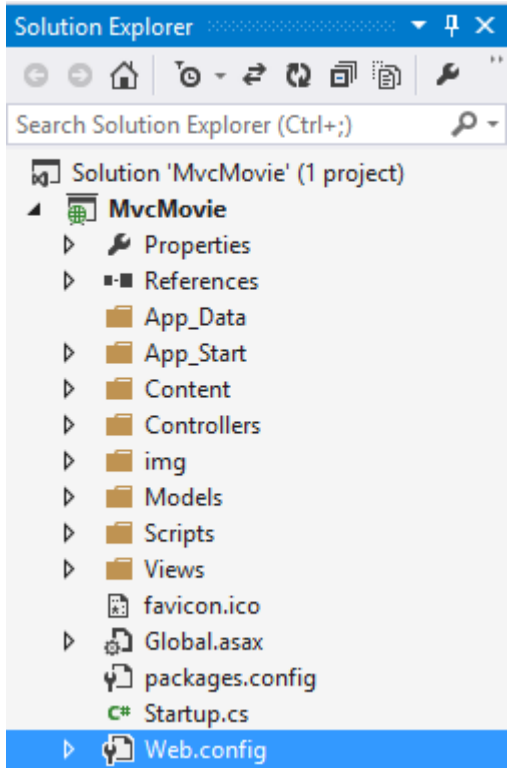
Adding a DbContext class

```
using System;
using System.Data.Entity;

namespace MvcMovie.Models
{
    public class Movie
    {
        public int ID { get; set; }
        public string Title { get; set; }
        public DateTime ReleaseDate { get; set; }
        public string Genre { get; set; }
        public decimal Price { get; set; }
    }

    public class MovieDBContext : DbContext
    {
        public DbSet<Movie> Movies { get; set; }
    }
}
```

EF namespace
DbContext
DbSet

EF database context
FETCH,
INSERT,
UPDATE

DB Connection string

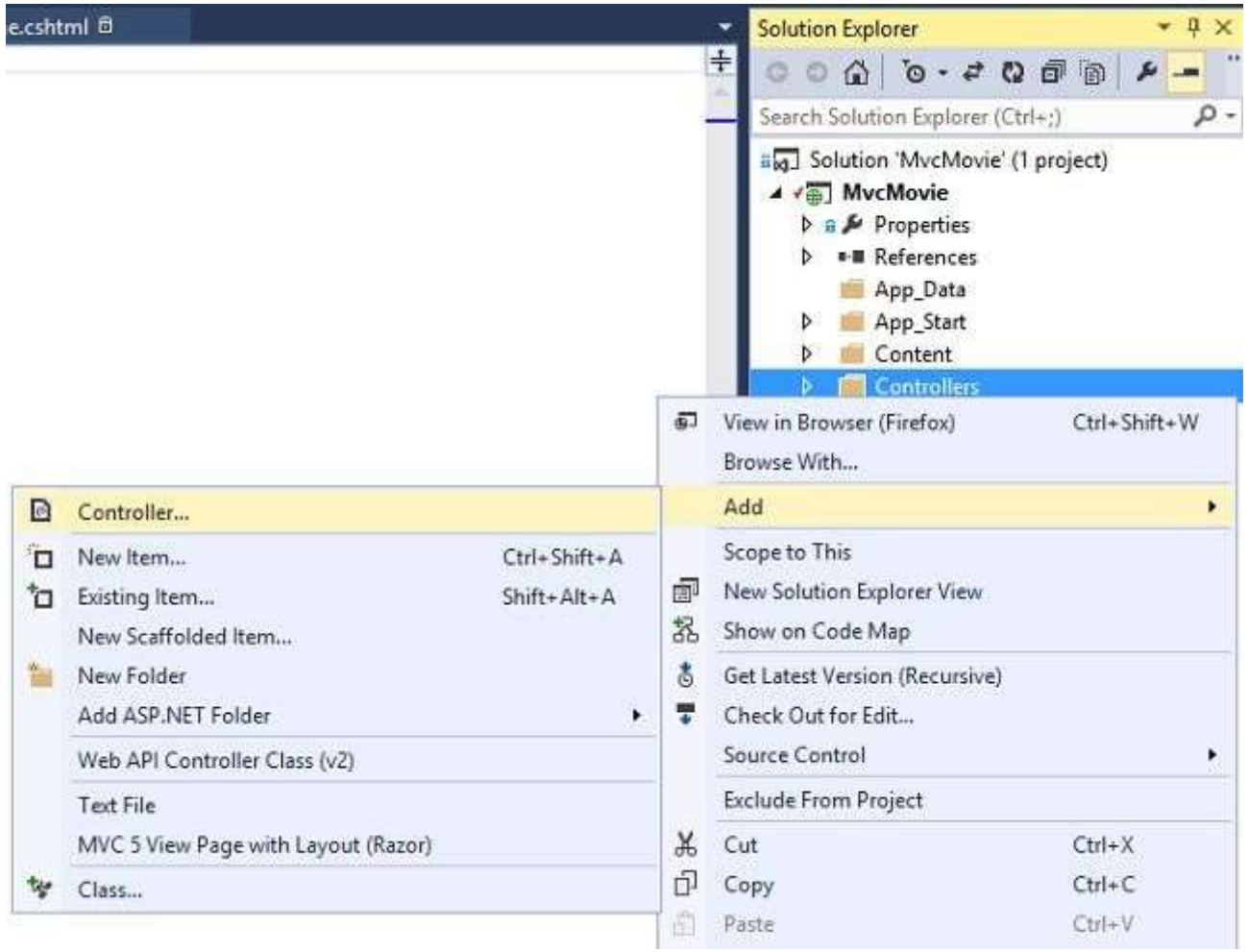



```
<?xml version="1.0" encoding="utf-8"?>
<!--
  For more information on how to configure your ASP.NET application, please visi
  http://go.microsoft.com/fwlink/?LinkId=301880
  -->
<configuration>
  <configSections>
    <!-- For more information on Entity Framework configuration, visit http://go
    <section name="entityFramework" type="System.Data.Entity.Internal.ConfigFile
  </configSections>
  <connectionStrings>
    <add name="DefaultConnection" connectionString="Data Source=(LocalDb)\v11.0;
      providerName="System.Data.SqlClient" />
  </connectionStrings>
  <appSettings>
    <add key="webpages:Version" value="3.0.0.0" />
    <add key="webpages:Enabled" value="false" />
    <add key="PreserveLoginUrl" value="true" />
    <add key="ClientValidationEnabled" value="true" />
    <add key="UnobtrusiveJavaScriptEnabled" value="true" />
  </appSettings>
  <system.web>
```

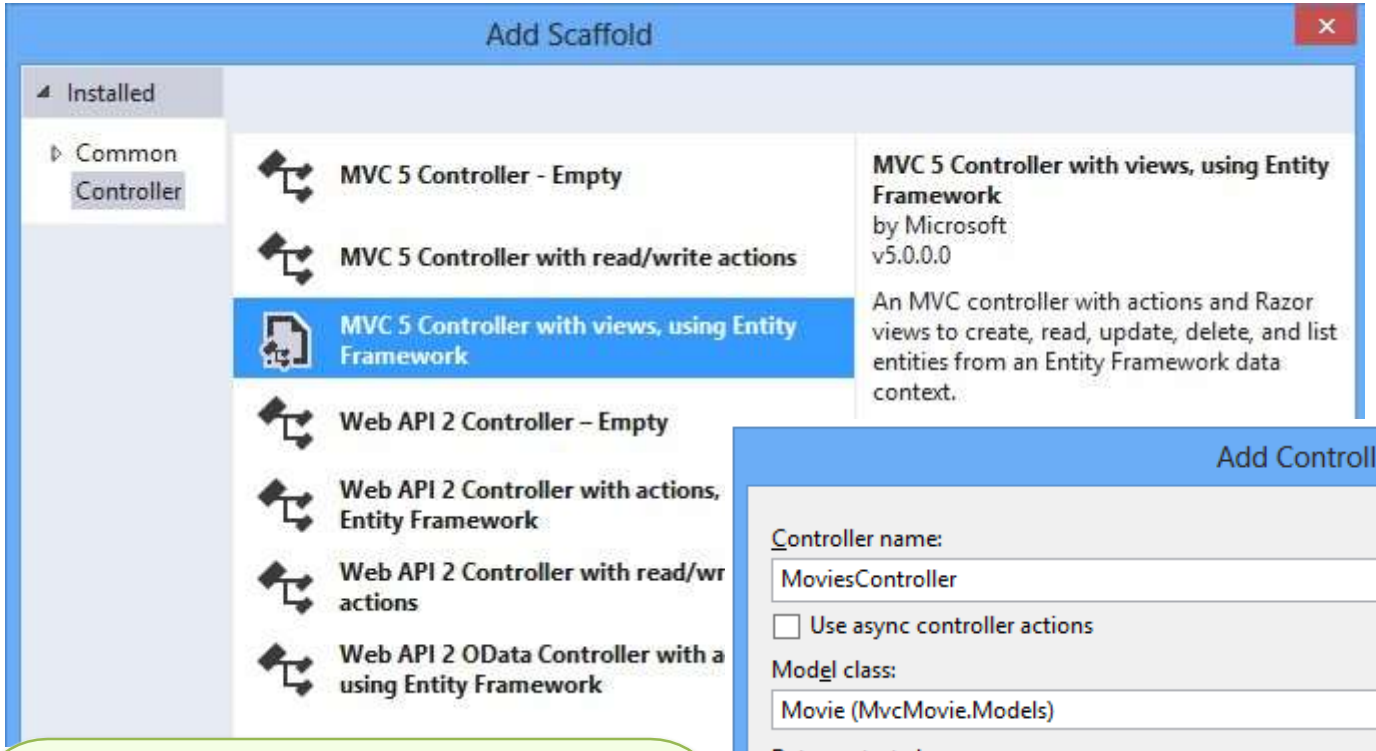
Separate connection string for each DbContex class

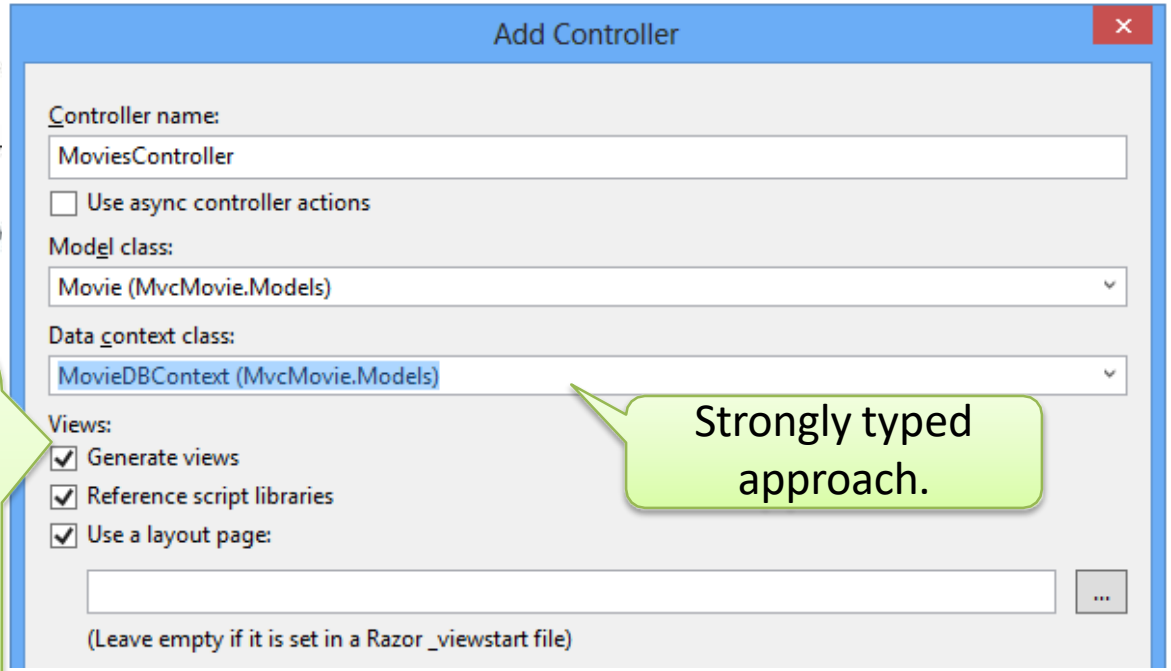
```
<connectionStrings>
    <add     name="MovieDBContext"
             connectionString="Data Source=(LocalDB)\v11.0;AttachDbFilename=|DataDirectory|\Movies.mdf;Integrated Security=True"
             providerName="System.Data.SqlClient"/>
    <add     name="DefaultConnection"
             connectionString="Data Source=(LocalDb)\v11.0;AttachDbFilename=|DataDirectory|\aspnet-MvcMovie-20130603030321.mdf;
             providerName="System.Data.SqlClient"/>
/>
```

Accessing Model from a Controller



Accessing Model from a Controller

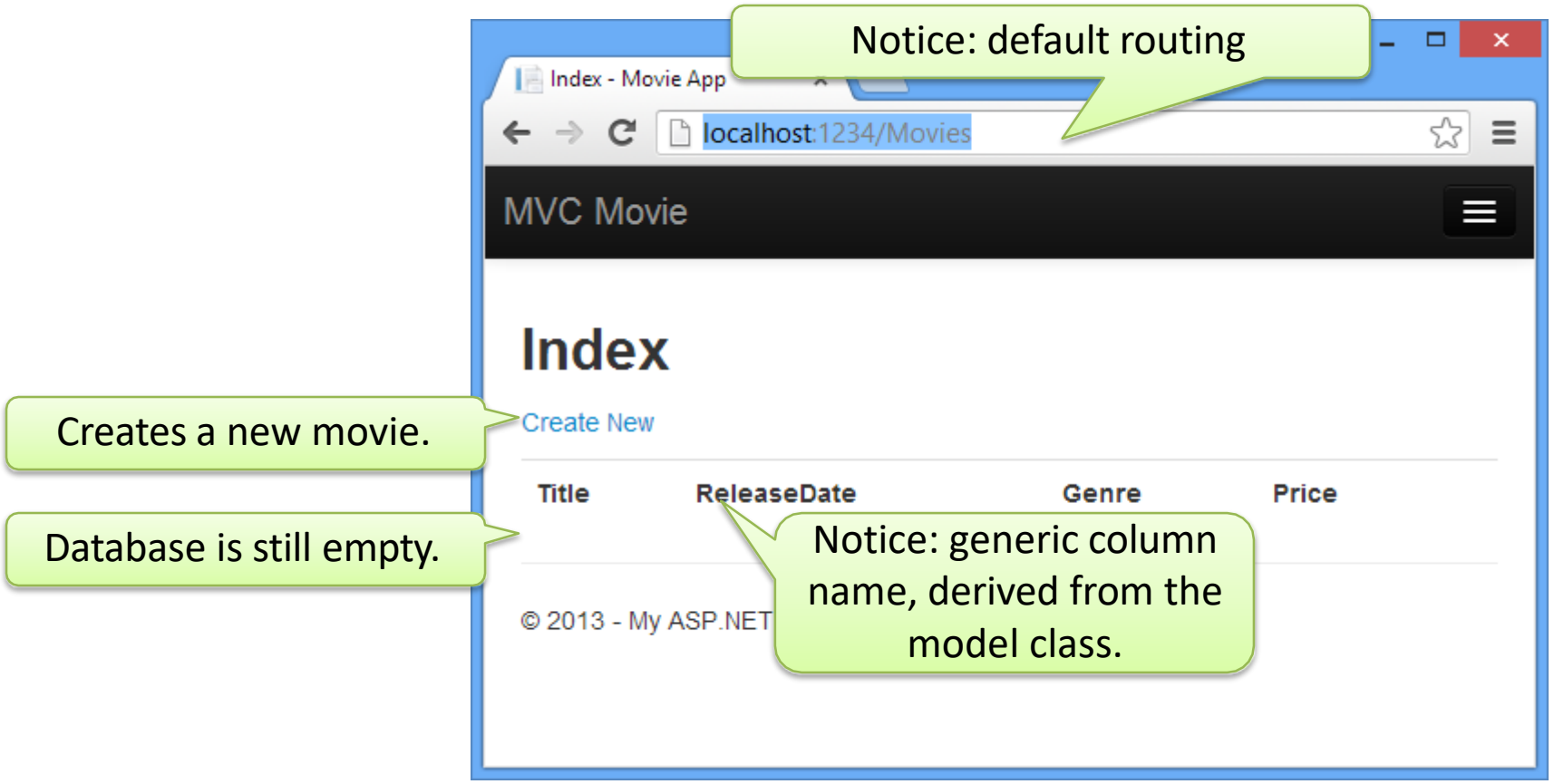




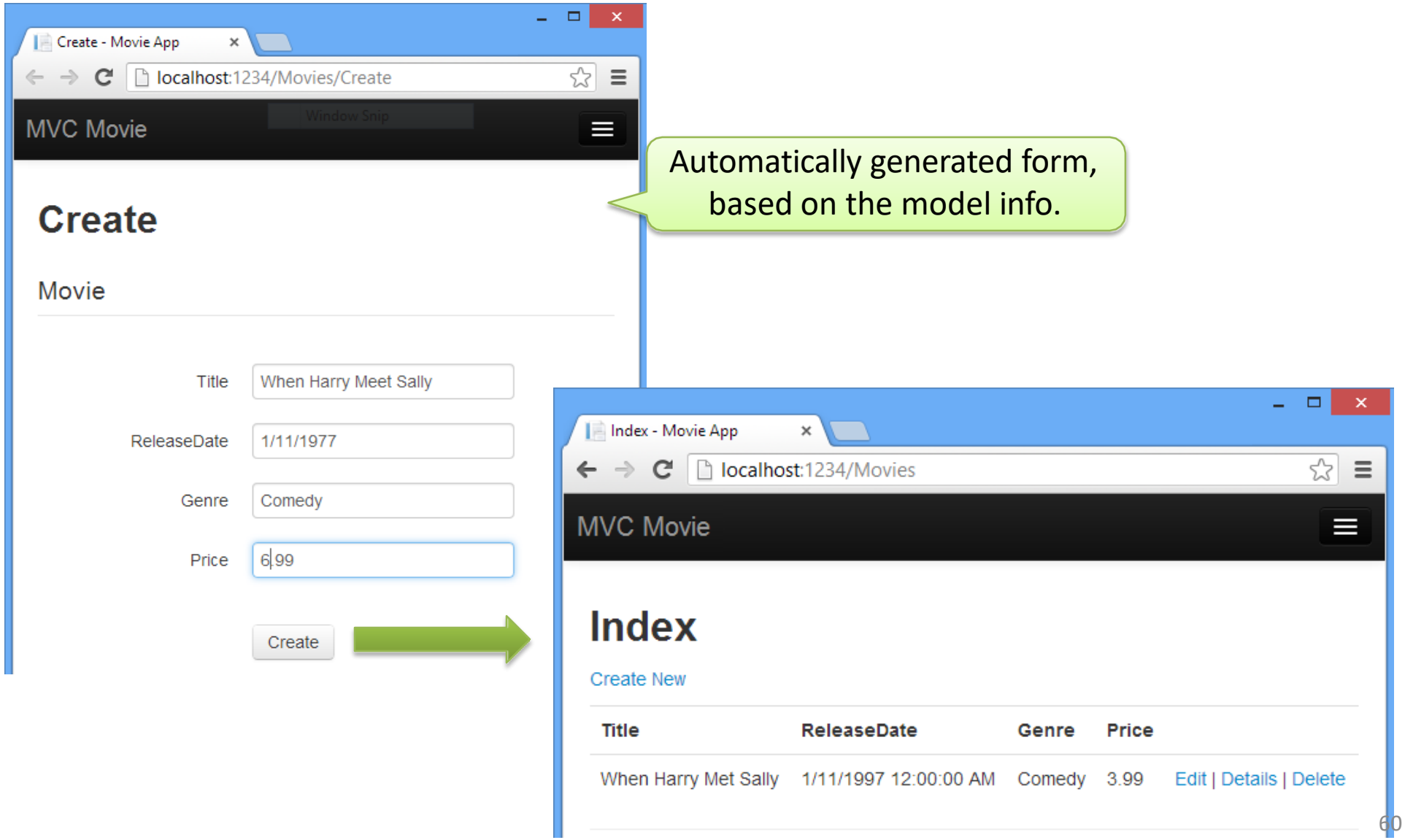
Strongly typed approach.

Visual Studio Creates:
A controller MoviesController.cs file in Controllers folder, Create.cshtml, Delete.cshtml, Details.cshtml, Index.cshtml in Views\Movies folder.

Run Application…



Creating a model object



Automatically generated form, based on the model info.

Generated Controller class

```
public class MoviesController : Controller
{
    private MovieDBContext db = new MovieDBContext();

    // GET: /Movies/
    public ActionResult Index()
    {
        return View(db.Movies.ToList());
    }
```

Instantiated DbContext instance.

Index method.

Strongly typed models

@model: Specifies class of the model

Id parameter generally passed as a part of the route.

```
public ActionResult Details(int? id)
{
    if (id == null)
    {
        return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
    }
    Movie movie = db.Movies.Find(id);
    if (movie == null)
    {
        return HttpNotFound();
    }
    return View(movie);
}
```

```
@model MvcMovie.Models.Movie

@{
    ViewBag.Title = "Details";
}

<h2>Details</h2>

<div>
    <h4>Movie</h4>
        <hr />
    <dl class="dl-horizontal">
        <dt>
            @Html.DisplayNameFor(model => model.Title)
        </dt>
         @*Markup omitted for clarity.*@
    </dl>
</div>
<p>
    @Html.ActionLink("Edit", "Edit", new { id = Model.ID }) |
    @Html.ActionLink("Back to List", "Index")
</p>
```

Communicates with the master page.

Context-sensitive data access.

Strongly typed models (cont.)

```
@model IEnumerable<MvcMovie.Models.Movie>
```

Index.cshtml

```
@foreach (var item in Model) {
    <tr>
        <td>
            @Html.DisplayFor(modelItem => item.Title)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.ReleaseDate)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Genre)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.Price)
        </td>
         <th>
            @Html.DisplayFor(modelItem => item.Rating)
        </th>
        <td>
            @Html.ActionLink("Edit", "Edit", new { id=item.ID }) |
            @Html.ActionLink("Details", "Details", { id=item.ID })  |
            @Html.ActionLink("Delete", "Delete", { id=item.ID })
        </td>
    </tr>
}
```

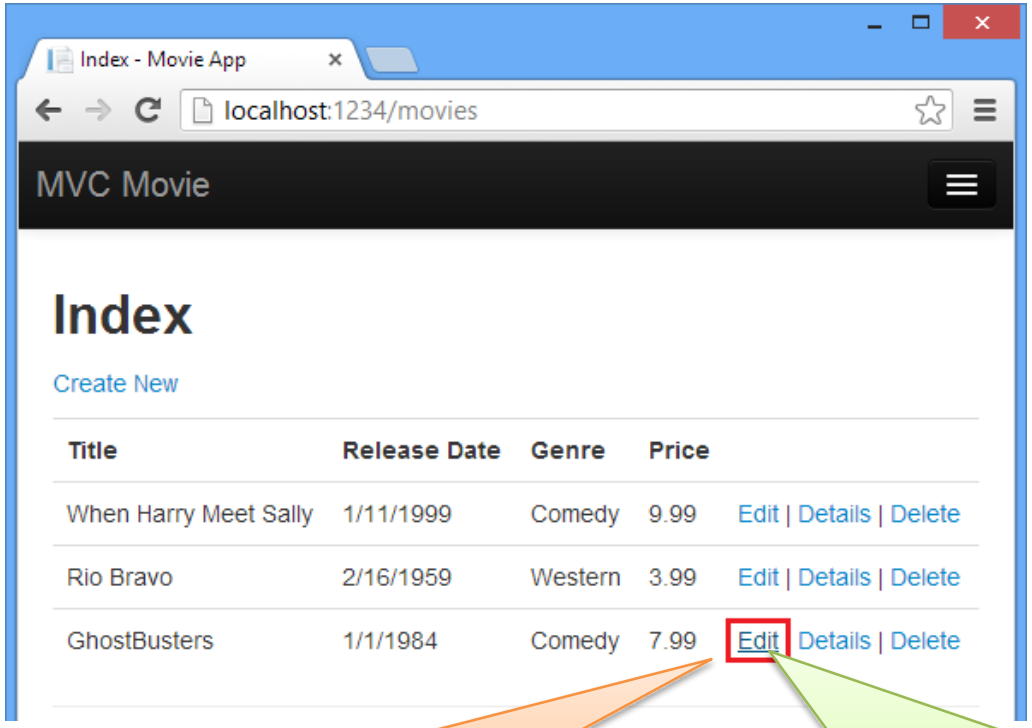
Model object is strongly typed.
Each item is a Movie object.

Full compile-time support.

```
@foreach (var item in Model) {
    <tr>
        <td>
            @Html.DisplayFor(modelItem => item.Title)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.ReleaseDate)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.)
        </td>
        <td>
            @Html.DisplayFor(modelItem => item.
        </td>
        <td>
            @Html.ActionLink("Edit", "Edit", ne
            @Html.ActionLink("Details", "Detail
            @Html.ActionLink("Delete", "Delete"
        </td>
    </tr>
}

</table>
```

Equals
Genre
GetHashCode
GetType
ID
Price
ReleaseDate
Title
ToString

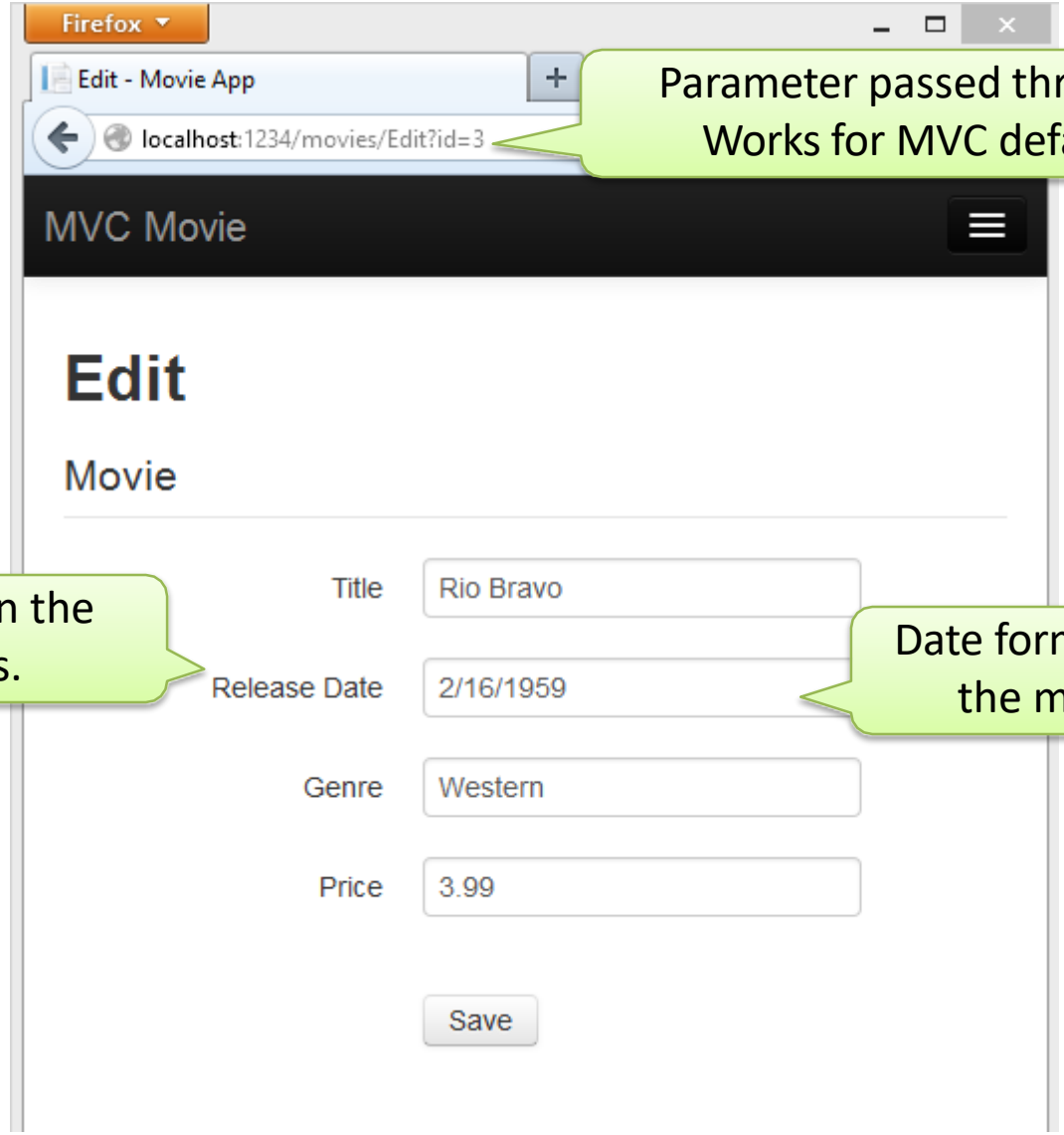
Edit View



Localhost:1234/movies/Edit/4

URL generated using Html Helpers!

Edit View (cont.)



Parameter passed through the URL query. Works for MVC default URL mapping.

Label defined in the model class.

Date format defined in the model class.

Edit View

```
@model MvcMovie.Models.Movie

@{
    ViewBag.Title = "Edit";
}
<h2>Edit</h2>
@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()
    <div class="form-horizontal">
        <h4>Movie</h4>
        <hr />
        @Html.ValidationSummary(true)
        @Html.HiddenFor(model => model.ID)

        <div class="form-group">
            @Html.LabelFor(model => model.Title, new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.Title)
                @Html.ValidationMessageFor(model => model.Title)
            </div>
        </div>
```

Generates hidden anti-forgery token.

Generates html label.

Generates text box.

Generates validation message.

```
<input
    name="__RequestVerificationToken"
    type="hidden"
    value="UxY6bkQyJCXO3Kn5AXg-6TXxOj6yVBi9tghHaQ5Lq_qwKvcojNXEEfcbn-FGh_0vuw4tS_BRk7QQQHlJp8AP4_X4orVNoQnp2cd8kXhykS01"
/>
```

Property annotations

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

namespace MvcMovie.Models
{
    public class Movie
    {
        public int ID { get; set; }
        public string Title { get; set; }

        [Display(Name = "Release Date")]
        [DataType(DataType.Date)]
        [DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}", ApplyFormatInEditMode = true)]
        public DateTime ReleaseDate { get; set; }
        public string Genre { get; set; }
        public decimal Price { get; set; }
    }

    public class MovieDBContext : DbContext
    {
        public DbSet<Movie> Movies { get; set; }
    }
}
```

Annotations namespace.

Overrides default label name on the view page.

Specifies type of the data: displays only date part.

Workaround for a bug in Chrome ☺

Edit actions

- Implemented as Controller's operations

HTTP GET operation

```
// GET: /Movies/Edit/5
public ActionResult Edit(int? id)
{
    if (id == null)
    {
        return new HttpStatusCodeResult(Http
    }
    Movie movie = db.Movies.Find(id);
    if (movie == null)
    {
        return HttpNotFound();
    }
    return View(movie);
}
```

[HttpGet] annotation by default.

HTTP POST operation

Prevents request forgery

```
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Edit([Bind(Include="ID,Title,ReleaseDate,Genre,Price")] Movie movie)
{
    if (ModelState.IsValid)
    {
        db.Entry(movie).State = EntityState.Modified;
        db.SaveChanges();
        return RedirectToAction("Index");
    }
    return View(movie);
}
```

[Bind] attribute – a security mechanism that prevents over-posting data to the model.

Processing the POST request

HTTP POST method.

Validates the forgery token.

```
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Edit([Bind(Include="ID,Title,ReleaseDate,Genre,Price")] Movie movie)
{
    if (ModelState.IsValid)
    {
        db.Entry(movie).State = EntityState.Modified;
        db.SaveChanges();
        return RedirectToAction("Index");
    }
    return View(movie);
}
```

Checks if sent data are valid – server side validation, compared to client-side validation (javascript)

Redirects after successful update.

In case of invalid data, the original form is returned back to the client, displaying error messages

Movie

Title: Rio Bravo

Release Date: abc

The field Release Date must be a date.

Genre: Western

Price: xyz

The field Price must be a number.

Save

HTTP methods – best practices

- HttpGet and HttpPost method overloads
- All methods that modify data SHOULD use HttpPost method overload
- Modifying data in HttpGet method
  - security risk
  - Violates HTTP best practices
  - Violates REST architectural pattern
- GET method SHOULD NOT have any side effect and SHOULD NOT modify persistent data

Details method - Controller

```
public ActionResult Details(int? id)
{
    if (id == null)
    {
        return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
    }
    Movie movie = db.Movies.Find(id);
    if (movie == null)
    {
        return HttpNotFound();
    }
    return View(movie);
}
```

Delete method - Controller

HttpGet method.
Selects an objects and returns Details page.

```
// GET: /Movies/Delete/5
public ActionResult Delete(int? id)
{
    if (id == null)
    {
        return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
    }
    Movie movie = db.Movies.Find(id);
    if (movie == null)
    {
        return HttpNotFound();
    }
    return View(movie);
}
```

Asp .net maps a segment of URL to a method.
Attribute ActionName is necessary to provide valid URL routing.
The same URL maps to different action methods, based on used HTTP method.

```
// POST: /Movies/Delete/5
[HttpPost, ActionName("Delete")]
[ValidateAntiForgeryToken]
public ActionResult DeleteConfirmed(int id)
{
    Movie movie = db.Movies.Find(id);
    db.Movies.Remove(movie);
    db.SaveChanges();
    return RedirectToAction("Index");
}
```

HttpPost method.
Deletes an object having the given id.

RULE:
Never use a HttpGet method to modify the model.
Opens security holes, architecturally bad!

Data Validation

- Keep Things DRY
  (Don’t Repeat Yourself)
- Declarative validation rules in one place
  (Model class)
  - Regular expressions
  - Range validation
  - Length validation
  - NULL values validation
  - Data formatting
- Validation rules enforced before saving changes
  to the database!

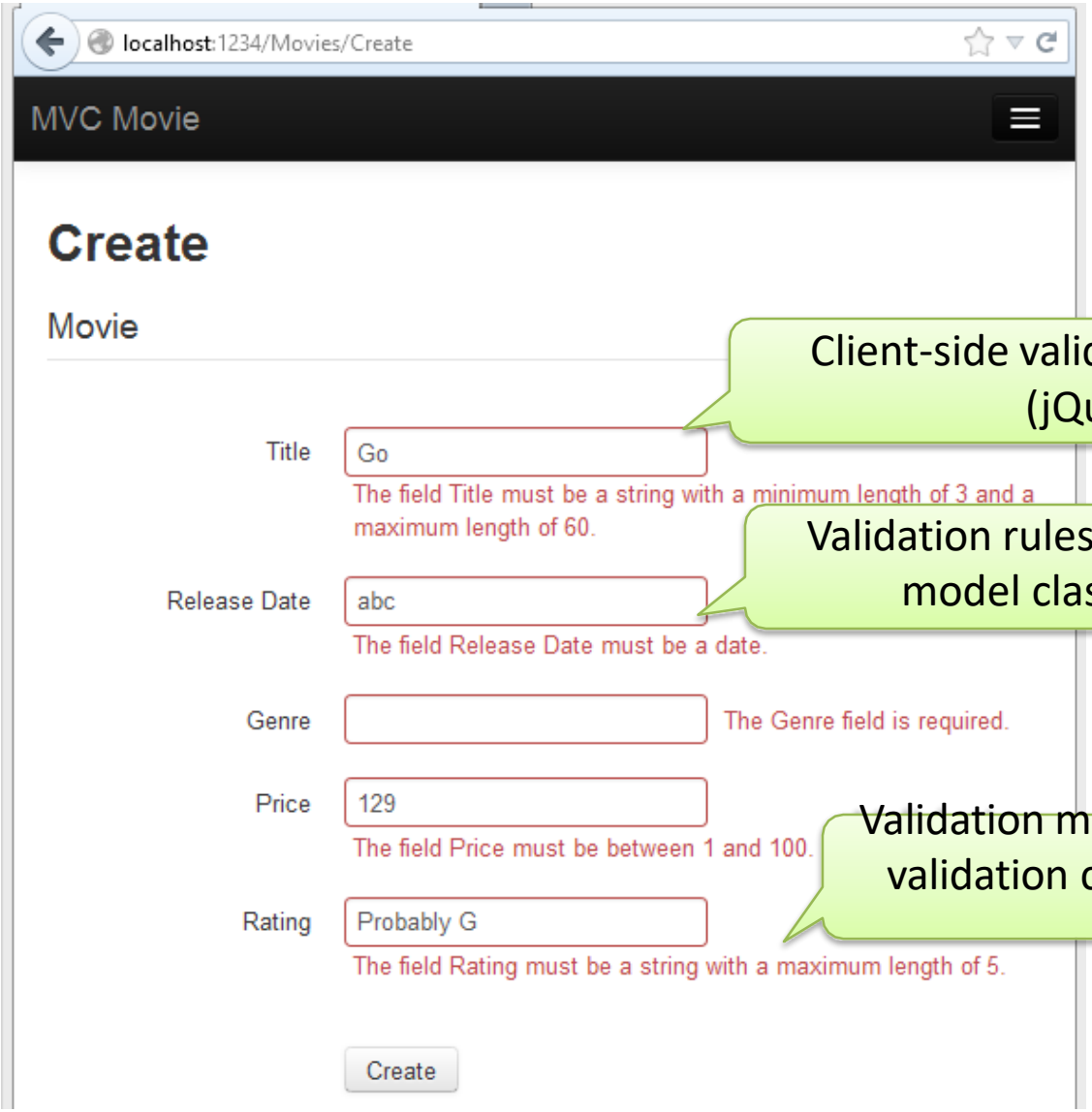
Validation rules – Model

```
public class Movie
{
    public int ID { get; set; }
    [StringLength(60, MinimumLength = 3)]
    public string Title { get; set; }

    [Display(Name = "Release Date")]
    [DataType(DataType.Date)]
    [DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}", ApplyFormatInEditMode = true)]
    public DateTime ReleaseDate { get; set; }

    [RegularExpression(@"^[A-Z]+[a-zA-Z''-'\s]*$")]
    [Required]
    [StringLength(30)]
    public string Genre { get; set; }

    [Range(1, 100)]
    [DataType(DataType.Currency)]
    public decimal Price { get; set; }

    [RegularExpression(@"^[A-Z]+[a-zA-Z''-'\s]*$")]
    [StringLength(5)]
    public string Rating { get; set; }
}
```

```
MovieDBContext db = new MovieDBContext();
Movie movie = new Movie();
movie.Title = "Gone with the Wind";
db.Movies.Add(movie);
db.SaveChanges();        // <= Will throw server side validation exception
```

Several validation rules failed.

Data Validation - View



Client-side validation: javascript (jQuery).

Validation rules picked up from the model class annotations.

Validation messages derived from the validation constraints in the model class.

Data Validation – View (cont.)

```
@model MvcMovie.Models.Movie
@{
    ViewBag.Title = "Create";
}
<h2>Create</h2>
@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()
    <div class="form-horizontal">
        <h4>Movie</h4>
        <hr />
        @Html.ValidationSummary(true)
        <div class="form-group">
            @Html.LabelFor(model => model.Title, new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.Title)
                @Html.ValidationMessageFor(model => model.Title)
            </div>
        </div>
        <div class="form-group">
            <div class="col-md-offset-2 col-md-10">
                <input type="submit" value="Create" class="btn btn-default" />
            </div>
        </div>
    </div>
}
<div>
    @Html.ActionLink("Back to List", "Index")
</div>
@section Scripts {
    @Scripts.Render("~/bundles/jqueryval")
}
```

Validation message derived from the validation constraints specified for the given Property (Title)

Data Validation - Controller

```
public ActionResult Create()
{
    return View();
}
// POST: /Movies/Create
// To protect from overposting attacks, please enable the specific properties you want to bind
// more details see http://go.microsoft.com/fwlink/?LinkId=317598.
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Create([Bind(Include = "ID,Title,ReleaseDate,Genre,Price,Rating")] Movie mo
{
    if (ModelState.IsValid)
    {
        db.Movies.Add(movie);
        db.SaveChanges();
        return RedirectToAction("Index");
    }
    return View(movie);
}
```

HttpGet method displays initial Create form.

HttpPost method that does create a new object.

Server-side data validation check.

DataType attributes

- Provide only hits for the view engine
  to format the data
- Date, Time, PhoneNumber, EmailAddress,…
- Automatic provision of type specific features
  e.g. “mailto: …” link for EmailAddress
- Do NOT provide any Validation
  (just presentation hints)

DisplayFormat annotation

- Used to explicitly specify format of the data
- Example: redefining the default date format

```
[DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}", ApplyFormatInEditMode = true)]
public DateTime EnrollmentDate { get; set; }
```

```
public class Movie
{
   public int ID { get; set; }
   [Required,StringLength(60, MinimumLength = 3)]
   public string Title { get; set; }
   [Display(Name = "Release Date"),DataType(DataType.Date)]
   public DateTime ReleaseDate { get; set; }
   [Required]
   public string Genre { get; set; }
   [Range(1, 100),DataType(DataType.Currency)]
   public decimal Price { get; set; }
   [Required,StringLength(5)]
   public string Rating { get; set; }
}
```

It is possible to specify validation properties in one line!

Thank You!!!