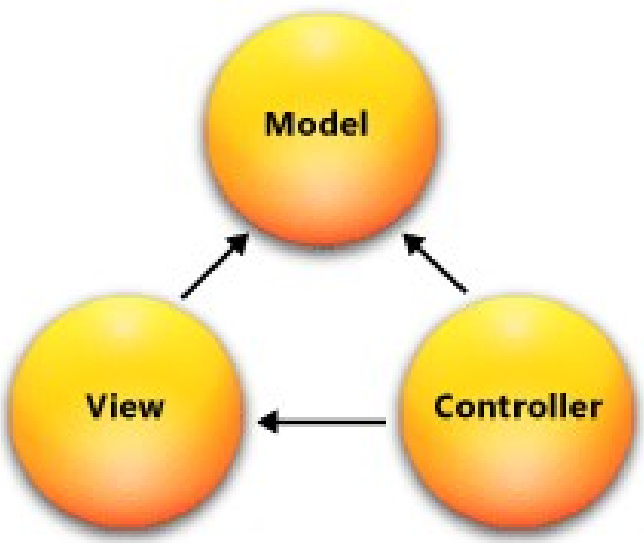
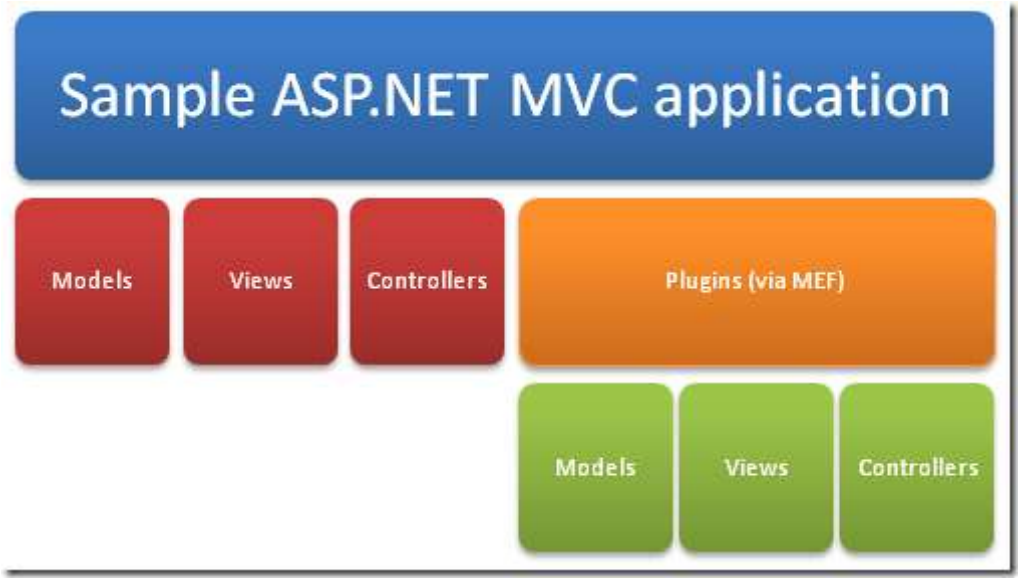
What is MVC?

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



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
    - …



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

When to use MVC approach?

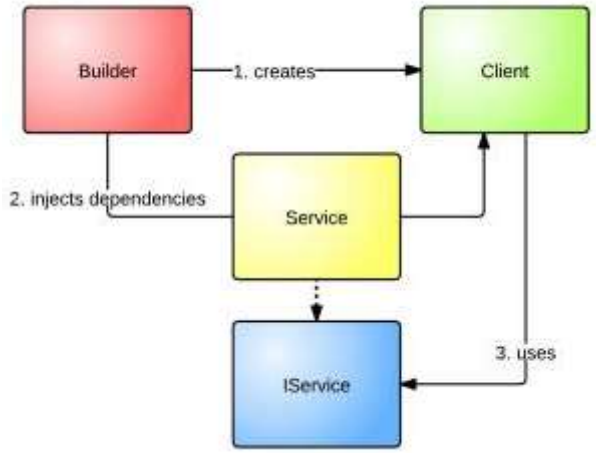
- Advantages:
  - Easier to manage complexity (divide and conquer)
  - It does not use server forms and view state
  - Front Controller pattern (rich routing)
  - Better support for test-driven development
  - Ideal for distributed and large teams
  - High degree of control over the application behavior

ASP .NET MVC Features

- Separation of application tasks
  - Input logic, business logic, UI logic
- Support for test-driven development
  - Unit testing
  - No need to start app server
- Extensible and pluggable framework
  - Components easily replaceable or customized (view engine, URL routing, data serialization,…)

ASP .NET MVC Features (cont.)

- Support for Dependency Injection (DI)
  - Injecting objects into a class
  - Class doesn't need to create objects itself
- Support for Inversion of Control (IOC)
  - If an object requires another object, the first should get the second from an outside source (configuration file)



ASP .NET MVC Features (cont.)

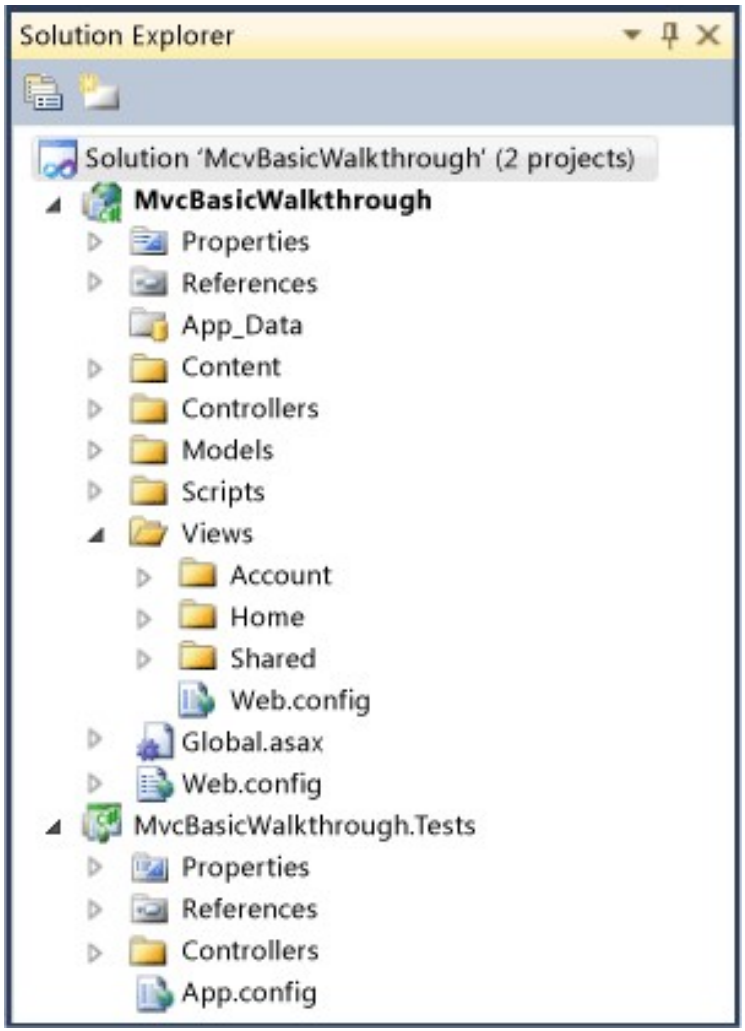
- Extensive support for ASP .NET routing
- Building apps with comprehensible and searchable URLs
- Customizable URLs
  - Adapted to work well with search engines
  - Adapted to REST addressing
  - Decoupled from resource files
- Use of existing ASP .NET features (backward compatibility)

ASP .NET MVC App Structure

- URLs mapped to controller classes
- Controller
  - handles requests,
  - executes appropriate logic and
  - calls a View to generate HTML response
- URL routing
  - ASP .NET routing engine (flexible mapping)
  - Support for defining customized routing rules
  - Automatic passing/parsing of parameters

ASP .NET App Structure

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



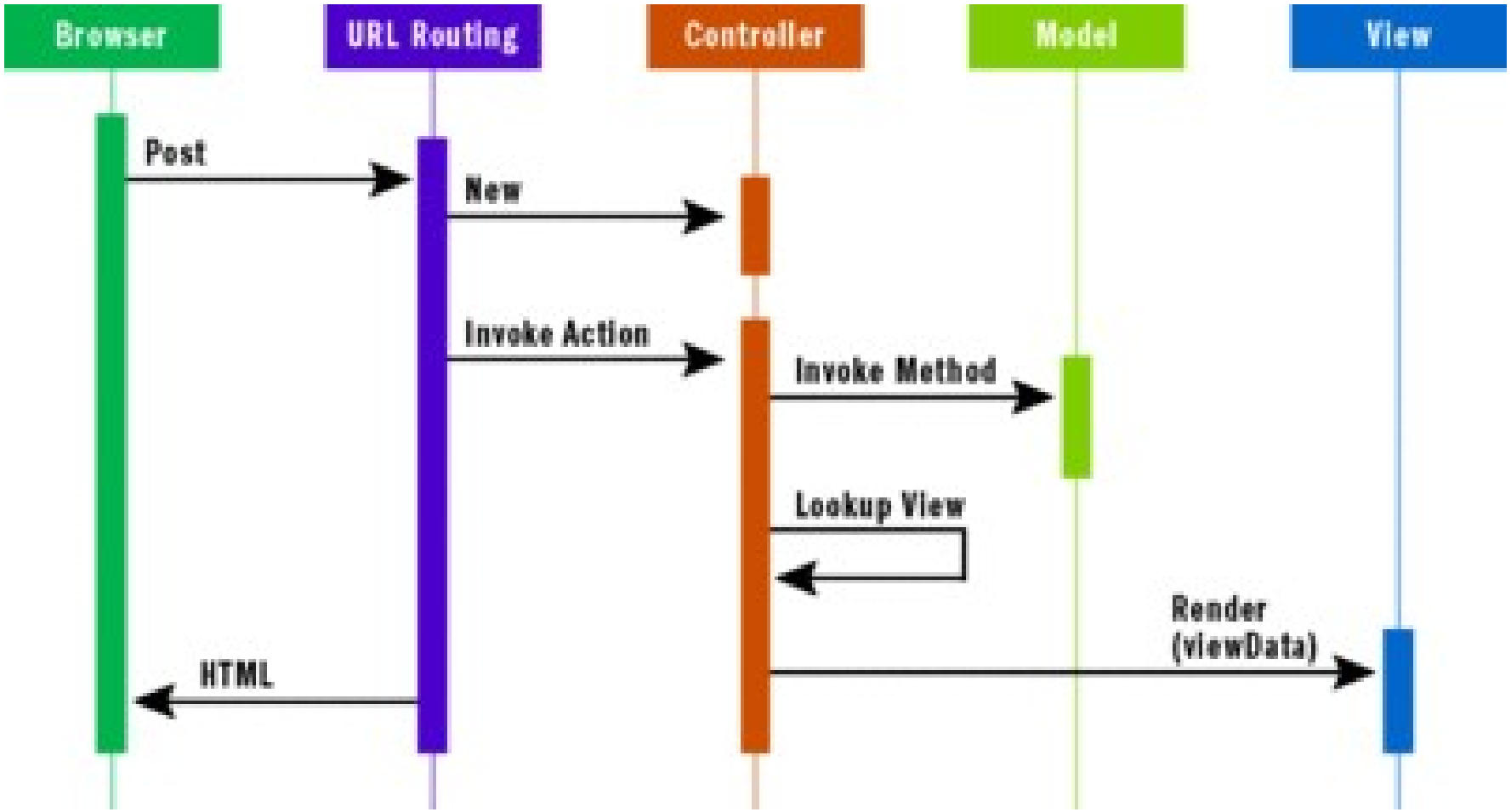
MVC App Execution

- Entry points to MVC:
  - UrlRoutingModule and MvcRouteHandler
- Request handling:
  - Select appropriate controller
  - Obtain a specific controller instance
  - Call the controller's Execute method

MVC App Execution - stages

- Receive first request for the application
  - Populating RouteTable
- Perform routing
- Create MVC Request handler
- Create controller
- Execute controller
- Invoke action
- Execute result
  - ViewResult, RedirectToRouteResult, ContentResult, FileResult, JsonResult, RedirectResult

MVC App Execution



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…

Razor – HelloWorld

- Uses @ for Razor blocks

razor file

```
<h1>Razor Example</h1>

<h3>
    Hello @name, the year is @DateTime.Now.Year
</h3>

<p>
    Checkout <a href="/Products/Details/@productId">this product</a>
</p>
```

.aspx file

```
<h1>Code Nugget Example with .ASPX file</h1>

<h3>
    Hello <%=name %>, the year is <%= DateTime.Now.Year %>
</h3>

<p>
    Checkout <a href="/Products/Details/<%=productId %>">this product</a>
</p>
```

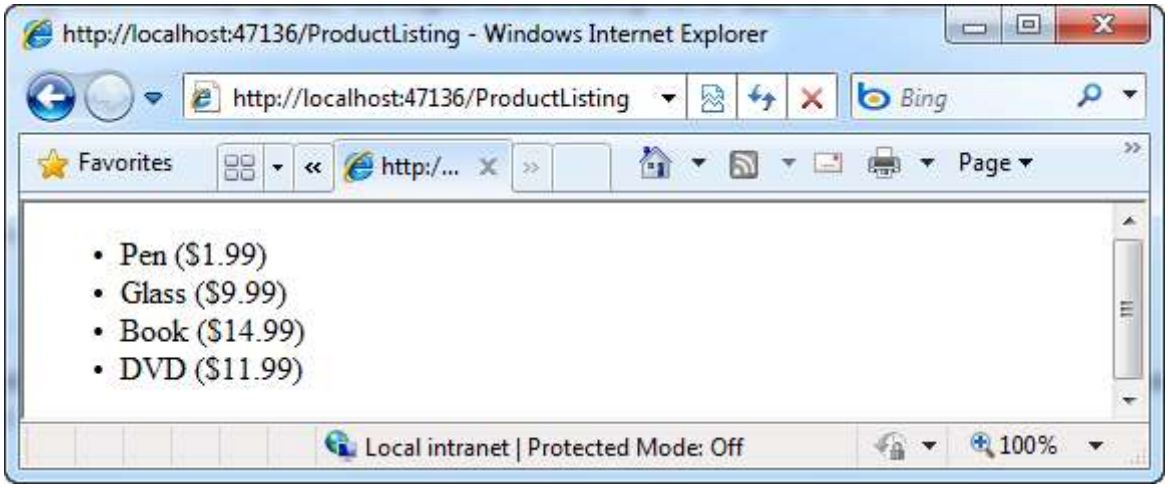
Loops and Nested HTML

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 can span multiple server code blocks!

Integrating Content and Code

Parser examines right-hand side of @ character.

```
<p>
    Send mail to scottgu@microsoft.com telling him the time: @DateTime.Now.
</p>
```

```
<p>
    Send mail to scottgu@microsoft.com telling him the time: 7/2/2010 1:53:12 PM.
</p>
```

```
@if (DateTime.Now.Year == 2010) {
    <span>
        if year is 2010 then print this <br/>
        multi-line text block and
        the date: @DateTime.Now
    </span>
}
```

Identifying nested content with HTML block tag

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.

Complete HTML 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">

            <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>
```

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.

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>
```

```
<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>

@section menu {
    <ul id="sub-menu">
        <li>About Item 1</li>
        <li>About Item 2</li>
    </ul>
}

@section footer {
    <p>This is my custom footer for Home</p>
}
```

Named section.

Named section.

Master page – result html

```
<!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">

            <ul id="sub-menu">
                <li>About Item 1</li>
                <li>About Item 2</li>
            </ul>

        </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 3:34:05 PM
            </p>

        </div>

        <div id="footer">

            <p>This is my custom footer for Home</p>

        </div>

    </body>
</html>
```

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>
}
```

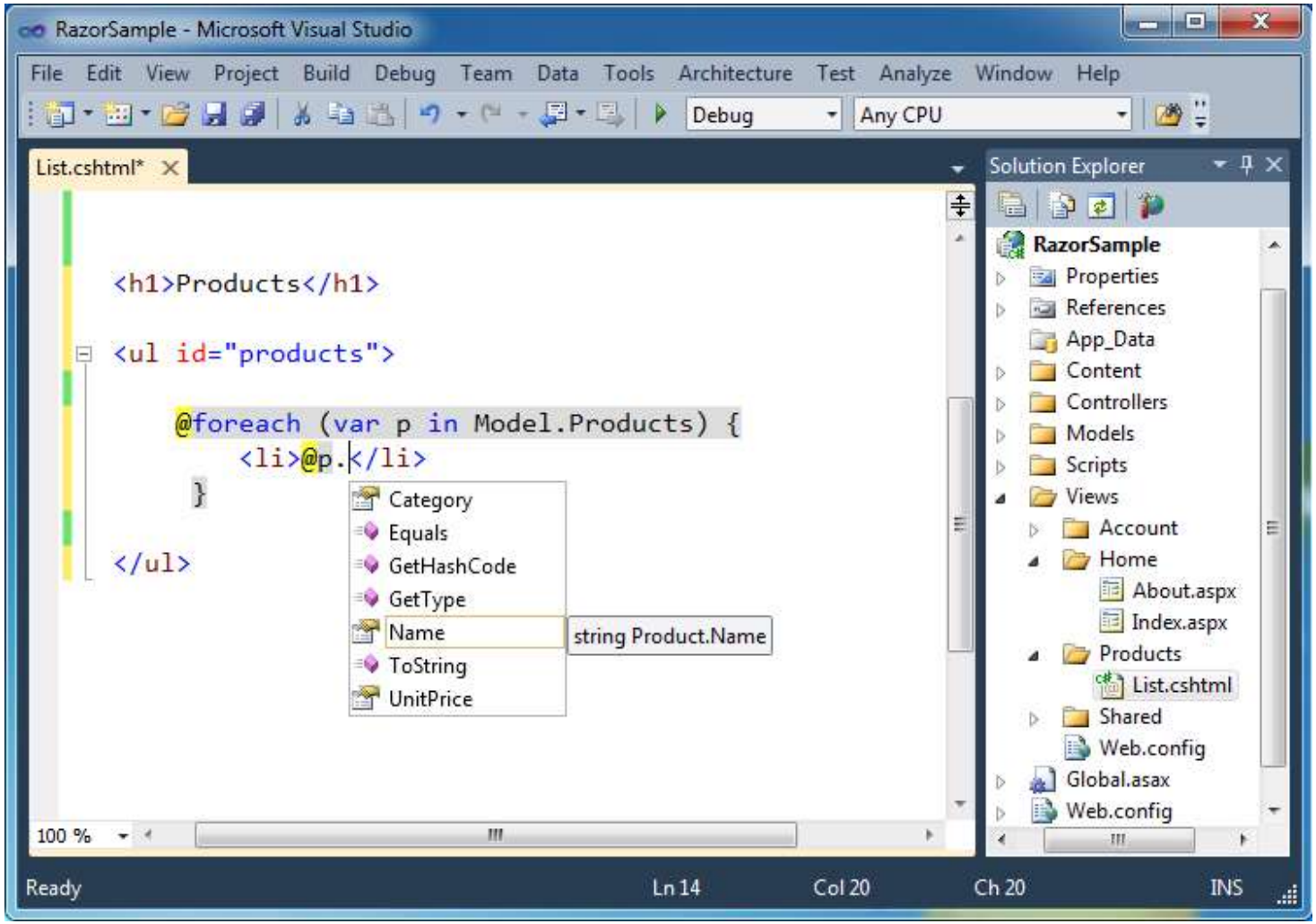
HTML Helper definition

HTML Helper should be placed to Views\Helper directory.

HTML Helper Invocation

```
<body>

    <h1>Here are My Products</h1>

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

</body>
```

Visual Studio support

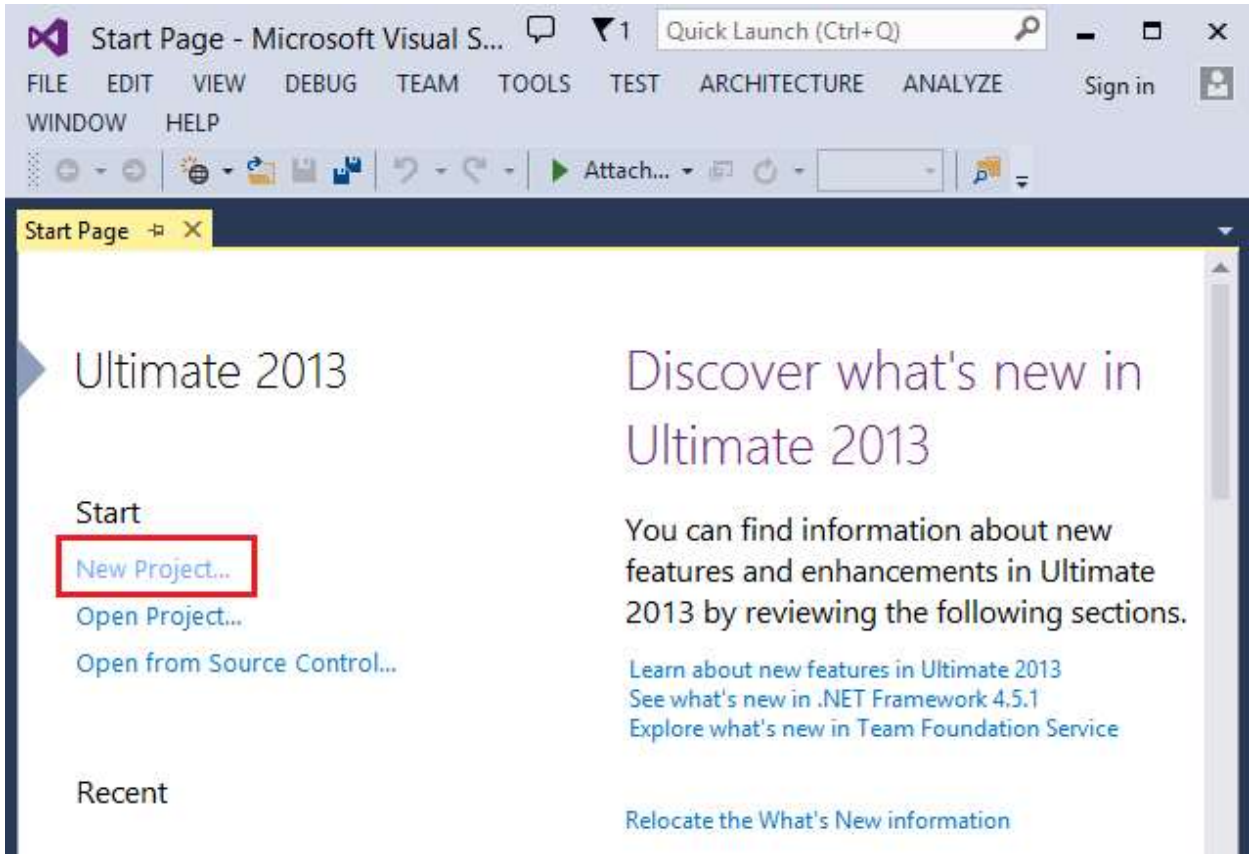


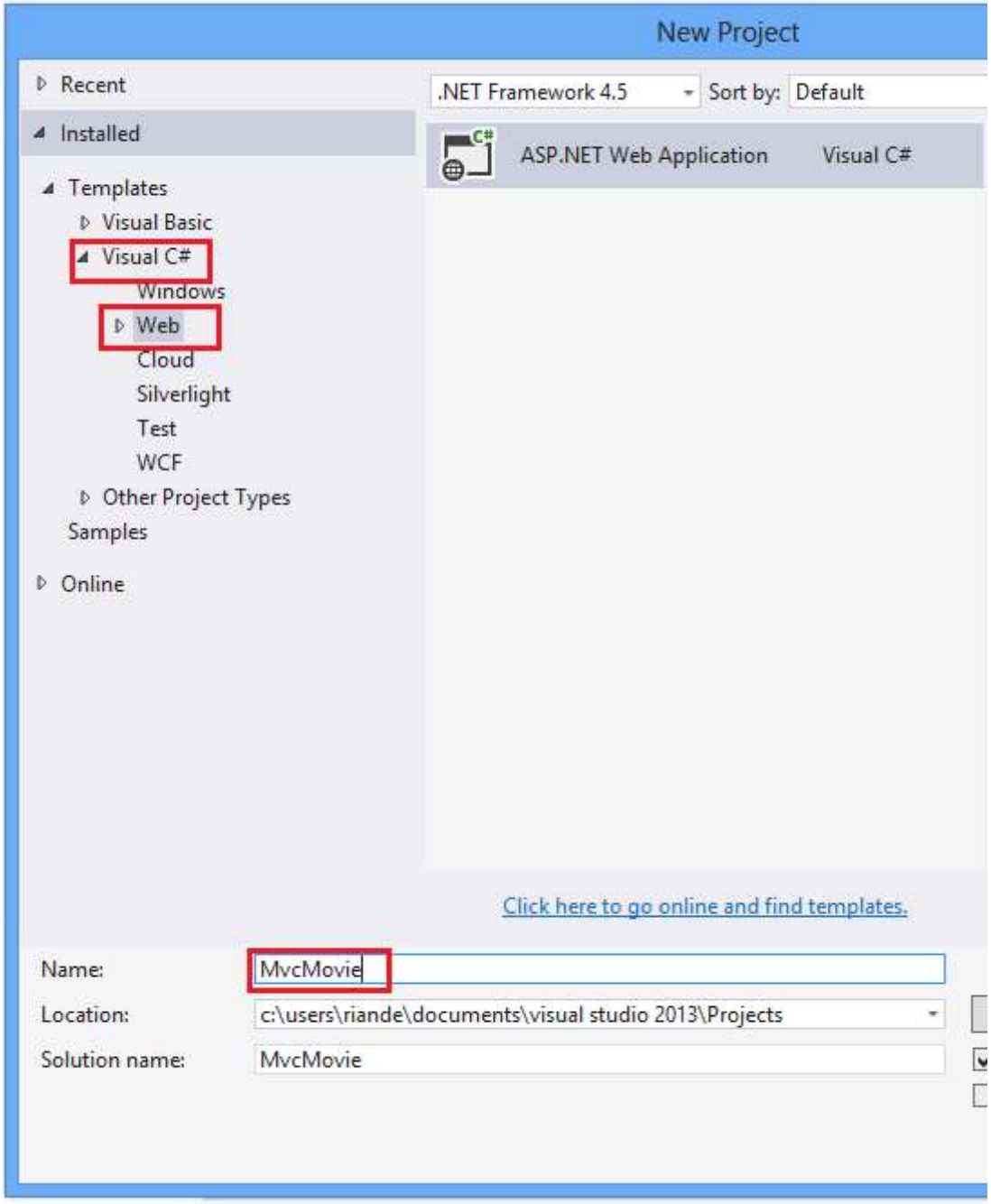
Razor – Summary

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

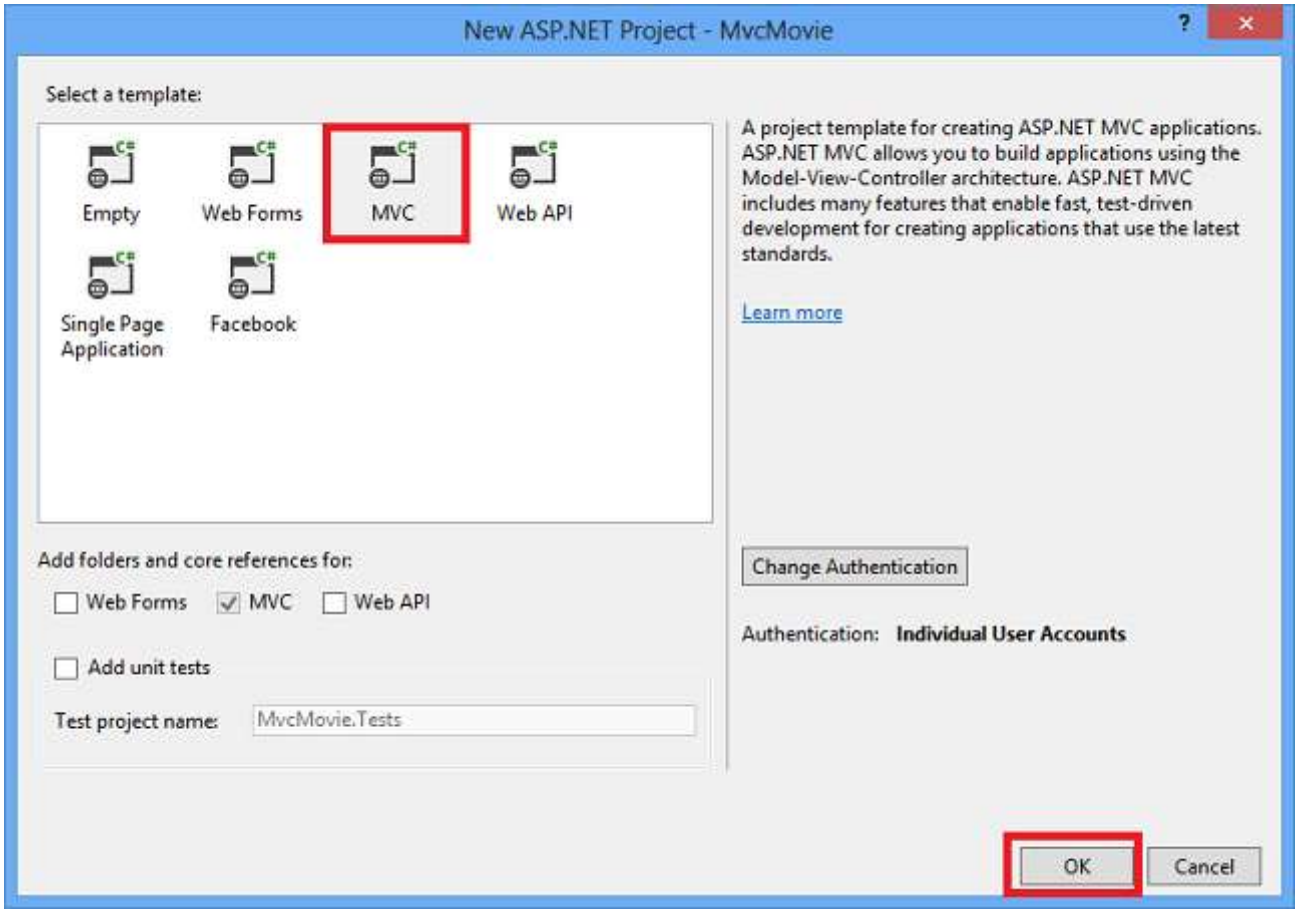
CREATING ASP .NET MVC APPLICATION

New Project …

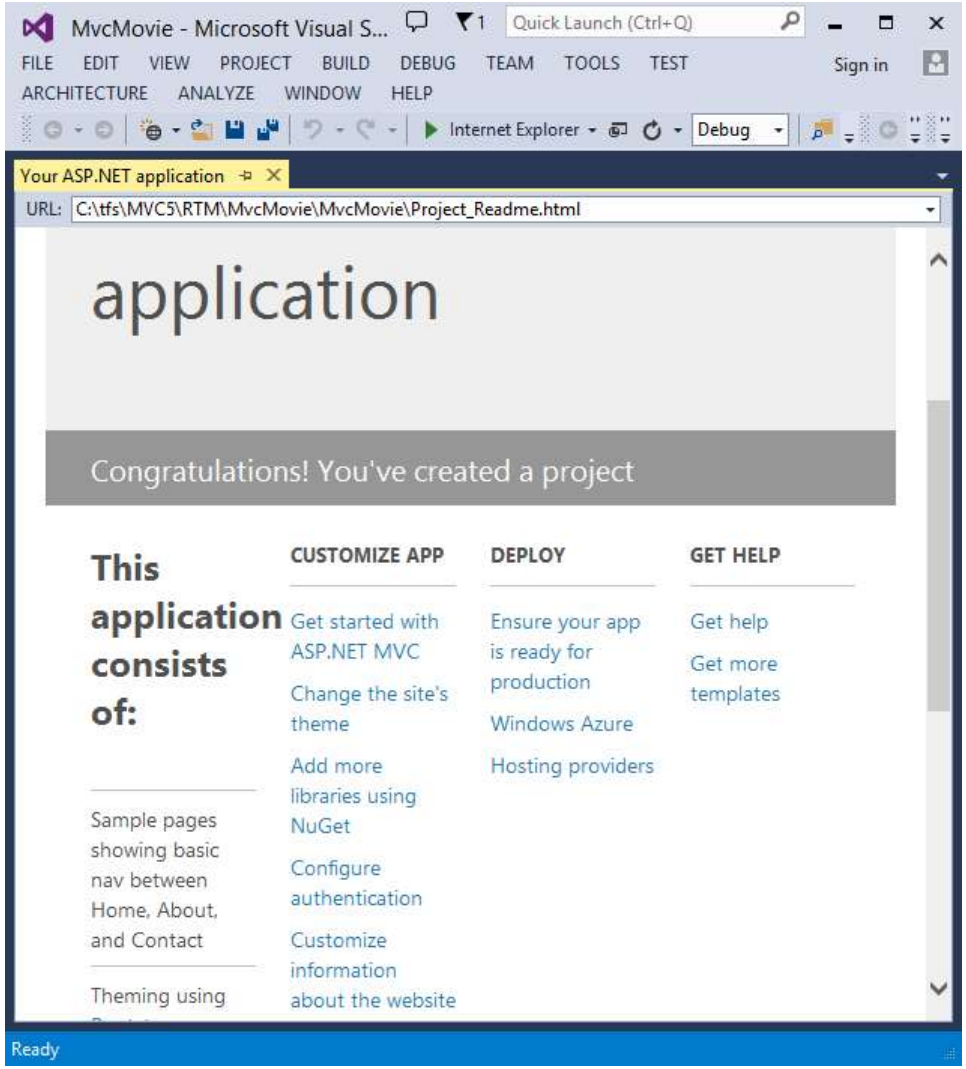

Start Page - Microsoft Visual S...
Quick Launch (Ctrl+Q)
FILE EDIT VIEW DEBUG TEAM TOOLS TEST ARCHITECTURE ANALYZE
Sign in
WINDOW HELP
Attach...
Start Page
Ultimate 2013
Start
New Project...
Open Project...
Open from Source Control...
Recent
Discover what's new in Ultimate 2013
You can find information 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

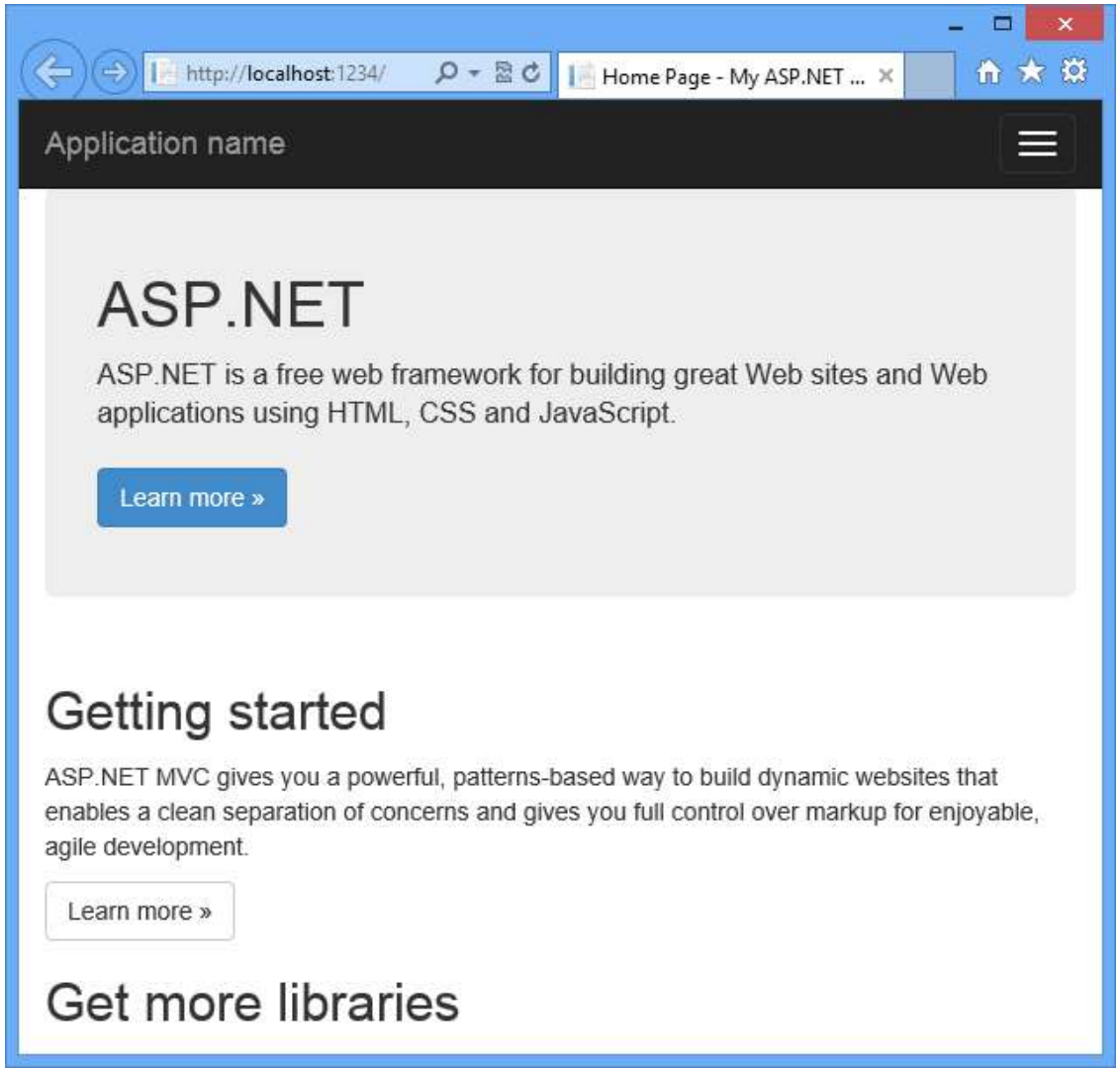
Select the project template



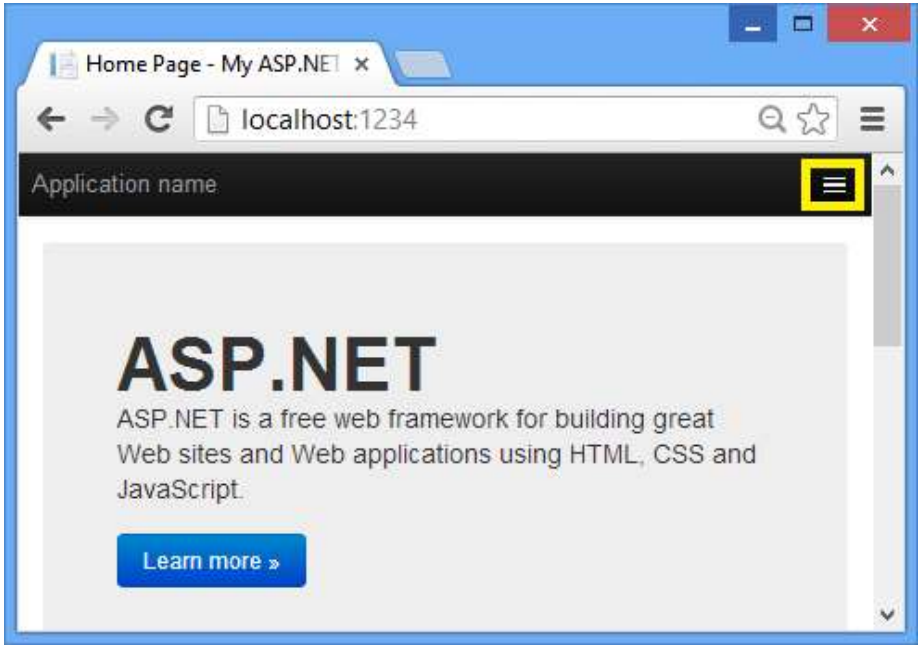
ASP .NET MVC App Home page

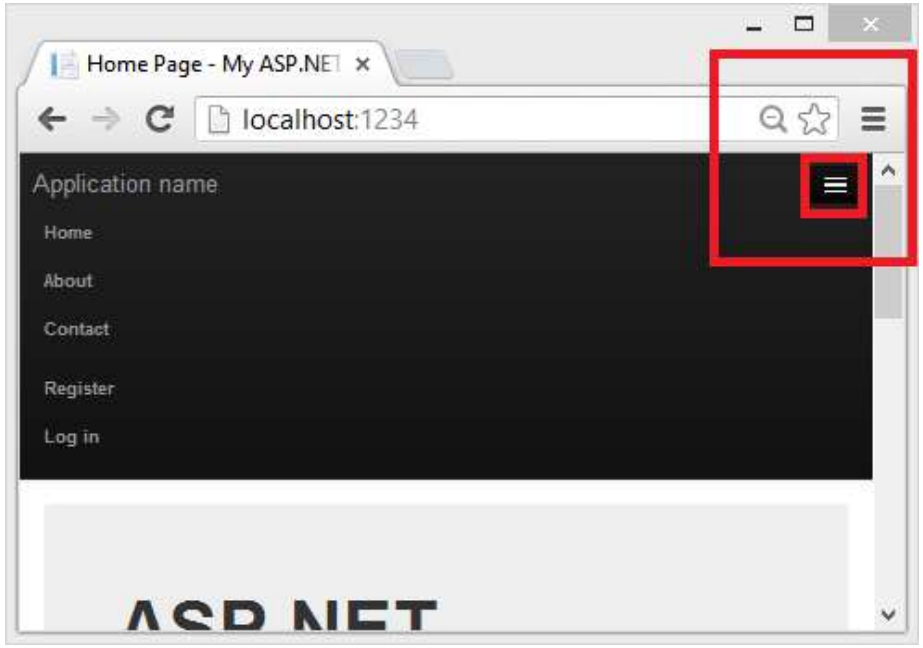


Run the application…



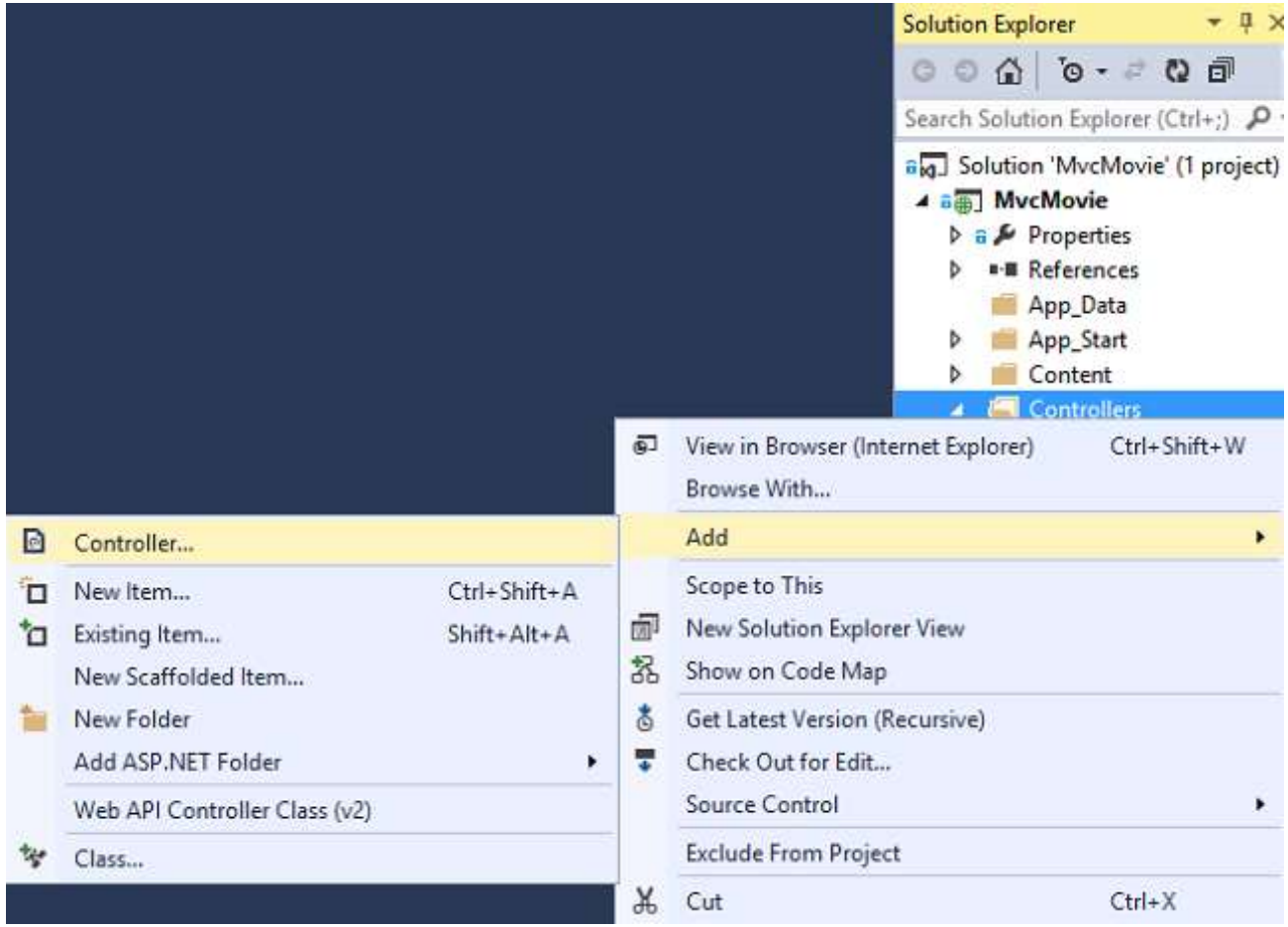
Expand the default App menu



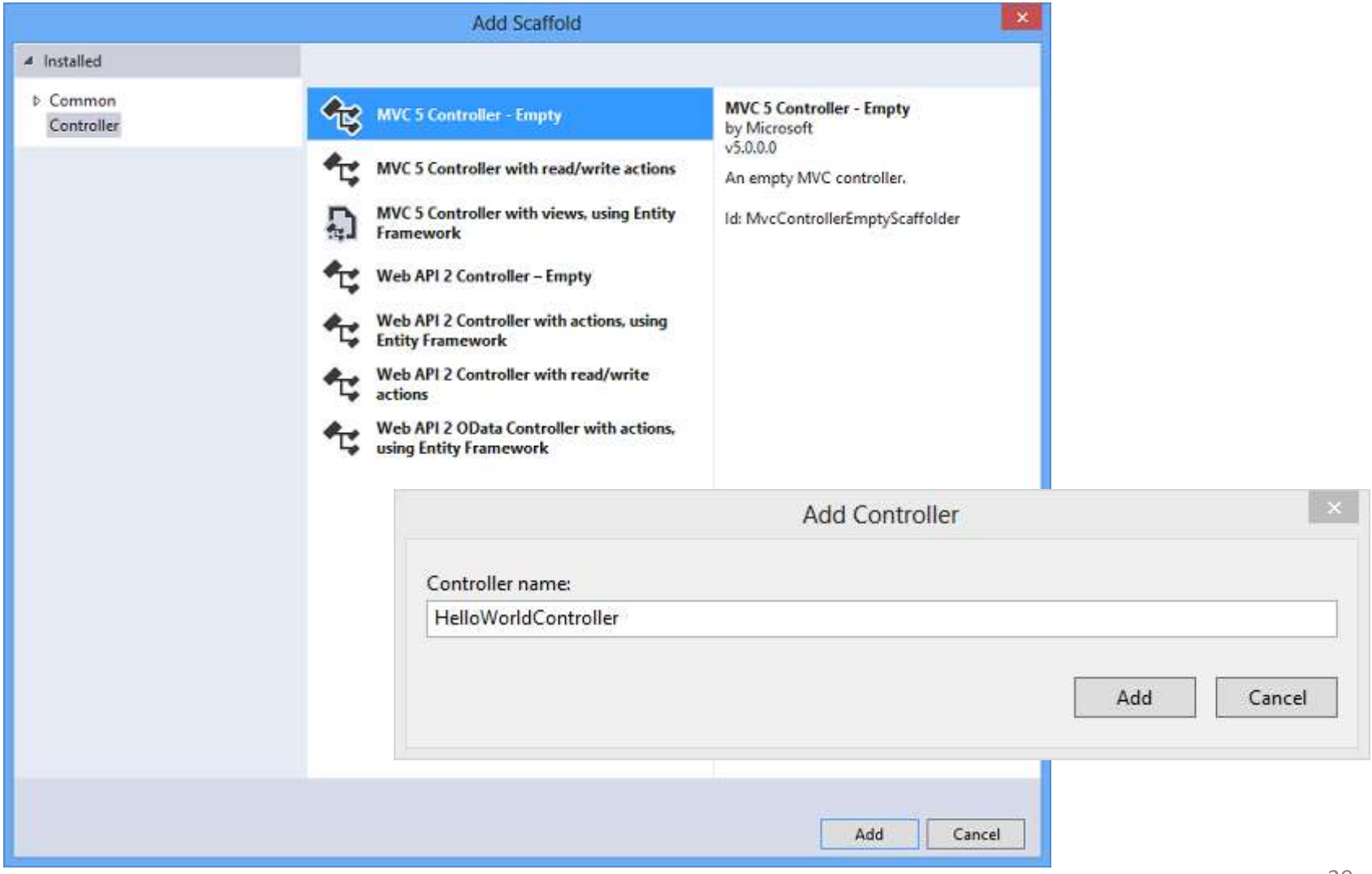


ADDING CONTROLLER

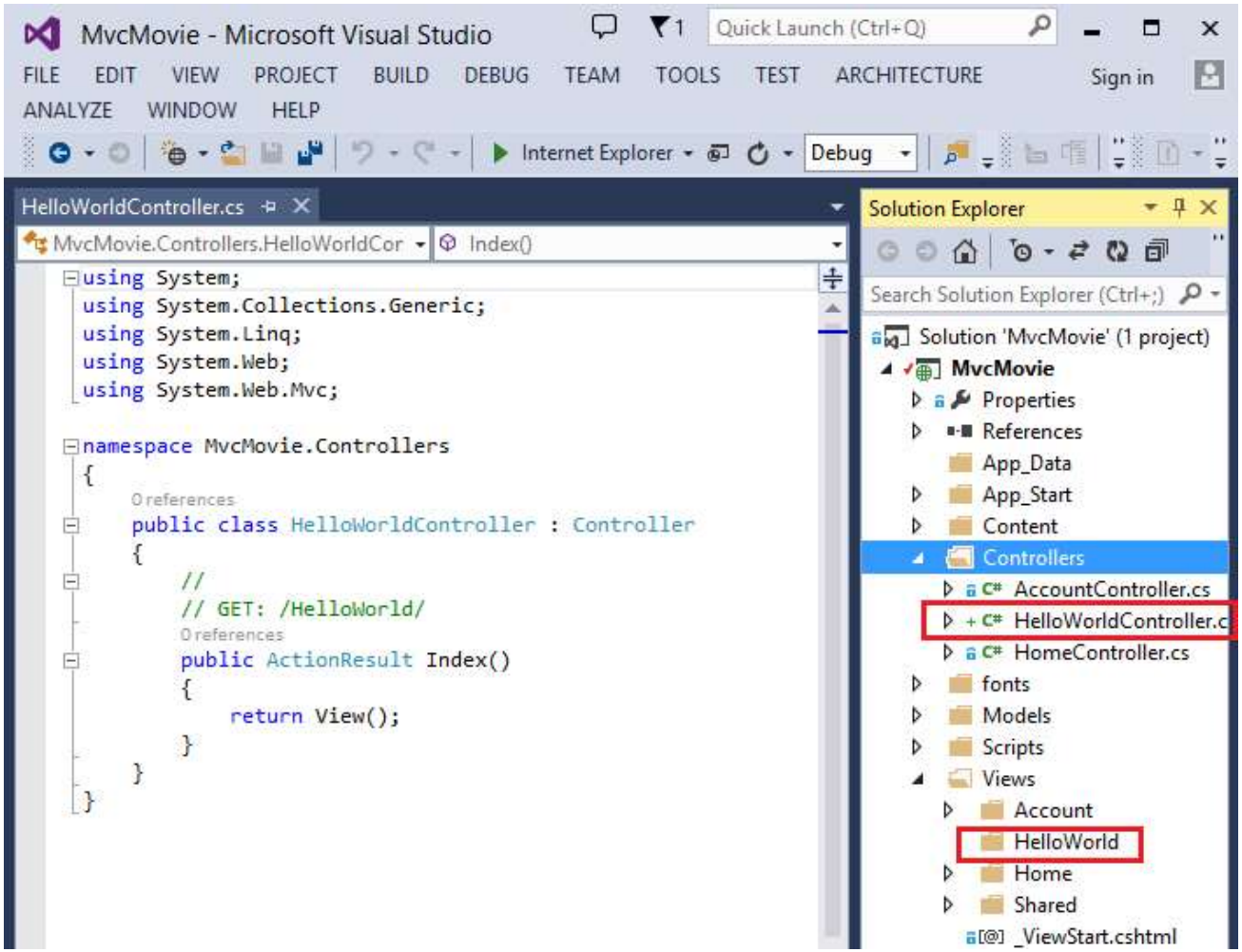
Adding controller



Adding controller (cont.)

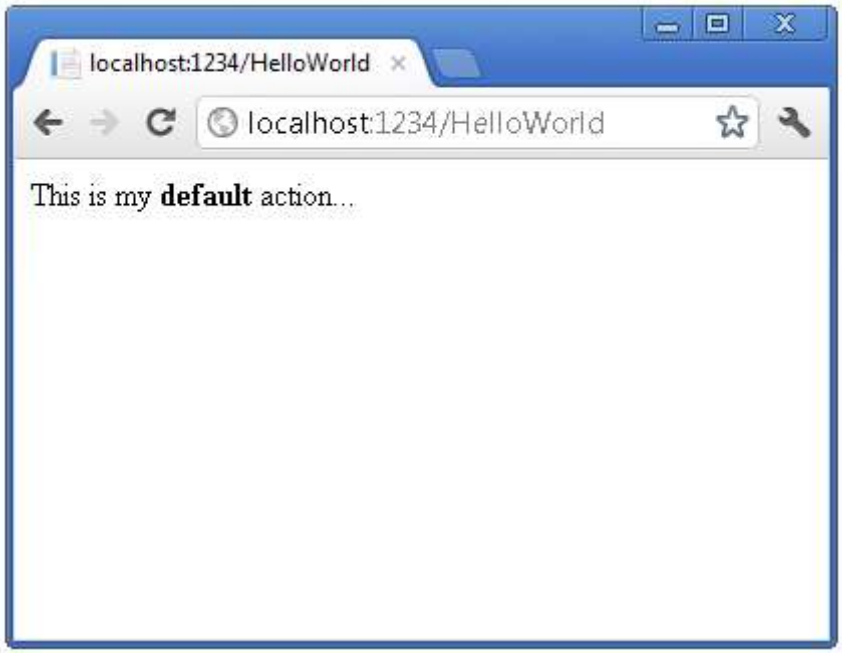


Adding a controller (cont.)



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!

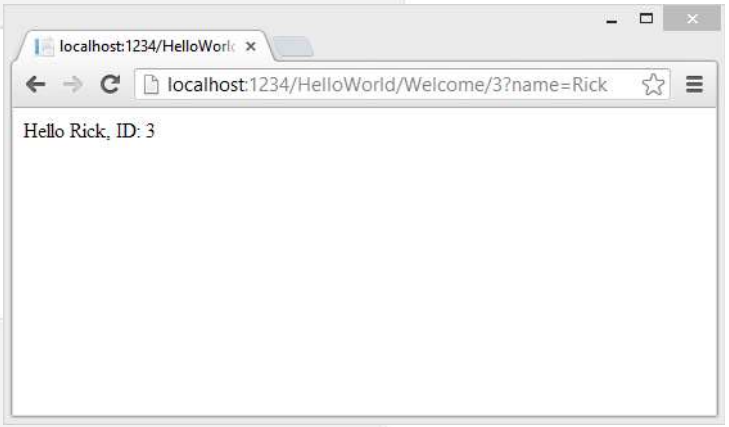
```
public string Welcome(string name, int numTimes = 1) {
    return HttpUtility.HtmlEncode("Hello " + name + ", NumTimes is: " + numTimes);
}
```

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 }
    );
}
```

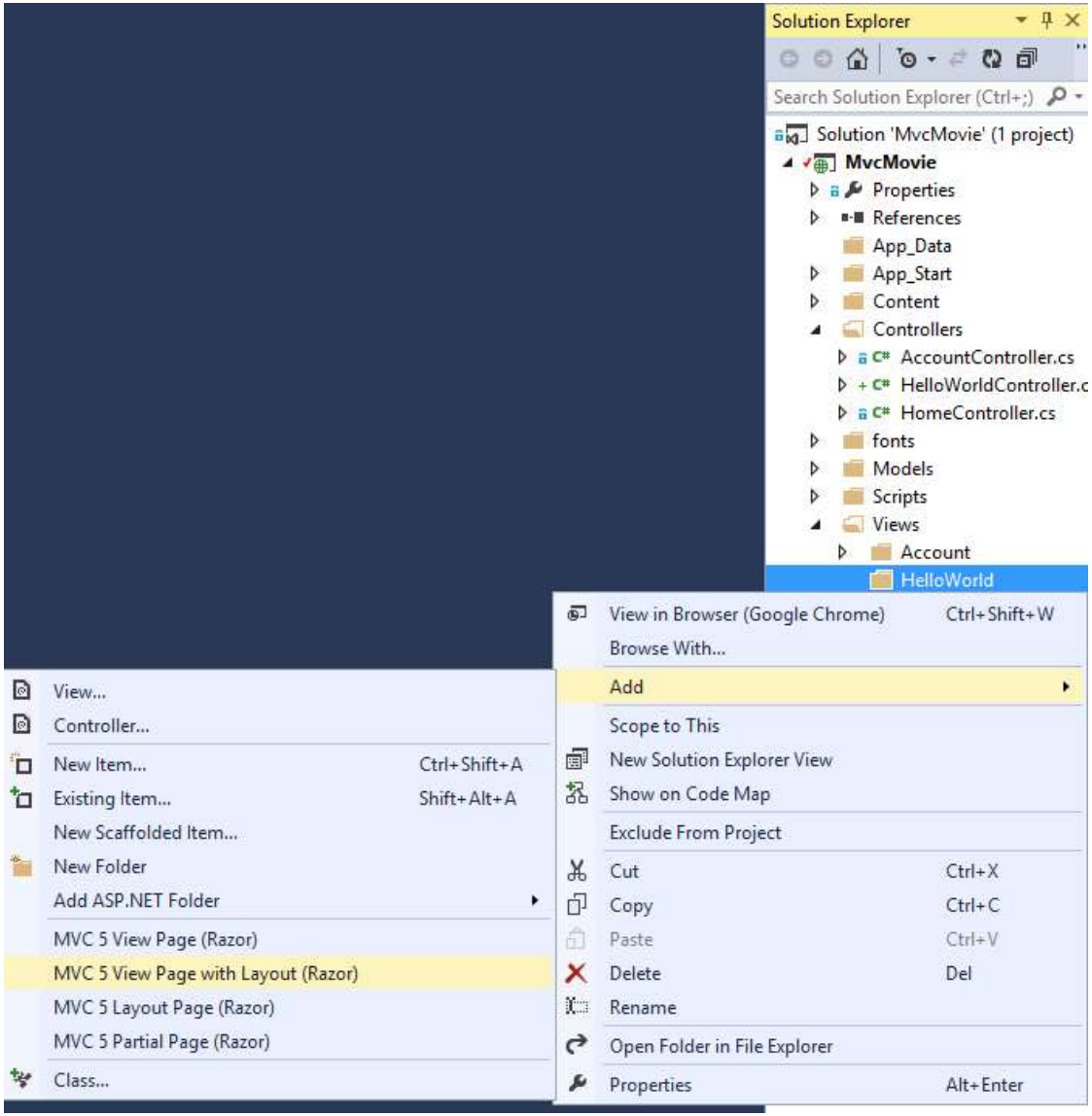
ADDING A VIEW

Views

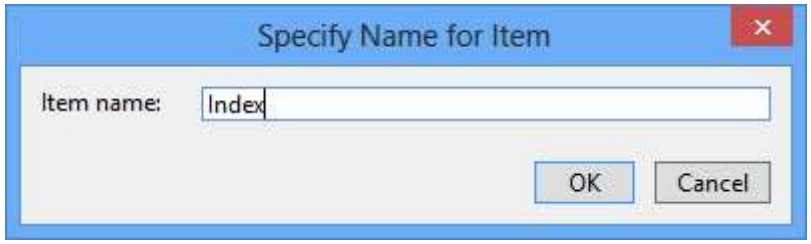
- Views created using Razor view engine
- Controller method returns View object
- Controller method return type is ActionResult
- Common pattern: all view pages share the same master layout page

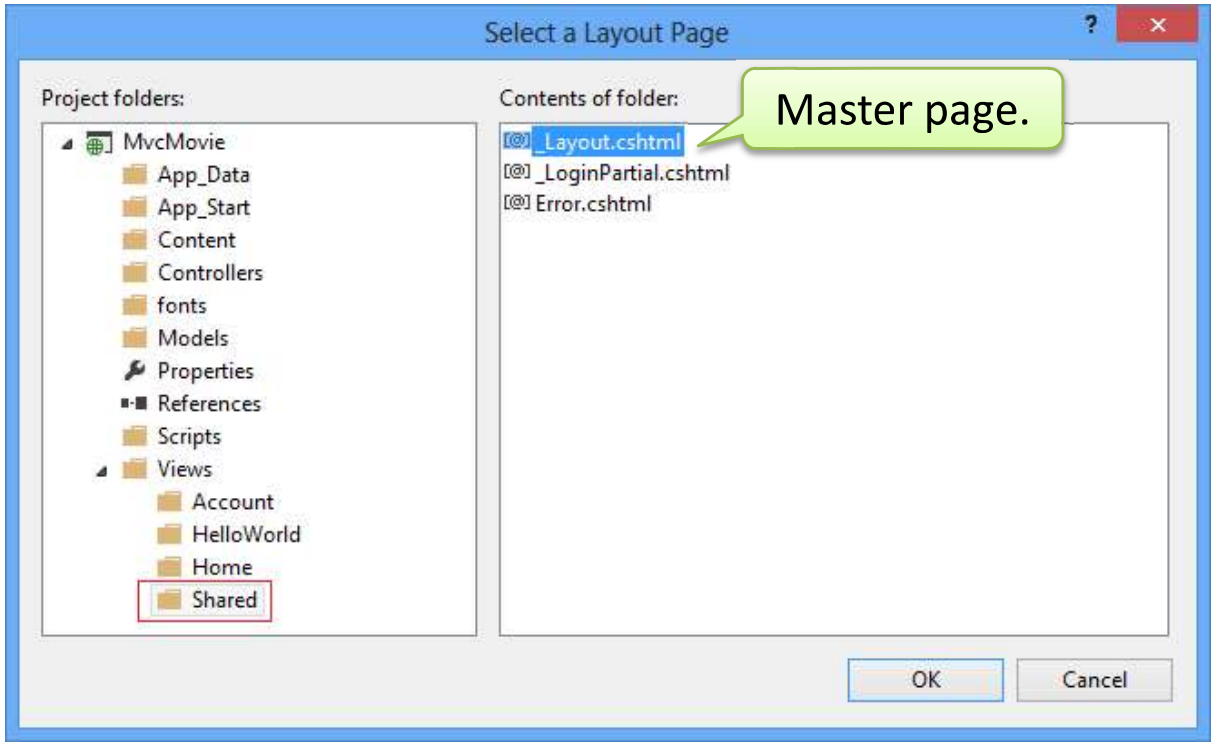
```
public ActionResult Index()
{
    return View();
}
```

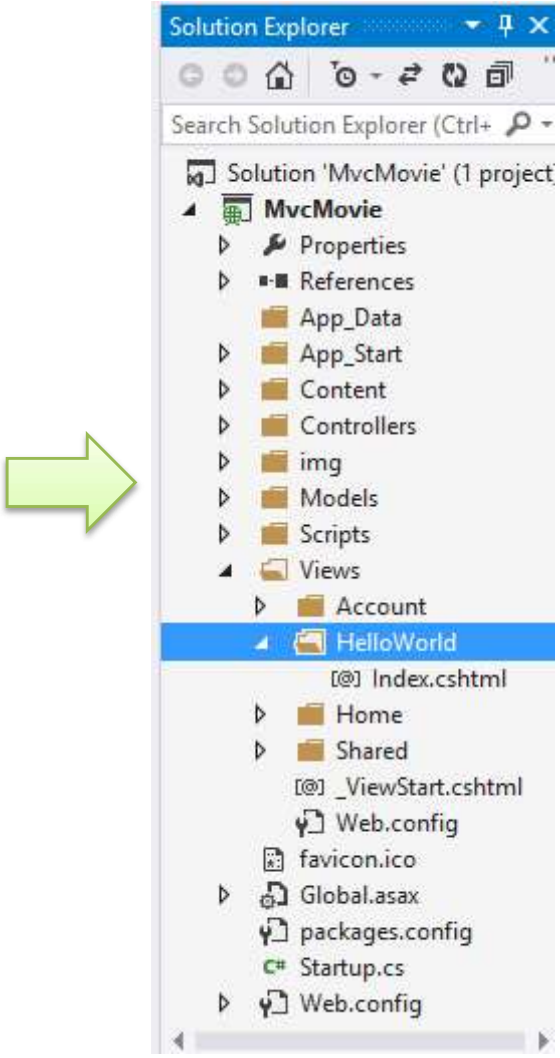
Create View page



Create View page



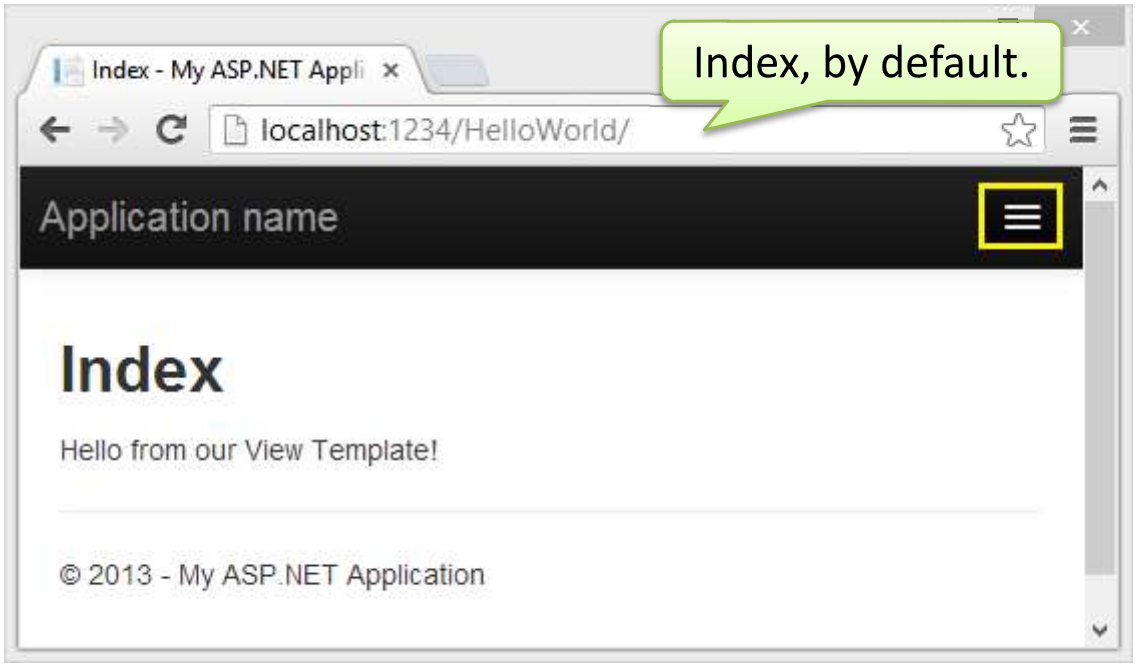




Implementing View page

Selected master page.

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

@{
    ViewBag.Title = "Index";
}

<h2>Index</h2>

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




```
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

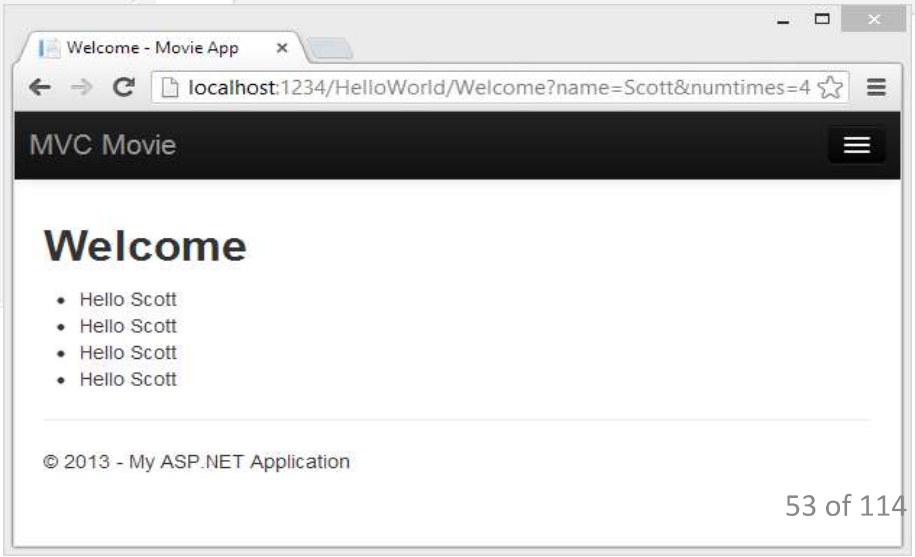
Controller

```
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>
```

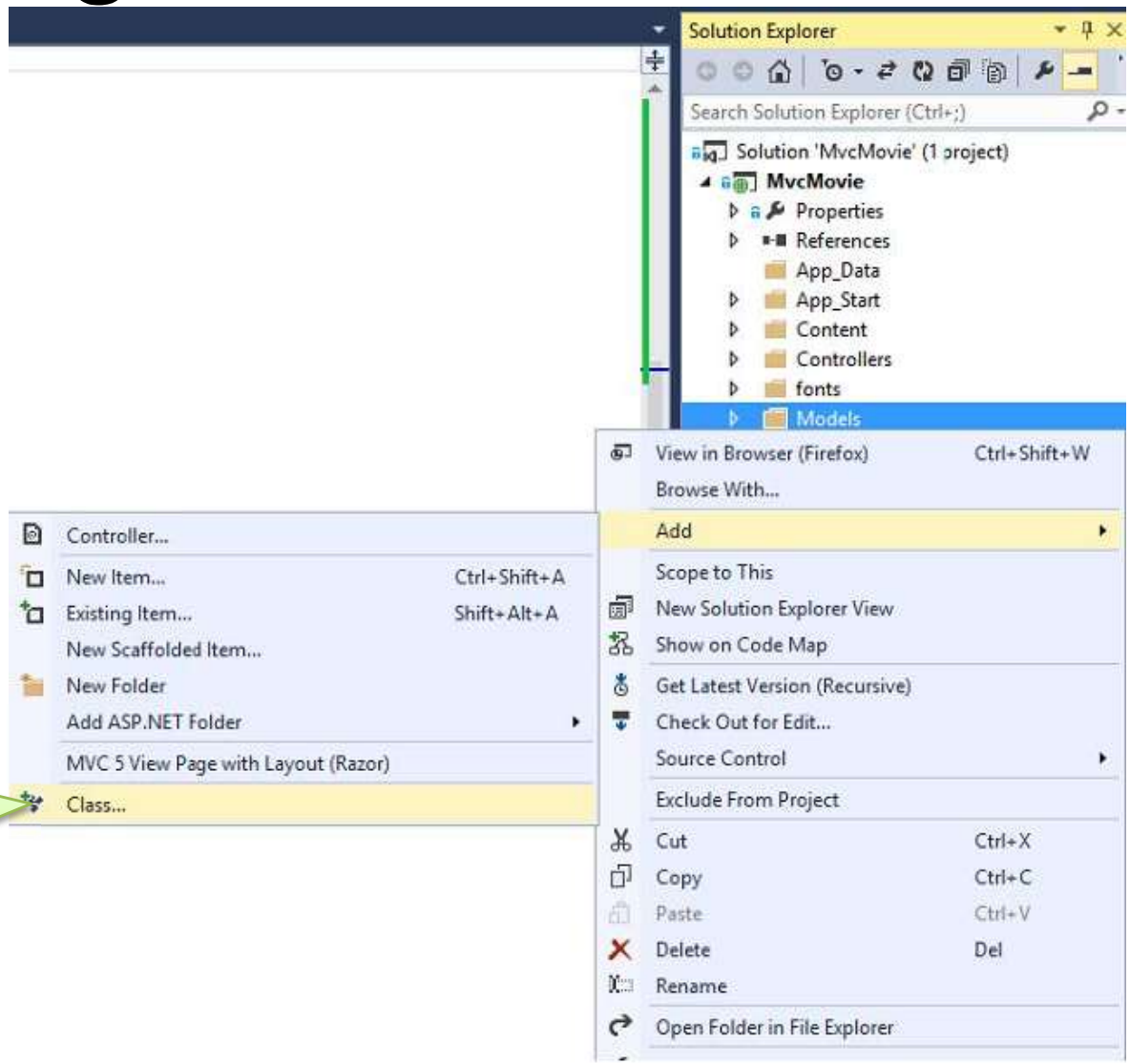


ADDING A MODEL

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; }
    }
}
```

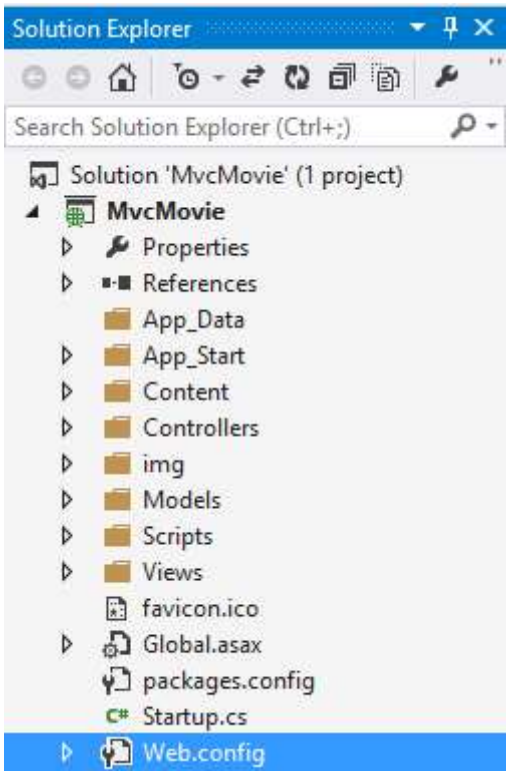
Adding a DbContext class

```
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

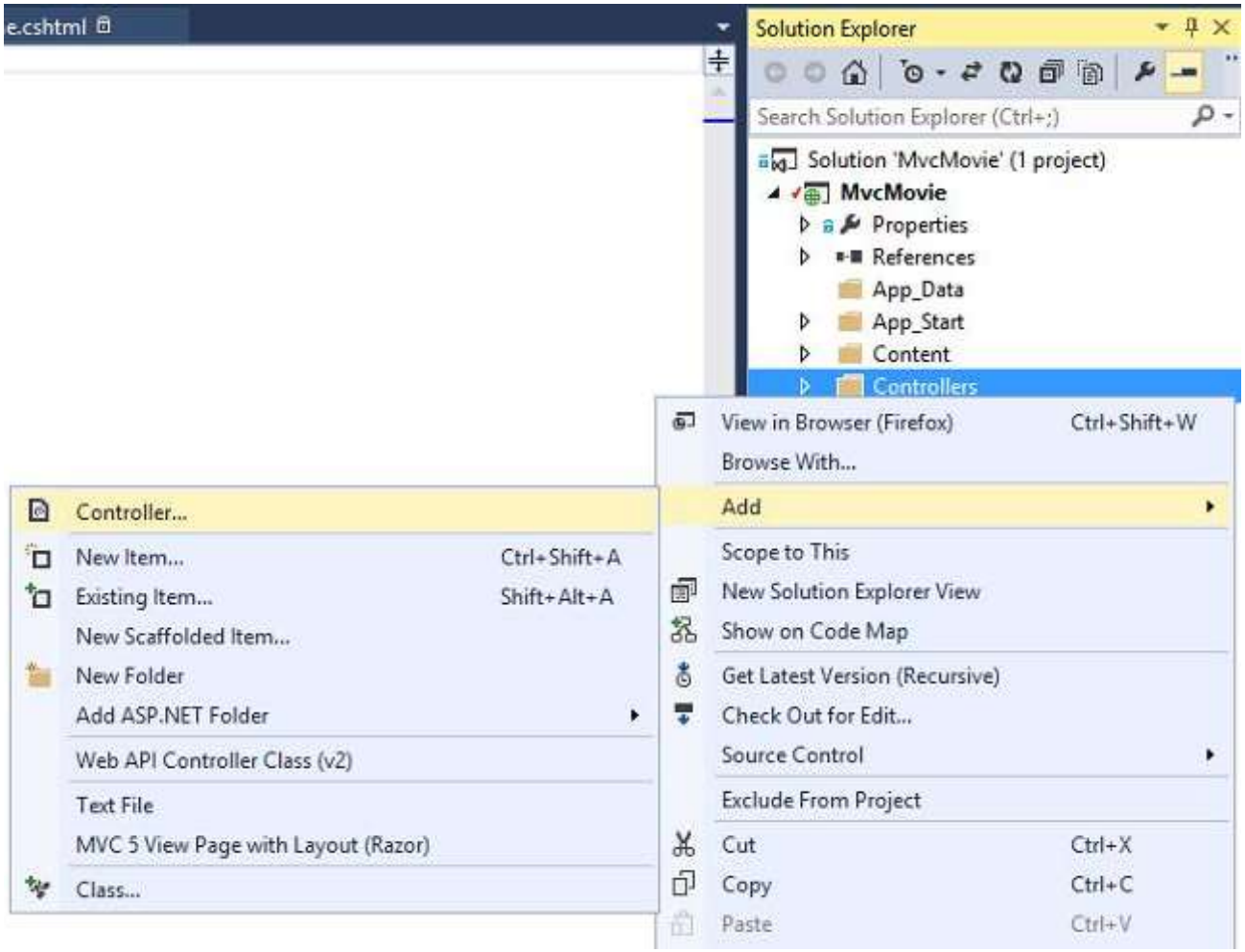



```
<?xml version="1.0" encoding="utf-8"?>
<!--
  For more information on how to configure your ASP.NET application, please visi
  http://go.microsoft.com/fwlink/?LinkId=301880
  -->
<configuration>
  <configSections>
    <!-- For more information on Entity Framework configuration, visit http://go
    <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>
```

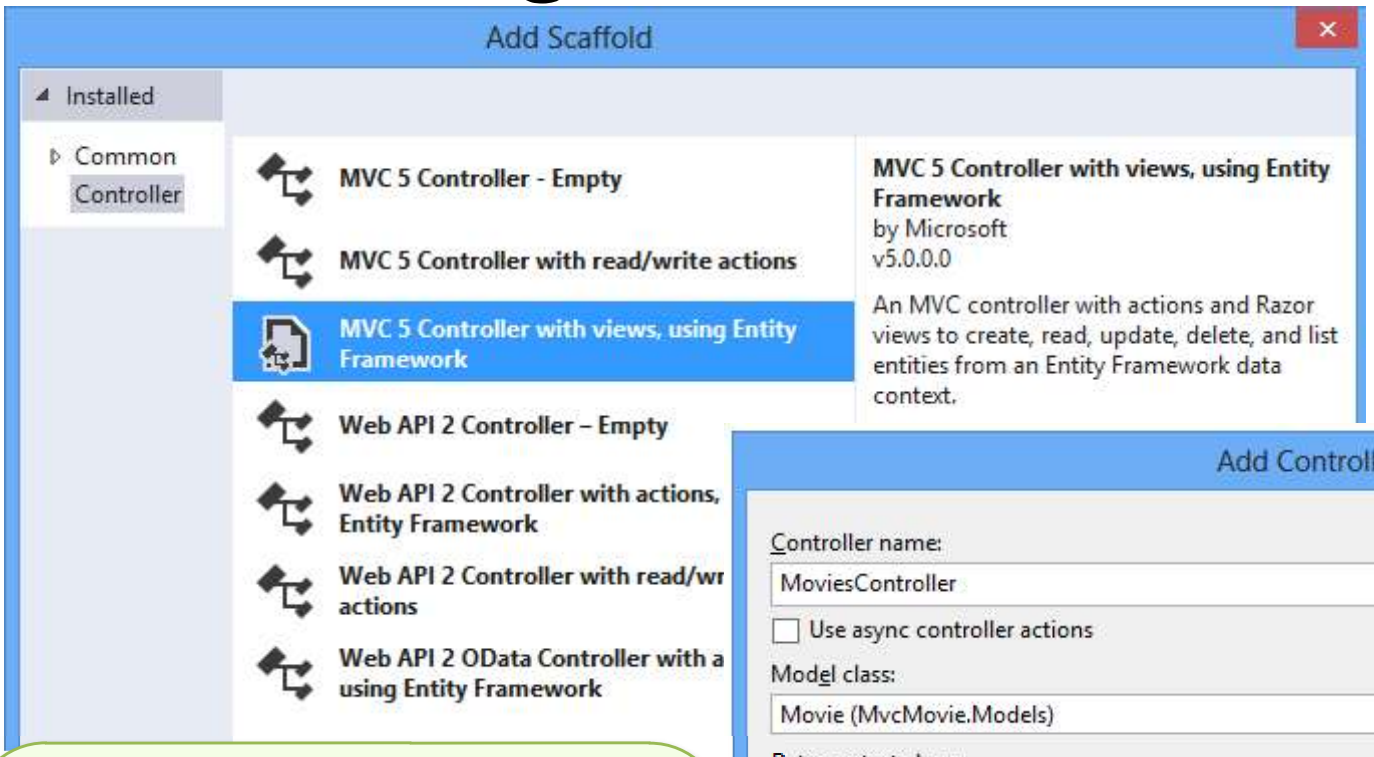
Separate connection string for each DbContex class

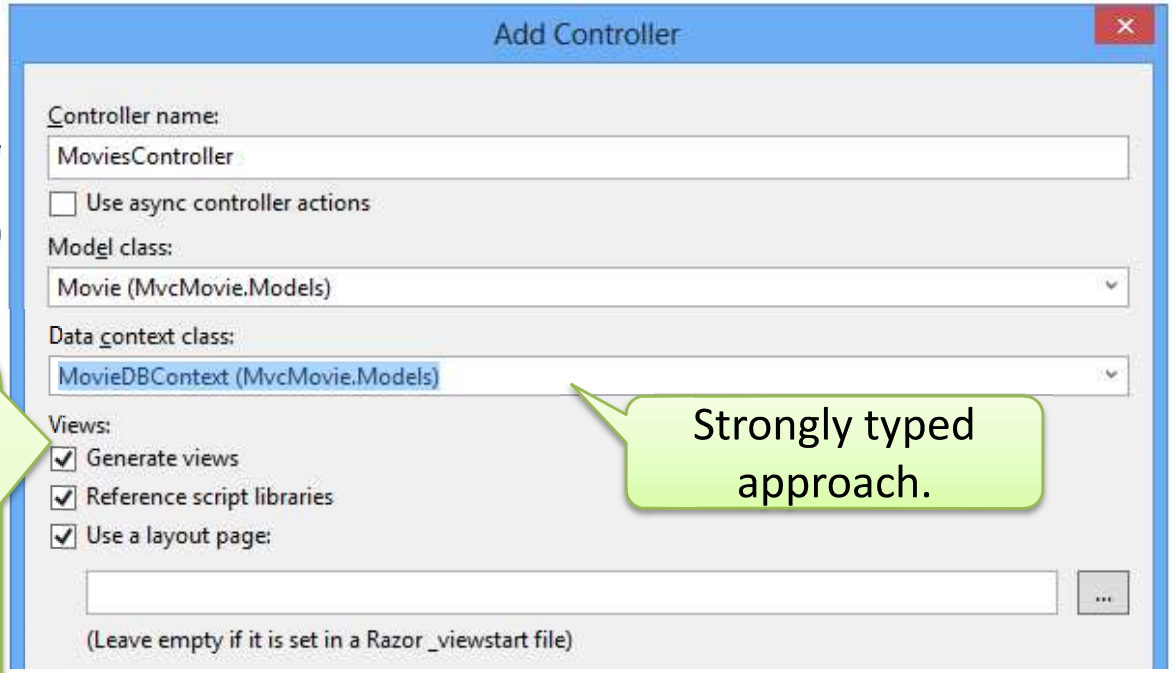
```
<connectionStrings>
    <add    name="MovieDBContext"
            connectionString="Data Source=(LocalDB)\v11.0;AttachDbFilename=|DataDirectory|\Movies.mdf;Integrated Security=True"
            providerName="System.Data.SqlClient"/>
    <add    name="DefaultConnection"
            connectionString="Data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

Accessing Model from a Controller



Accessing Model from a Controller

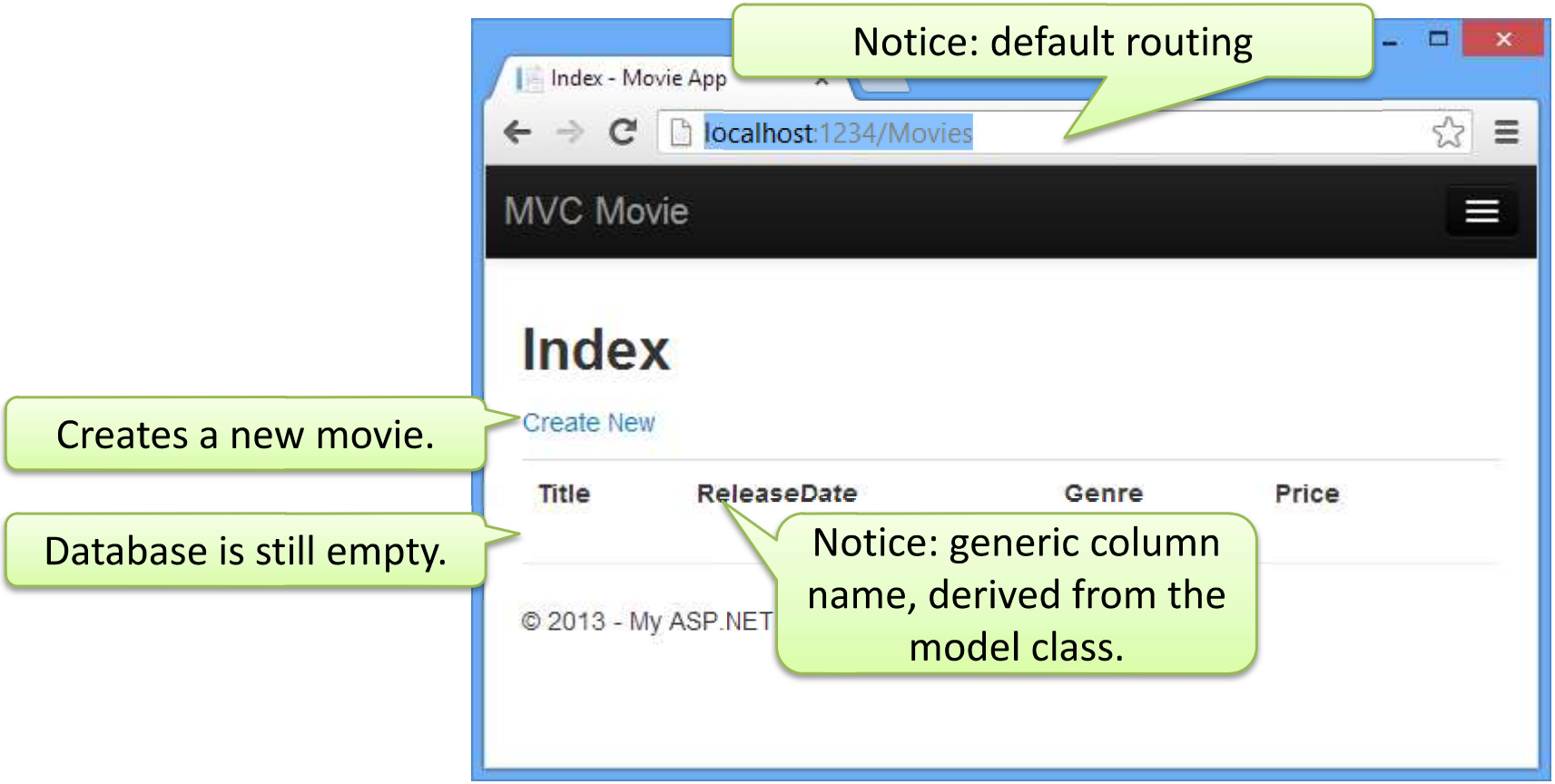




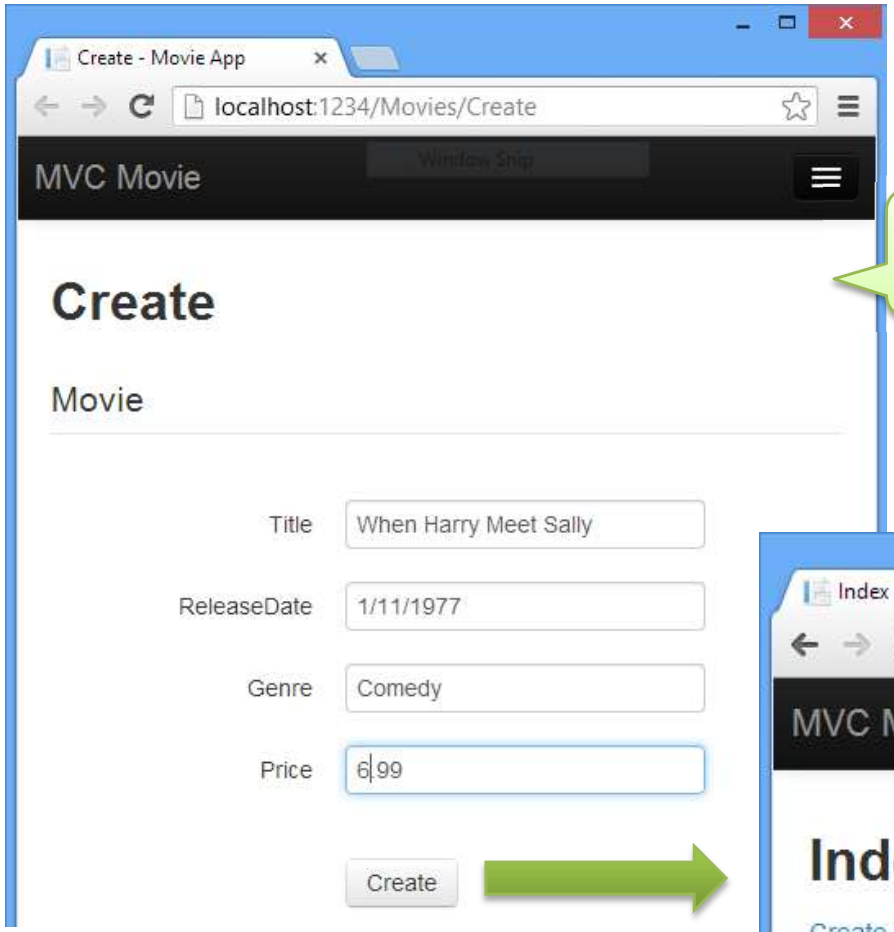
Strongly typed approach.

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

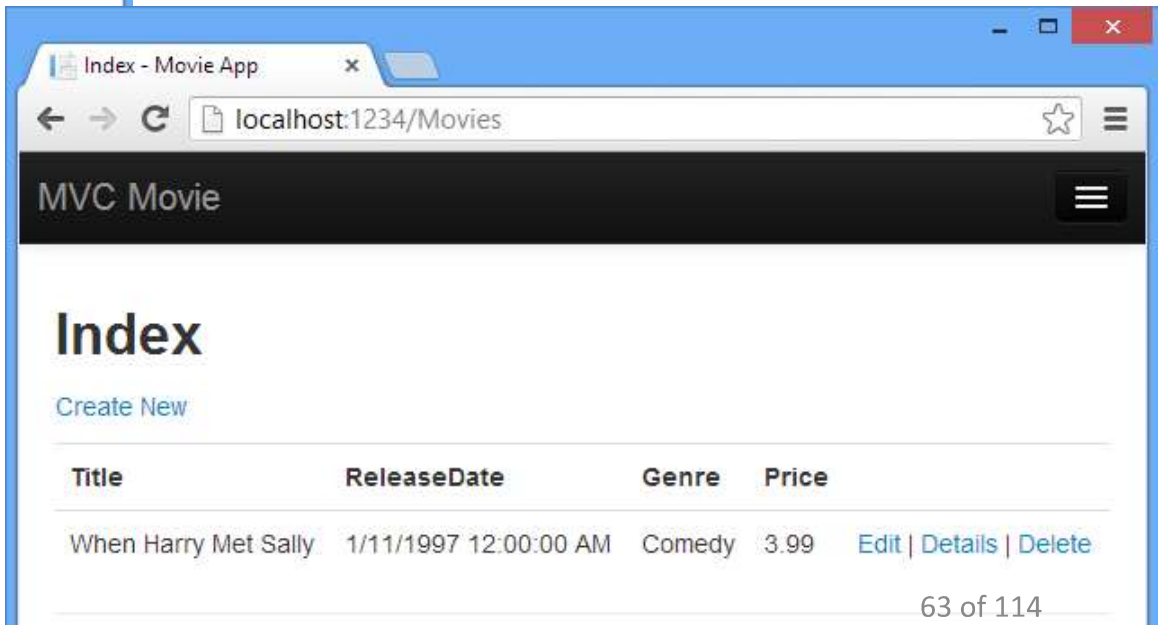
Run Application…



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

- MVC provides strongly typed way of passing data from Controller to View
- Better compile-time checking
- Richer IntelliSense in VS code editor

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.)

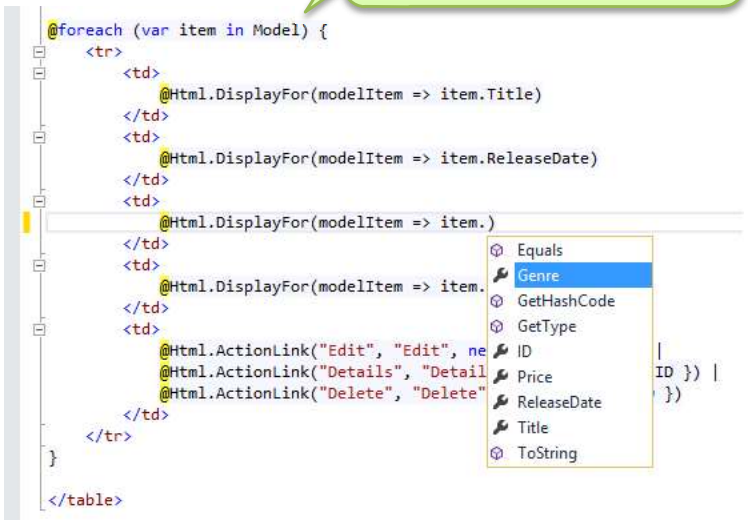
Index.cshtml

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

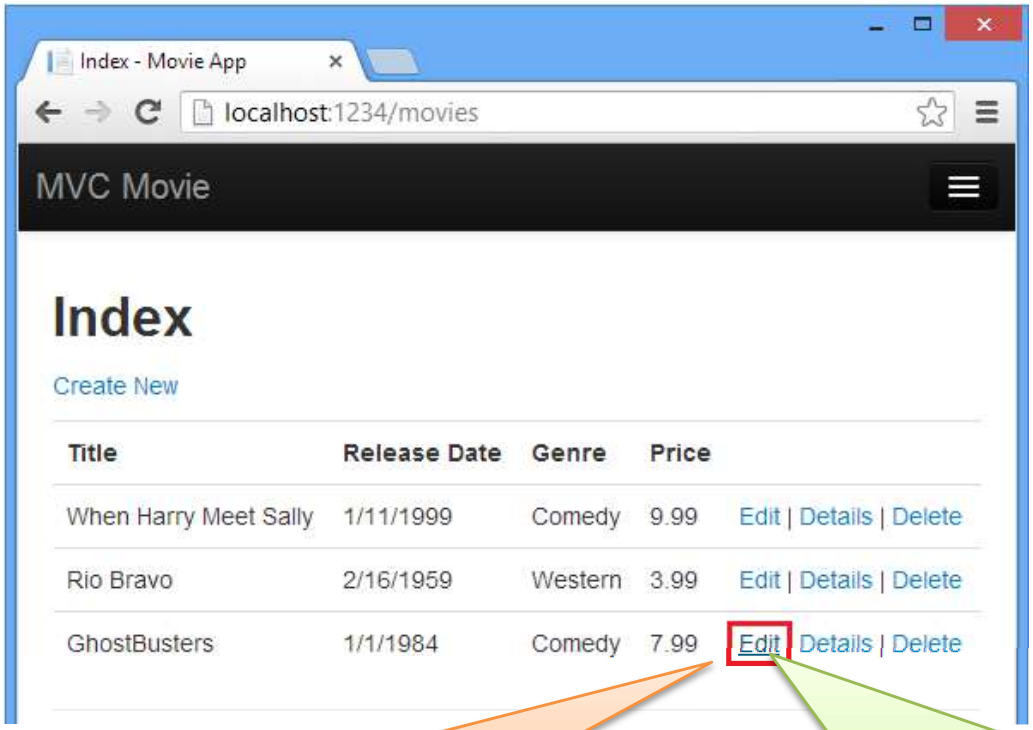
Model object is strongly typed.
Each item is a Movie object.

```
@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>
}
```

Full compile-time support.



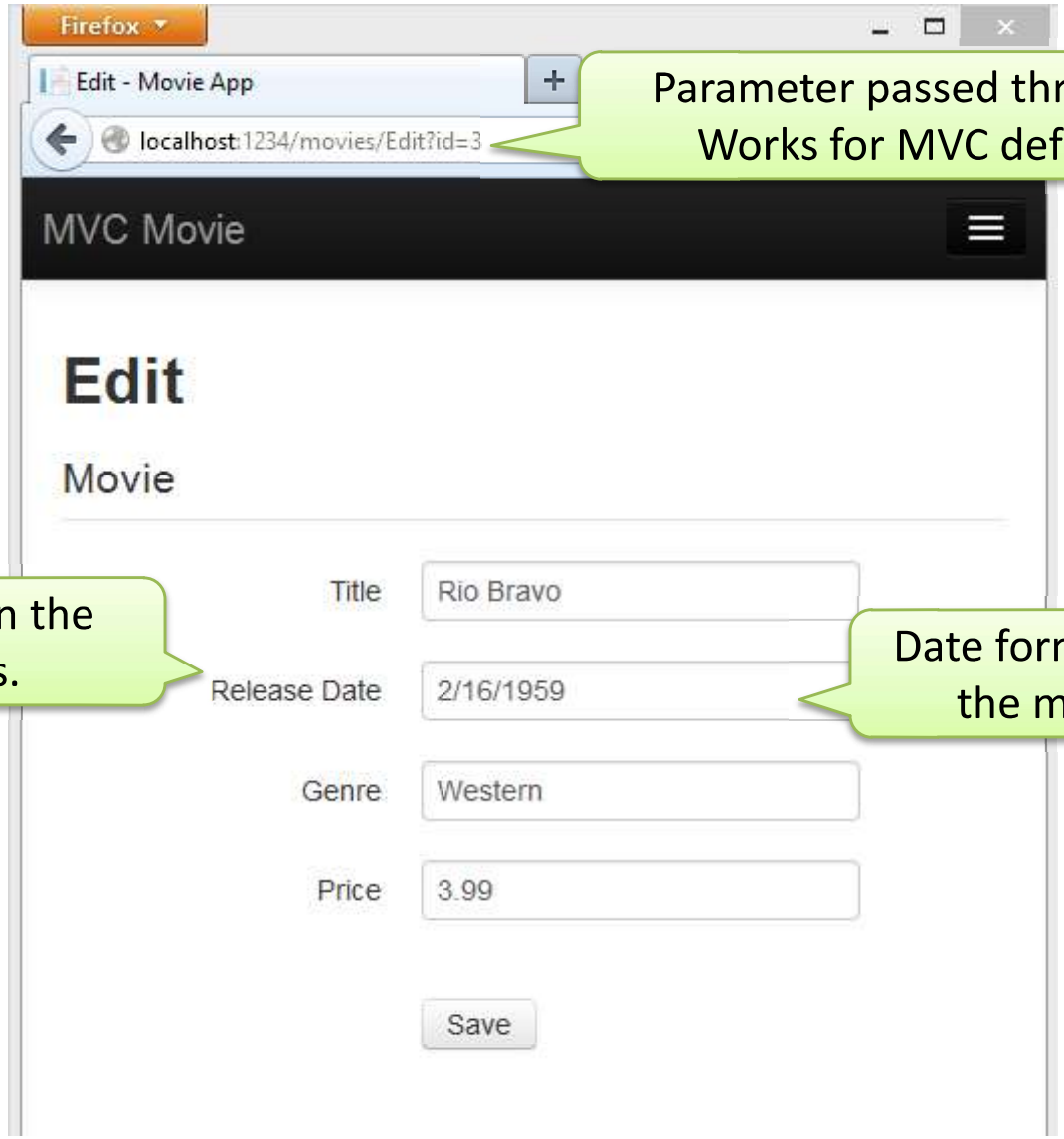
Edit View



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 ☺

ActionLink helper

- Html.ActionLink – generates a link according to a given URL mapping policy
- Primer:
  Html.ActionLink(“Edit", “Edit", new{id=item.ID)}

Anonymous object – specifies ID of an object

Controller action name.

```
<td>
    @Html.ActionLink("Edit Me", "Edit", new { id=item.ID }) |
    @Html.
    @Html.
```

(extension) MvcHtmlString HtmlHelper.ActionLink(string linkText, string actionName, object routeValues)
Returns an anchor element (a element) that contains the virtual path of the specified action.

Exceptions:
System.ArgumentException

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.

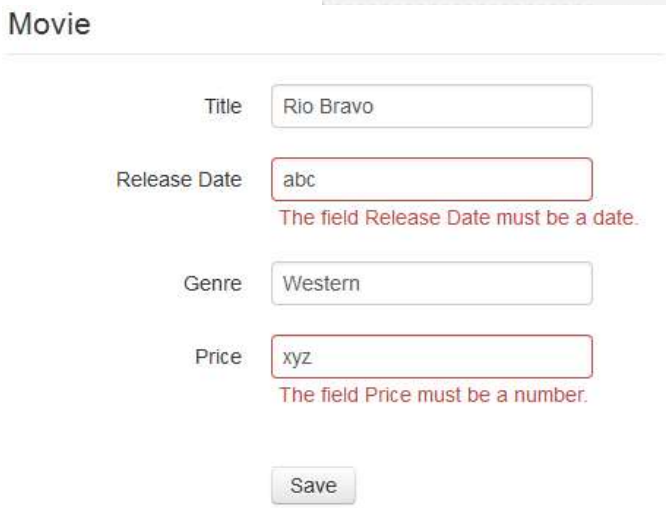
Validates the forgery token.

```
[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

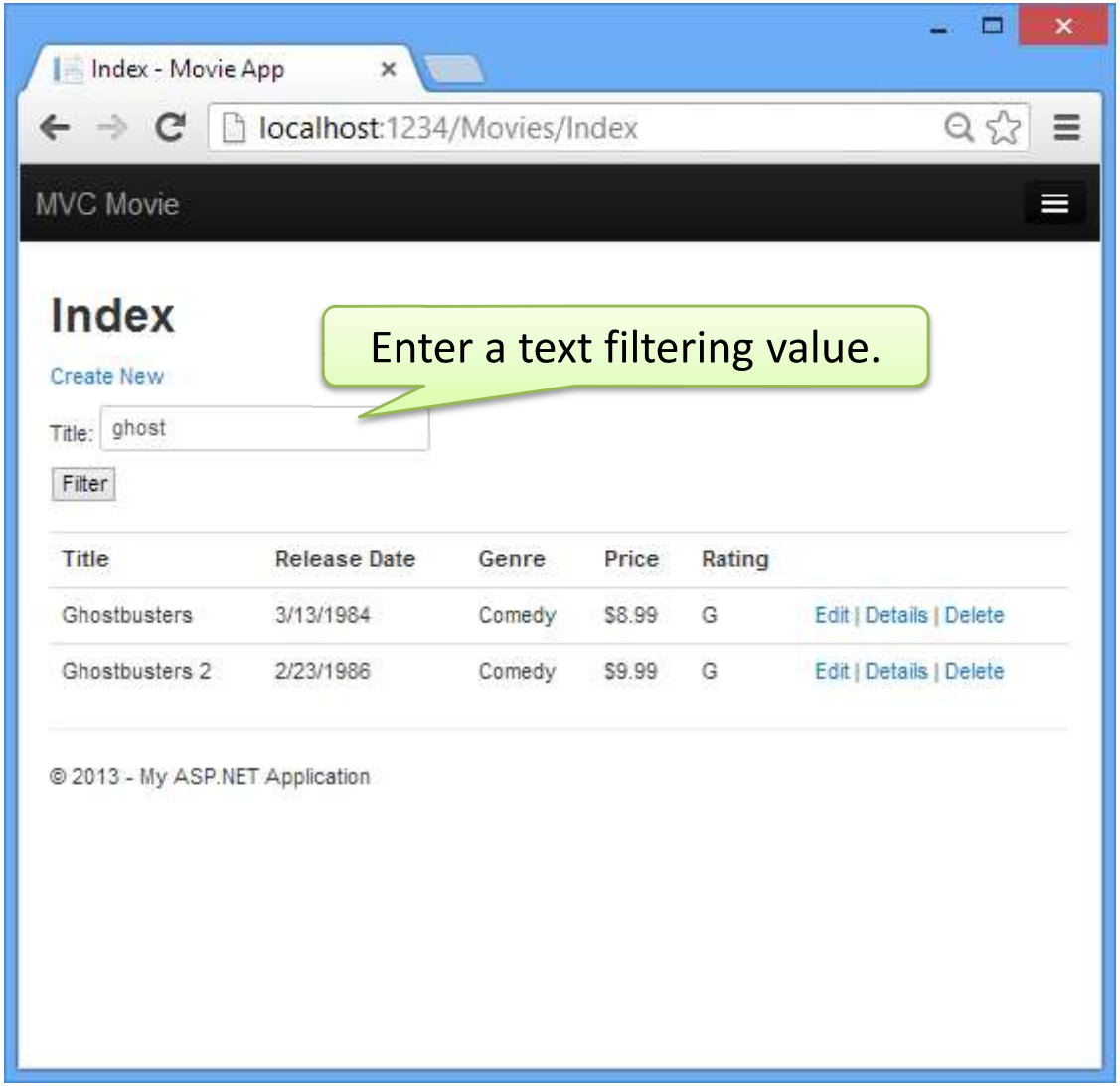


HTTP methods – best practices

- 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

ADDING SEARCH

Search form – Index.cshtml



View/Controller – changes

View (changes)

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

@{
    ViewBag.Title = "Index";
}

<h2>Index</h2>

<p>
    @Html.ActionL                                    ")

     @using (Html.BeginForm()){
         <p> Title: @Html.TextBox("SearchString") <br />
         <input type="submit" value="Filter" /></p>
        }
</p>
```

Default form method = POST!

Controller – changed signature of the method Index.

```
public ActionResult Index(string searchString)
{
     var movies = from m in db.Movies
                  select m;

    if (!String.IsNullOrEmpty(searchString))
    {
        movies = movies.Where(s => s.Title.Contains(searchString));
    }

    return View(movies);
}
```

LINQ query definition (NOT execution!)

Lambda expression

Use overriden BeginForm method to force HttpGet method.

```
@using (Html.BeginForm("Index","Movies",FormMethod.Get))
```

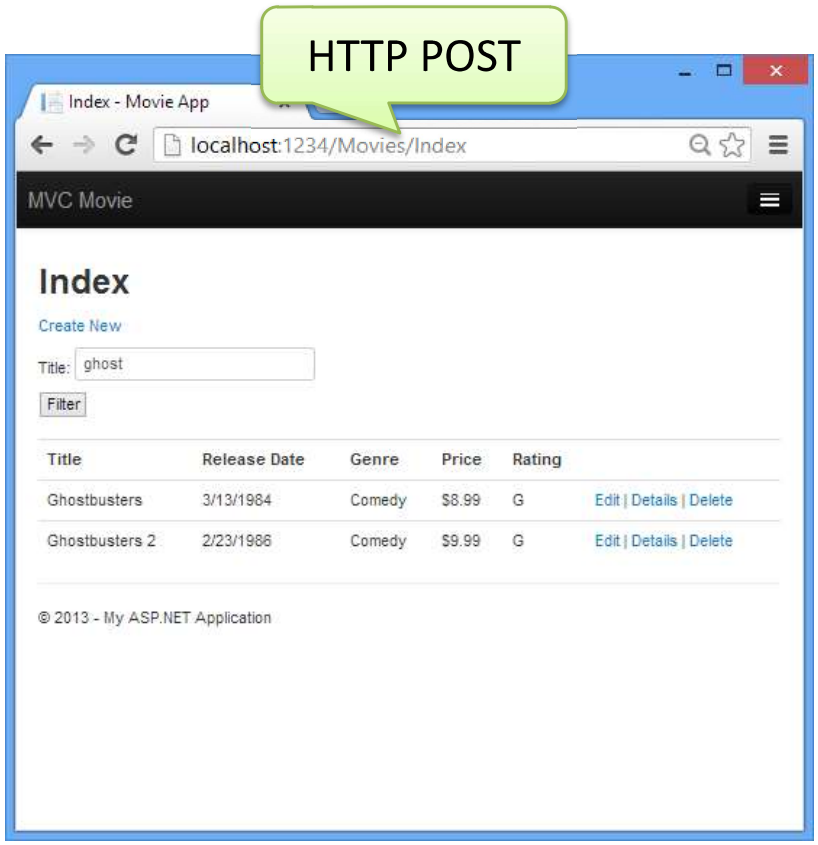
```
@Html.ActionLink("Create New", "Create")
@using (Html.BeginForm("SearchIndex","Movies",FormMethod.Get))
{
    <p>
@*  <in
    </in
    }
```

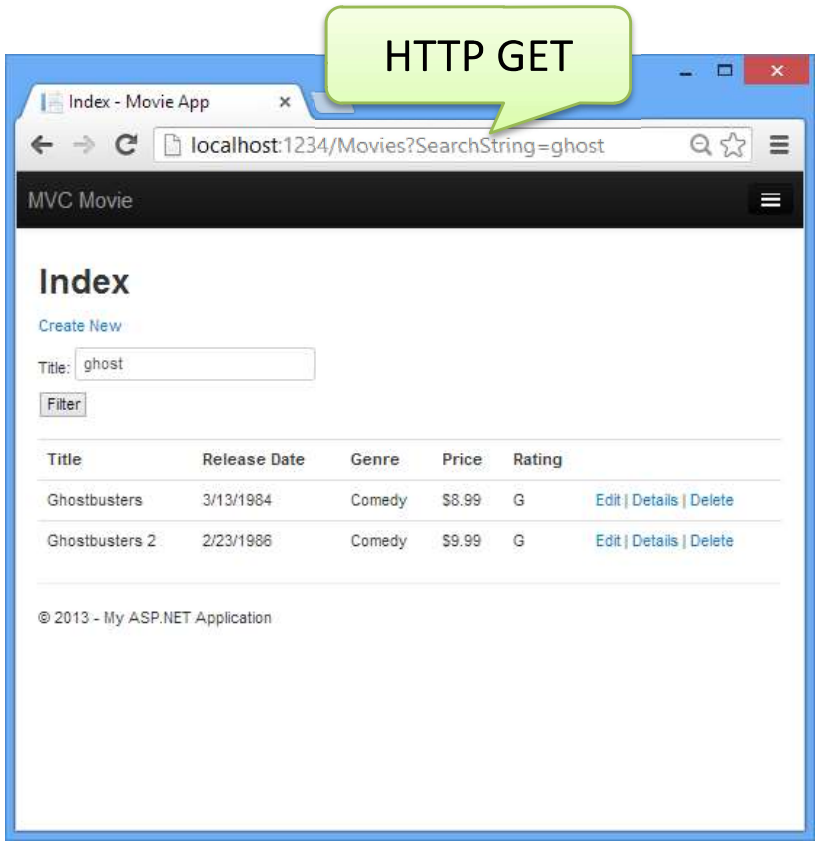
▲ 5 of 13 ▼ (extension) MvcForm HtmlHelper.BeginForm(string actionName, string controllerName, **FormMethod method**)

Writes an opening <form> tag to the response. When the user submits the form, the request will be processed by an action method.

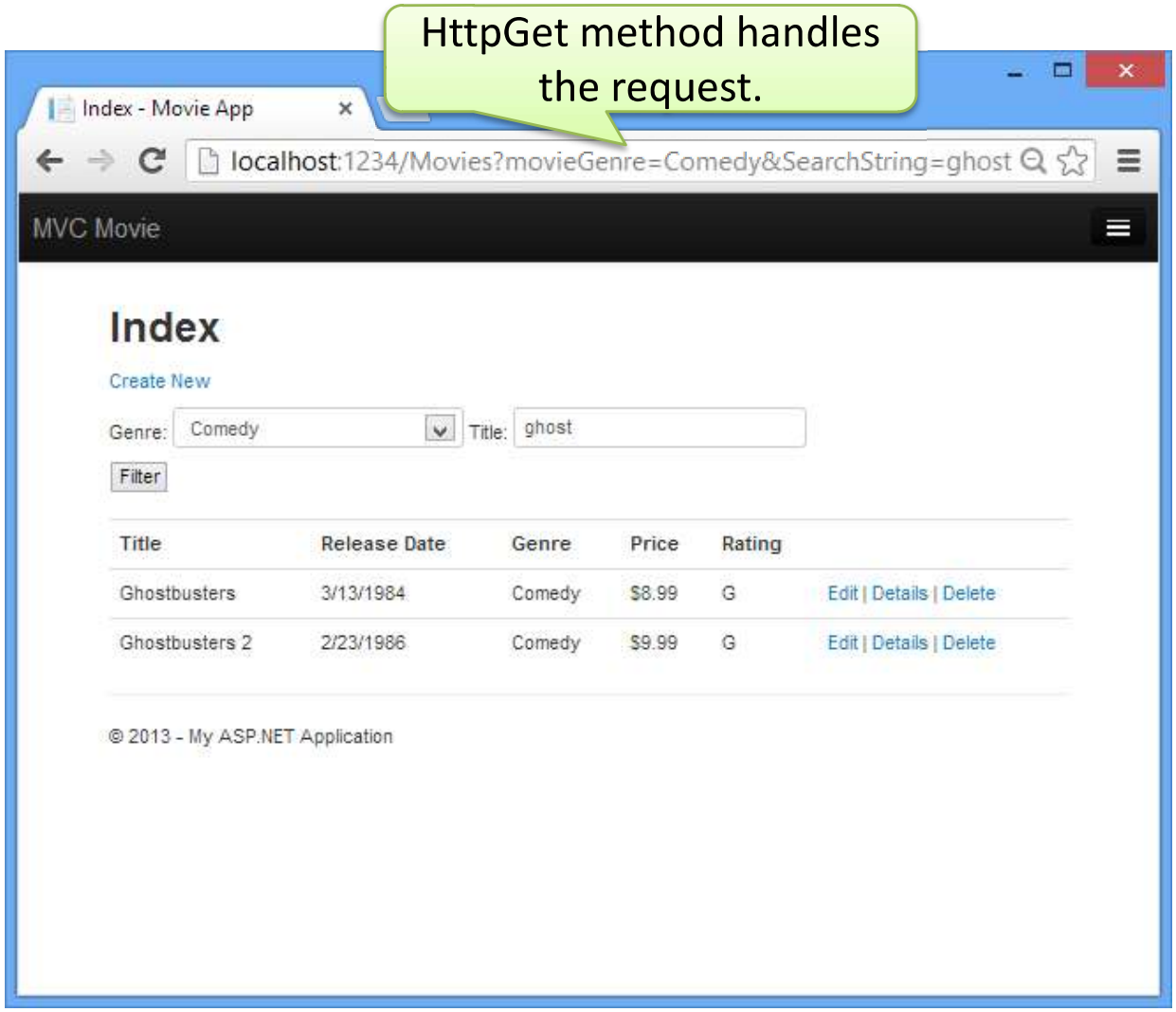
method: *The HTTP method for processing the form, either GET or POST.*

Searching movies – URL query





Adding search by Genre



Search by Genre – View

```
@model IEnumerable<MvcMovie.Models.Movie>
@{
    ViewBag.Title = "Index";
}
<h2>Index</h2>
<p>
    @Html.ActionLink("Create New", "Create")
    @using (Html.BeginForm("Index", "Movies", FormMethod.Get))
    {
    <p>
        Genre: @Html.DropDownList("movieGenre", "All")
        Title: @Html.TextBox("SearchString")
        <input type="submit" value="Filter" />
    </p>
    }
</p>
<table class="table">
```

DropDown list markup.

Preselected value.

Parameter “movieGenre” is the key for populating dropdown list from ViewBag.

Search by Genre – Controller

```
public ActionResult Index(string movieGenre, string searchString)
{
    var GenreLst = new List<string>();

    var GenreQry = from d in db.Movies
                   orderby d.Genre
                   select d.Genre;

    GenreLst.AddRange(GenreQry.Distinct());
    ViewBag.movieGenre = new SelectList(GenreLst);

    var movies = from m in db.Movies
                 select m;

    if (!String.IsNullOrEmpty(searchString))
    {
        movies = movies.Where(s => s.Title.Contains(searchString));
    }

    if (!string.IsNullOrEmpty(movieGenre))
    {
        movies = movies.Where(x => x.Genre == movieGenre);
    }

    return View(movies);
}
```

Populating the list of genres in ViewBag.

Key movieGenre is the same as the parameter of the dropdown list.

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);
}
```

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

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.

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!

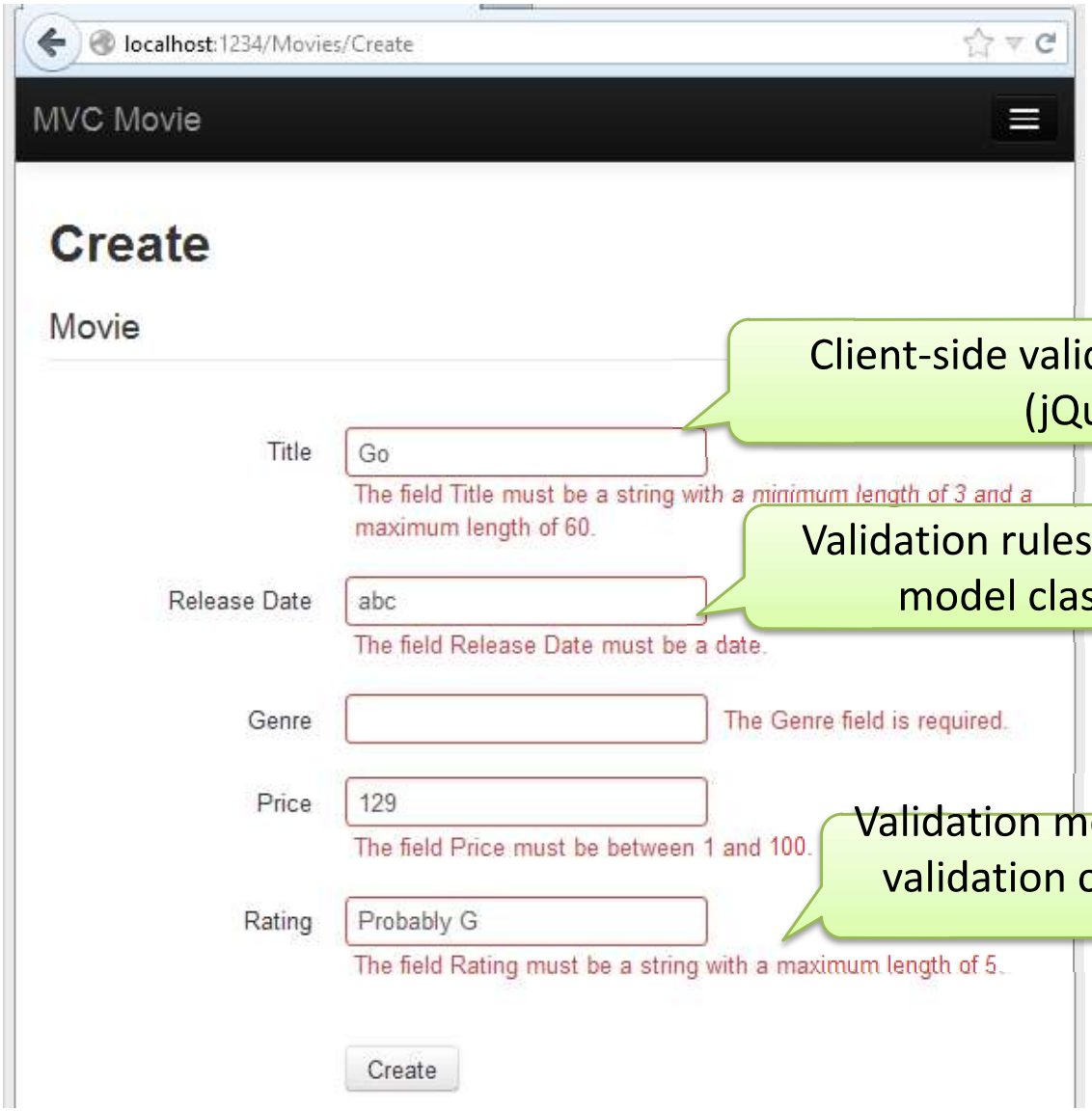
Validation rules – Model

```
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 mc
{
    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!

LAMBDA EXPRESSIONS

Introduction

- Expressions that use special syntax
- Anonymous functions used as data (variables, fields, parameters, return values)
- The anonymous functions are used to create delegates and expression trees
- Lambda expressions particularly helpful for writing LINQ queries
- Available from .NET 4.5

Operator =>

- Interpreted as “goes to”
- Used for declaring a lambda expression
- The same priority as assignment (=)
- Right associative operator
- Separates the parameters and function body

Left side	**=>**	**Right side**
An Empty parameter list		An expression
A formal parameter list		A Statement list inside curly brackects.
An implicit parameter list		

Anonymous functions

- Inline statements or expressions
- Used wherever a delegate type is expected
- It can initialize a named delegete
- It can be passed as the parameter where a named delegate type is expected
- Two kinds of anonymous functions
  - Anonymous methods
  - Lambda expressions

Evolution of delegates in C#

```
delegate void TestDelegate(string s);
static void M(string s)
{
    Console.WriteLine(s);
}

static void Main(string[] args)
{
    // Original delegate syntax required
    // initialization with a named method.
    TestDelegate testDelA = new TestDelegate(M);

    // C# 2.0: A delegate can be initialized with
    // inline code, called an "anonymous method." This
    // method takes a string as an input parameter.
    TestDelegate testDelB = delegate(string s) { Console.WriteLine(s); };

    // C# 3.0. A delegate can be initialized with
    // a lambda expression. The lambda also takes a string
    // as an input parameter (x). The type of x is inferred by the compiler.
    TestDelegate testDelC = (x) => { Console.WriteLine(x); };

    // Invoke the delegates.
    testDelA("Hello. My name is M and I write lines.");
    testDelB("That's nothing. I'm anonymous and ");
    testDelC("I'm a famous author.");

    // Keep console window open in debug mode.
    Console.WriteLine("Press any key to exit.");
    Console.ReadKey();
}
```

Named method

Inline code (anonymous method)

Lambda expression

Anonymous method

- No name, no overloading
- Created using the `delegate` keyword
- It is possible to add multiple statements inside its body

```
// Create a handler for a click event.
button1.Click += delegate(System.Object o, System.EventArgs e)
                 { System.Windows.Forms.MessageBox.Show("Click!"); };
```

```
// Create a delegate.
delegate void Del(int x);

// Instantiate the delegate using an anonymous method.
Del d = delegate(int k) { /* ... */ };
```

```
void StartThread()
{
    System.Threading.Thread t1 = new System.Threading.Thread
      (delegate()
            {
                System.Console.Write("Hello, ");
                System.Console.WriteLine("World!");
            });
    t1.Start();
}
```

Anonymous method (cont.)

- Scope of the parameters is the anonymous method block
- No jump from inside an anonymous method block to the outside, and vice versa.
- Cannot access `ref` and `out` parameters of an outer scope
- No unsafe code access inside its block
- Not allowed on the left side of the operator `is`.

Expression lambdas

- Lambda expression with an expression on the right side of the operator =>
- Used dominantly in construction of expression trees
- **`(input parameters) => expression`**
- Parentheses optional if lambda has one param.
- Input parameters separated by comma

Expression lambdas - examples

- **`(x, y) => x == y`**
  The parameters types inferred by the compiler
- **`(int x, string s) => s.Length > x`**
  Specify types of the parameters when the compiler cannot inferre them from the code.
- **`() => SomeMethod()`**
  Zero input parameters specified with empty parentheses.
  Note: a method call cannot be evaluated outside the .NET Framework (e.g. SQL Server)

Statement lambdas

- **`(input parameters) => {statement;}`**
- Statements enclosed in braces
- The body of a statement lambda can contain multiple statements (in practices, two-three)
- Cannot be used to create expression trees

```
delegate void TestDelegate(string s);
…
TestDelegate myDel = n => { string s = n + " " + "World"; Console.WriteLine(s); };
myDel("Hello");
```

Generic delegates – Func

- **`System.Func<T,TResult>`**
  T – argument type,
  TResult – return type (last type parameter)
- Useful for encapsulating user-defined expressions that are applied to all elements of a data set

A generic declaration of the delegate Func.

```
public delegate TResult Func<TArg0, TResult>(TArg0 arg0)
```

Example of usage.

```
Func<int, bool> myFunc = x => x == 5;
bool result = myFunc(4); // returns false of course
```

Func delegate (cont.)

- A lambda expression can be passed where Expression<Func> type is required
  - System.Linq.Queryable

```
int[] numbers = { 5, 4, 1, 3, 9, 8, 6, 7, 2, 0 };
int oddNumbers = numbers.Count(n => n % 2 == 1);
```

Output: 5, 1, 3, 9, 7

Compiler can infere the type of the parameter n.

```
var firstNumbersLessThan6 = numbers.TakeWhile(n => n < 6);
```

Output: 5, 4, 1, 3

```
var firstSmallNumbers = numbers.TakeWhile((n, index) => n >= index);
```

Output: 5, 4

Type inference in lambdas

- Compiler can infer the type of the parameters based on:
  - Lambda's body
  - Parameter's delegate type
- Example:

```
IEnumerable<Customer> customers=...
customers.Where(c => c.City == "London");
```

Standard query operator.

Lambda expressions – general rules

- The lambda must contain the same number of parameters as the delegate type
- Each input parameter in the lambda must be implicitly convertible to its corresponding delegate parameter
- The return value of the lambda (if any) must be implicitly convertible to the delegate's return type

Lambda expressions - examples

- **Func<int,int> f1 = x => x+1;**
- **Func<int,int> f2 = x => {return x+1;}**
- **Func<int,int> f3 = (int x) => x +1;**
- **Func<int,int> f4 = (int x) => {return x+1;}**
- **Func<int,int> f7 = delegate(int x) {return x+1;}**
- Invocation example:
  Console.Writeln(f1.Invoke(4));

Lambda expressions - examples

- **Func<int,int,int> f5= (x,y) => x*y**
  Invocation: Console.WriteIn(f5.Invoke(2,2));
- **Action f6 = () => Console.Writeline();**
  Function instance that does not receive any parameter nor returns value.
  Invocation: f6.Invoke();
- **Func<int> f8 = delegate { return 1+1;}**
  Invocation: Console.WriteIn(f8());

Language Integrated Query

LINQ

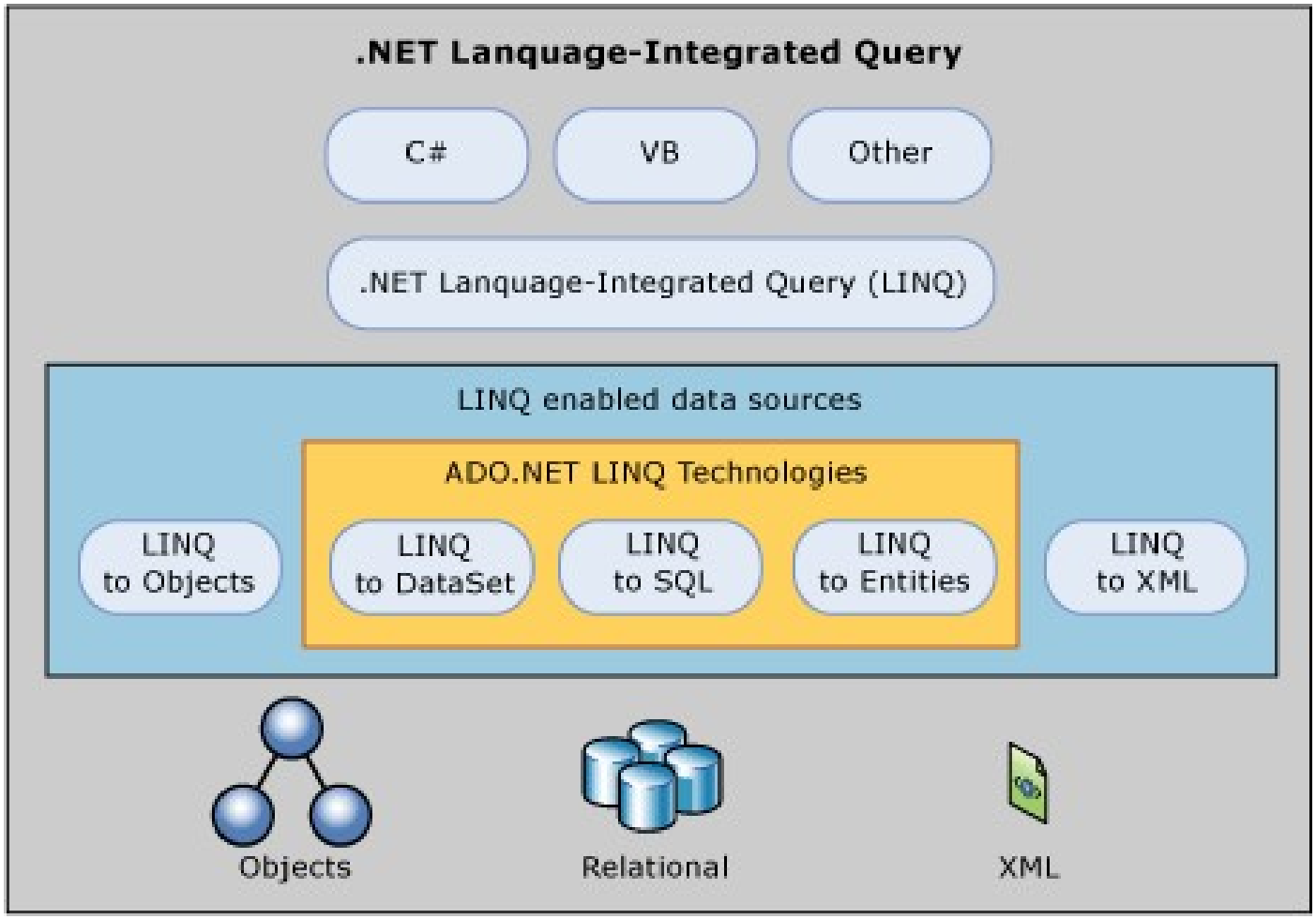
Content

- Undestand what LINQ is?
- Learn what problems solves
- See what its syntax looks like
- Know where LINQ can be used

What is LINQ?

- Language INtegrated Query
- It is part of programming language syntax
- Supported by: C#, VB, Delphi Prism
- Used for querying data
- Supported the following types of data sources
  - Relational data
  - XML data
  - objects

LINQ Architecture



ADO .NET vs. LINQ

ADO .NET

- OO library for relational data access
- Mapping from relational to OO objects needed!
- High Impedance Missmatch for mapping data from storage to objects in an application

LINQ

- SQL-Like syntax that deals with pure objects
- Reduces the Impedance Missmatch
- Makes data querying more efficient
- One still must know the format of the data

LINQ Adapters

- LINQ to Objects
- LINQ to SQL
- LINQ to XML
- LINQ to Entities
- It is possible to create own customized adapter
  - E.g. LINQ for Querying Twitter API