SoftServe

ASP.NET MVC 5. Part 1

Overview. Controllers. Views.

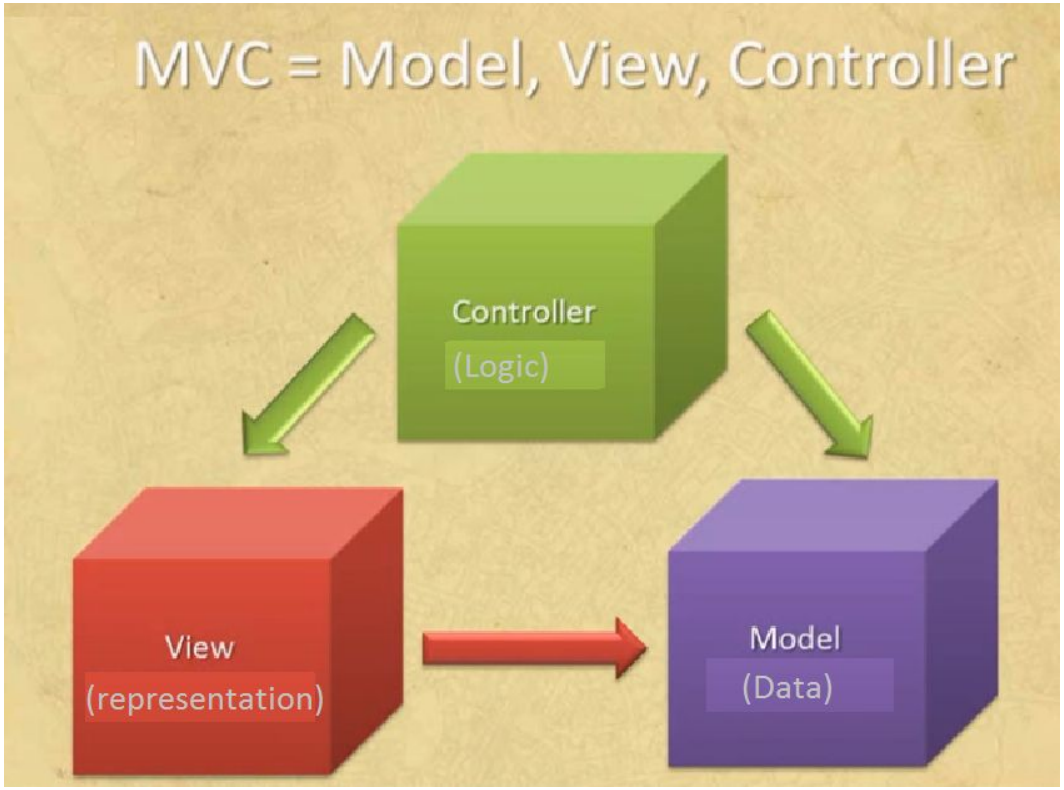
2014-11-25 by O. Shvets
Reviewed by O. Konovalenko

Agenda

SoftServe

- ASP.NET Architecture
- ASP.NET MVC 3, 4, 5
- Controllers
- Views

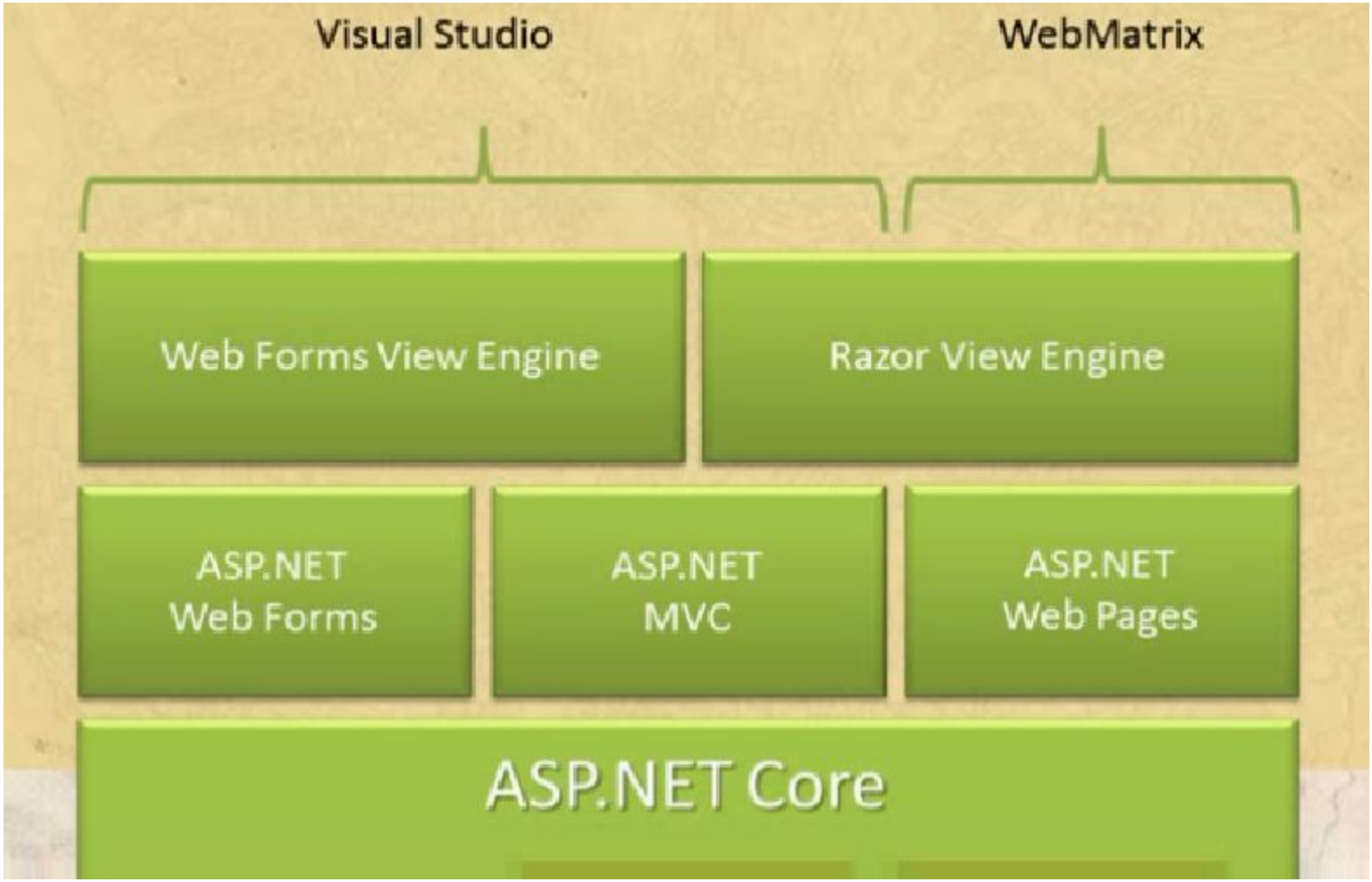
MVC Pattern



- **Controller** – application logic. Communicate with user. It receives and handles user queries, interrupts with Model, and returns results by View objects
- **Model** – contains classes that represent data, performs operations with data-bases and organizes relations between data-classes.
- **View** – performs UI representation. Works with model.

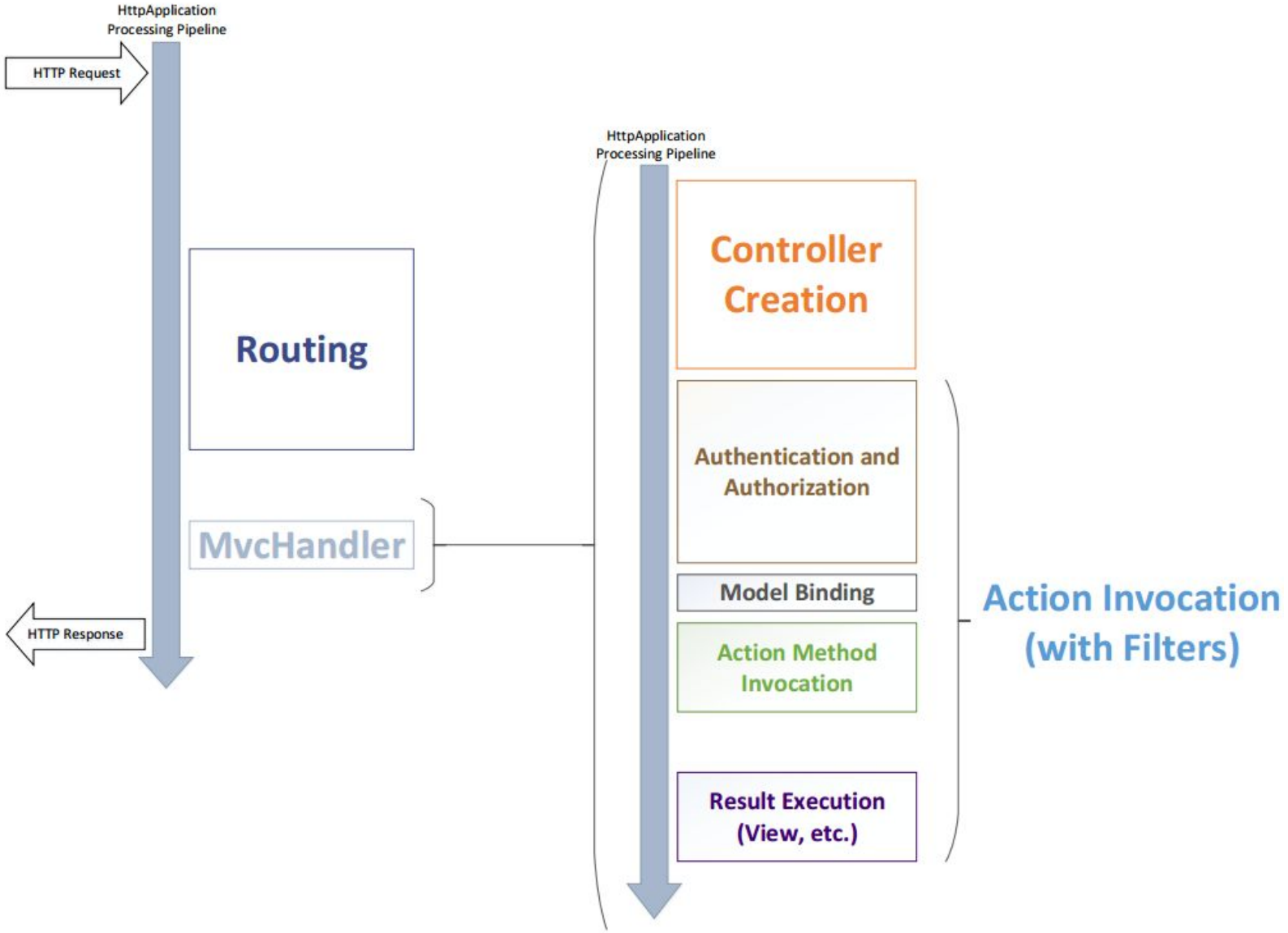
ASP.NET Architecture

SoftServe



Lifecycle of an ASP.NET MVC 5 Application

SoftServe



Benefits of ASP.NET MVC

SoftServe

- Higher quality requirements
  - Test Driven Development
- Cross platforms support
  - Windows, PDA, IPhone, …
- HTML code control
- Clear ULR navigation
  - http://musica.ua/groups/metallica
- Maintainable code and command work

What's new in ASP.NET MVC 3

SoftServe

- Extensible Scaffolding with MvcScaffold integration
- HTML 5 enabled project templates
- The Razor View Engine
- Support for Multiple View Engines
- Controller Improvements
- JavaScript and Ajax
- Model Validation Improvements
- Dependency Injection Improvements

What's new in ASP.NET MVC 4

SoftServe

- ASP.NET Web API
- Enhancements to Default Project Templates
- Mobile Project Template and Empty Project Template
- jQuery Mobile, the View Switcher, and Browser Overriding
- Task Support for Asynchronous Controllers
- Azure SDK
- Database Migrations
- Add Controller to any project folder
- Bundling and Minification
- Enabling Logins from Facebook and Other Sites Using OAuth and OpenID

What's new in ASP.NET MVC 5

SoftServe

- One ASP.NET project template
- ASP.NET Identity
- Bootstrap
- Authentication filters
- Filter overrides
- Attribute routing

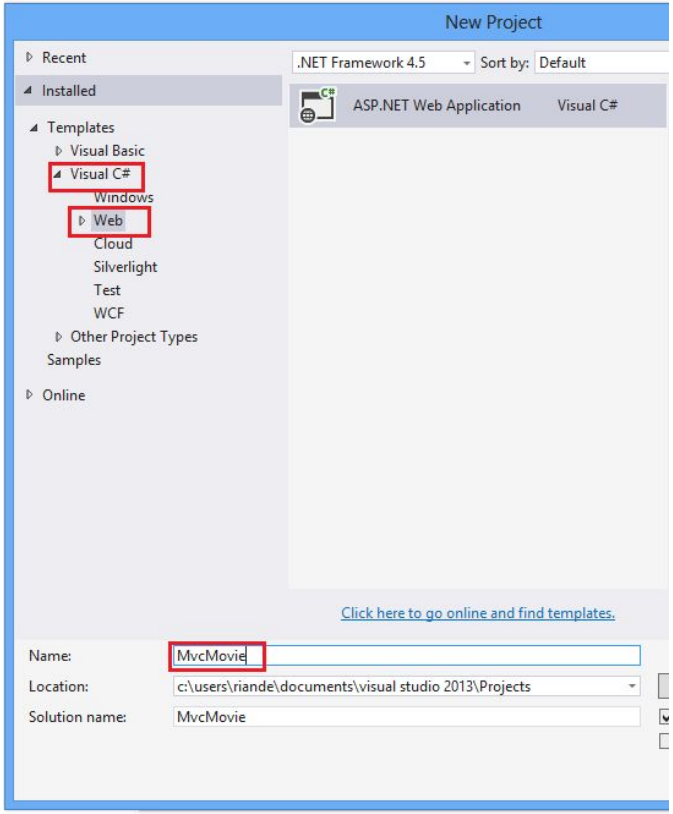
What's new in ASP.NET MVC 5.1 & 5.2

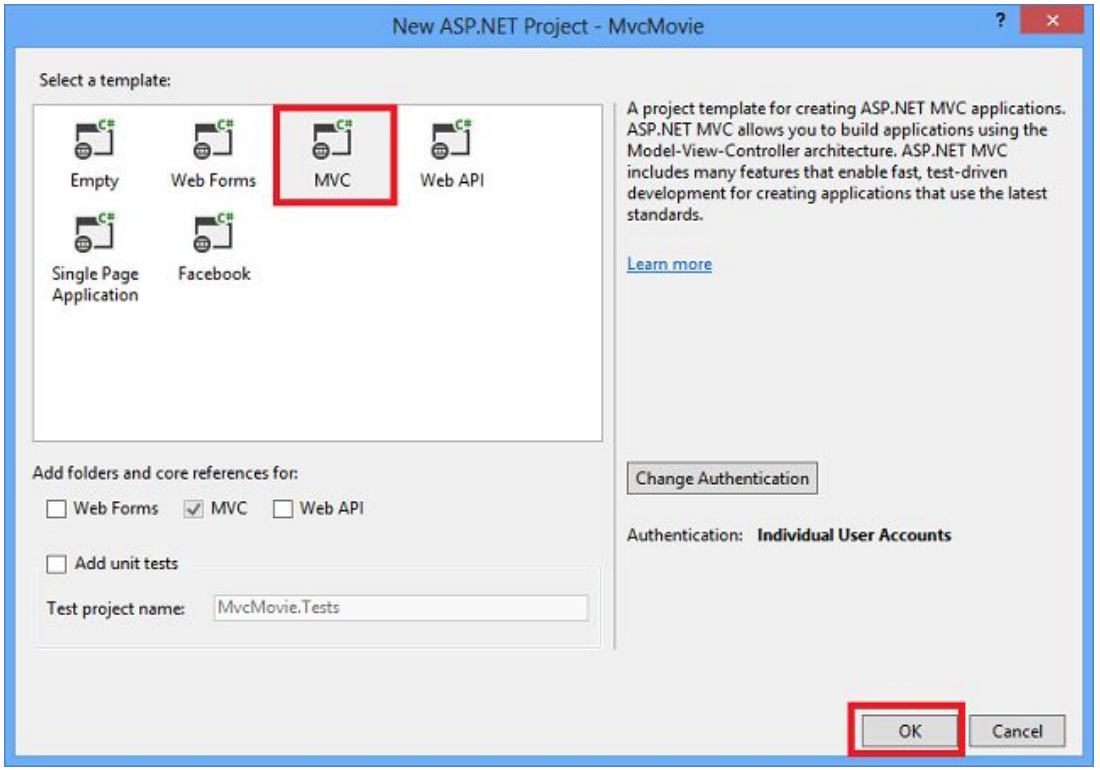
SoftServe

- New Features in ASP.NET MVC 5.1
  - Attribute routing improvements
  - Bootstrap support for editor templates
  - Enum support in views
  - Unobtrusive validation for MinLength/MaxLength Attributes
  - Supporting the 'this' context in Unobtrusive Ajax
- New Features in ASP.NET MVC 5.2
  - Attribute routing improvements

Create ASP.NET MVC 5 Application

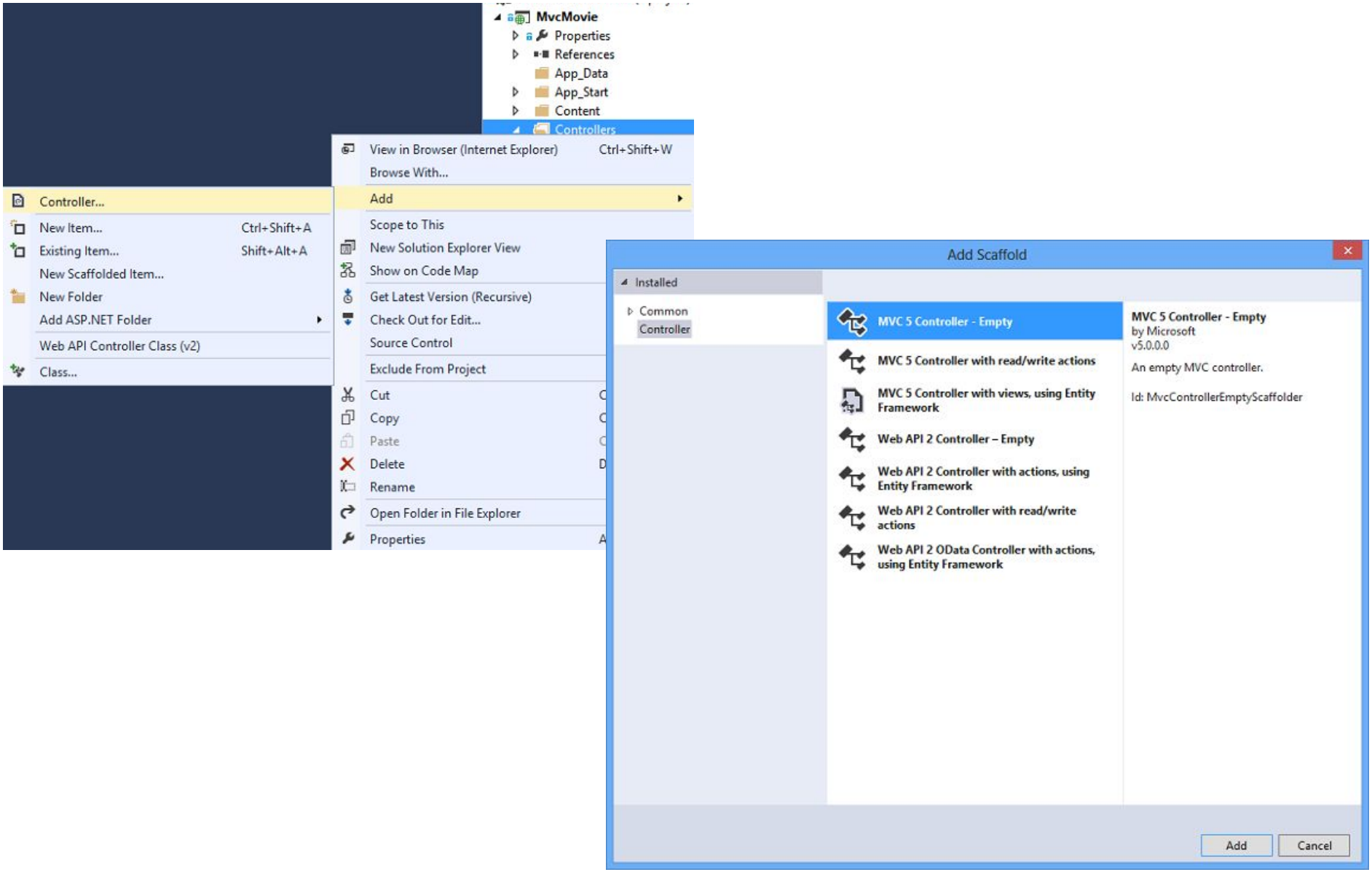
SoftServe





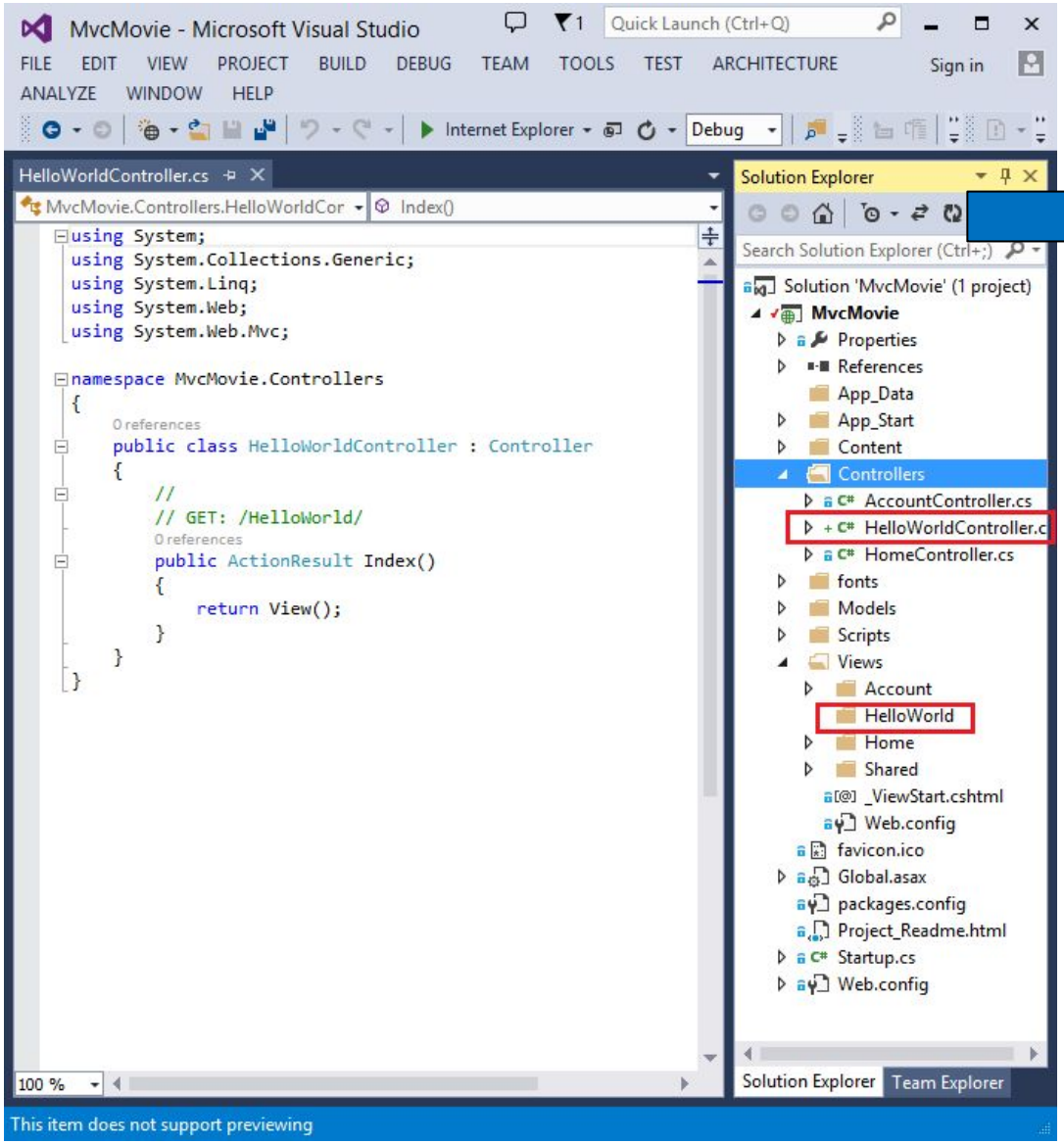
Adding a Controller

SoftServe



Our New HelloWorldController

SoftServe



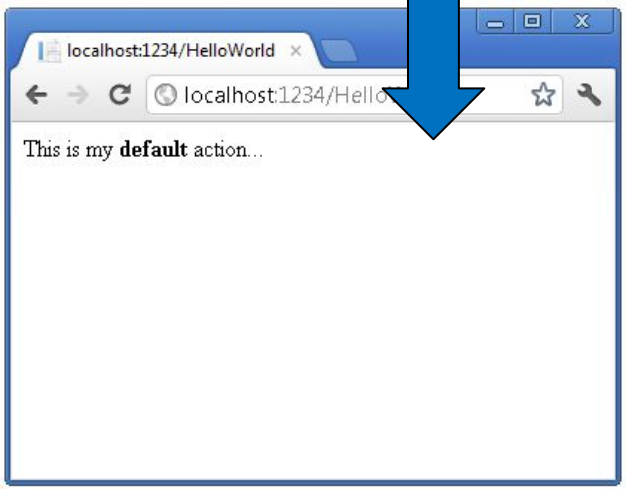

```
using System.Web;
using System.Web.Mvc;

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...";
        }
    }
}
```



The App_Start/RouteConfig.cs File

SoftServe

```
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 }
    );
}
```

Welcome Method with Parameters

SoftServe

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

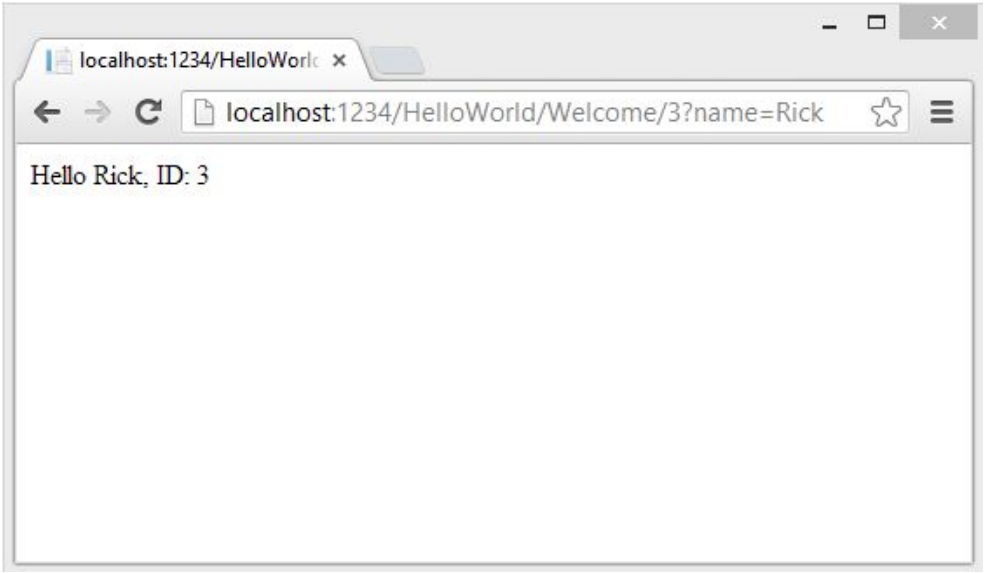
Matching the Route Parameter

SoftServe

```
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 }
    );
}
```

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



Passing Parameters As Route Data

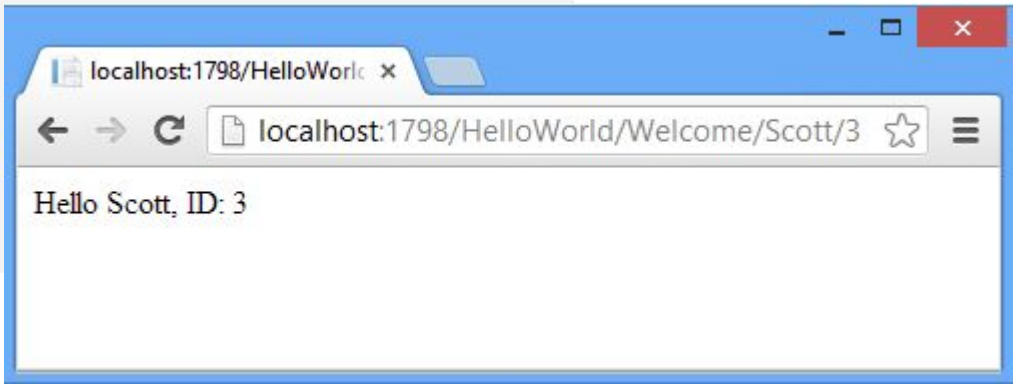
SoftServe

- In ASP.NET MVC applications, it's more typical to pass in parameters as route data than passing them as query strings

```
public class RouteConfig
{
    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 }
        );

        routes.MapRoute(
            name: "Hello",
            url: "{controller}/{action}/{name}/{id}"
        );
    }
}
```



URL Route Mapping Features

SoftServe

- You can include "-", ".", ";" or any other characters you want as part of your route rules
  - This would pass appropriate "language", "locale", and "category" parameters to a ProductsController:

    `{language}-{locale}/products/browse/{category}`

    `/en-us/products/browse/food`

    `language=en, locale=us, category=food`
  - You can use the "." file extension type at the end of a URL to determine whether to render back the result in either a XML or HTML format

    `products/browse/{category}.{format}`

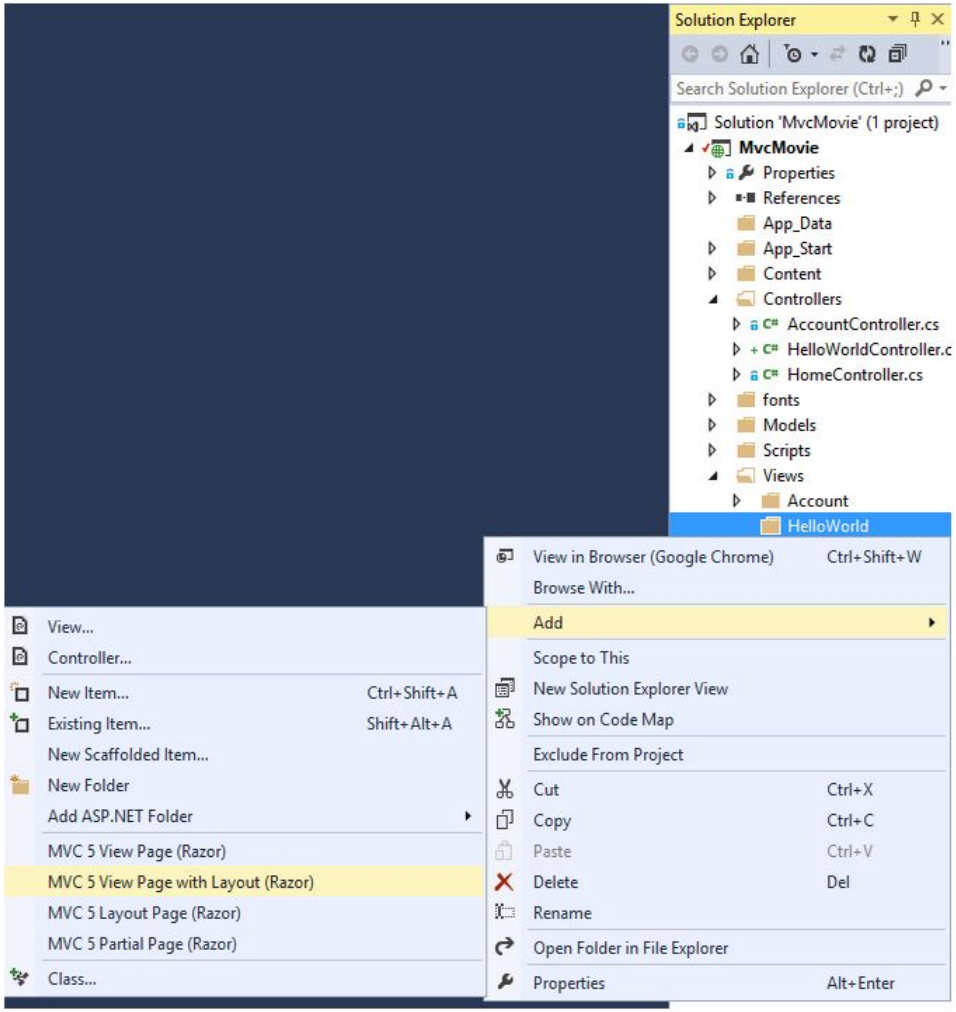
    `/products/browse/food.xml category=food, format=xml`

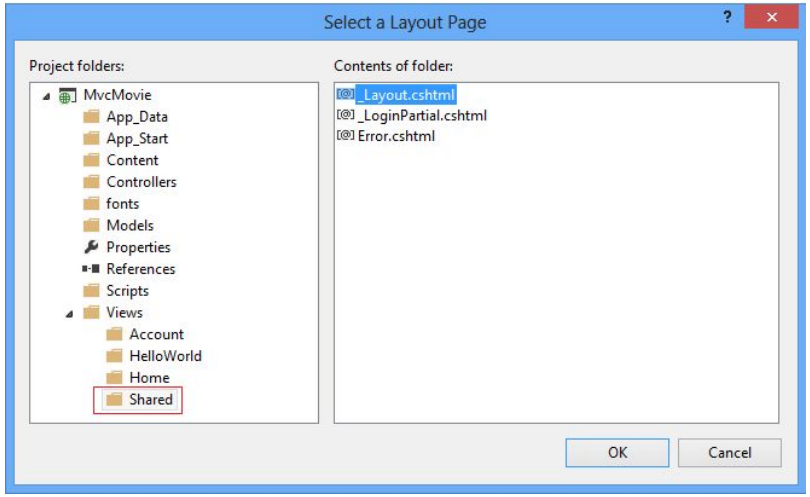
    `/products/browse/food.html category=food, format=html`

Adding a View

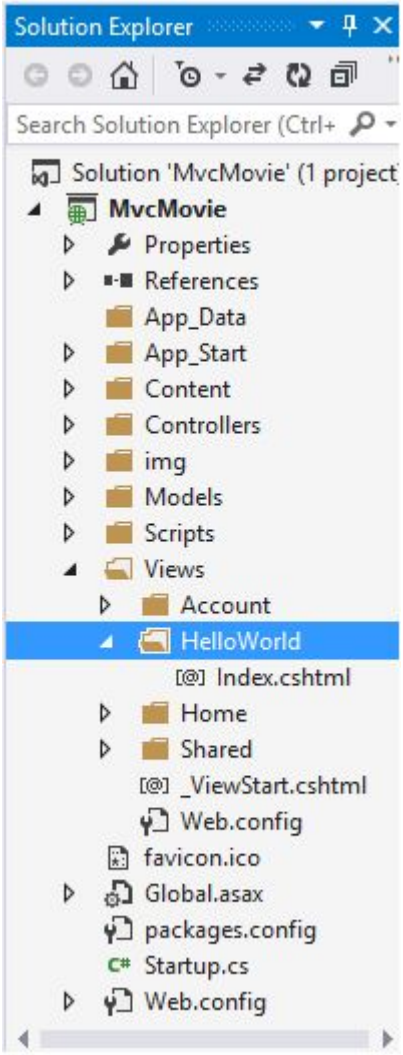
SoftServe

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





The View

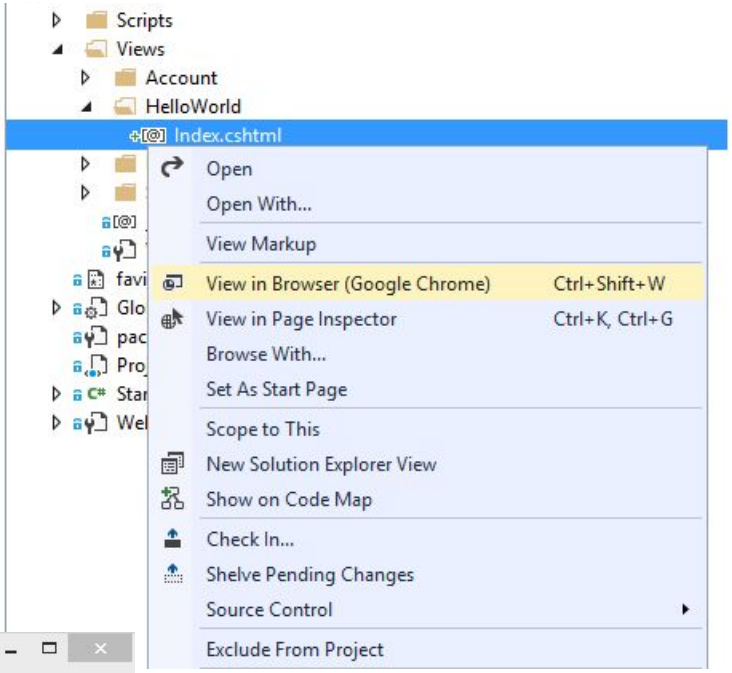


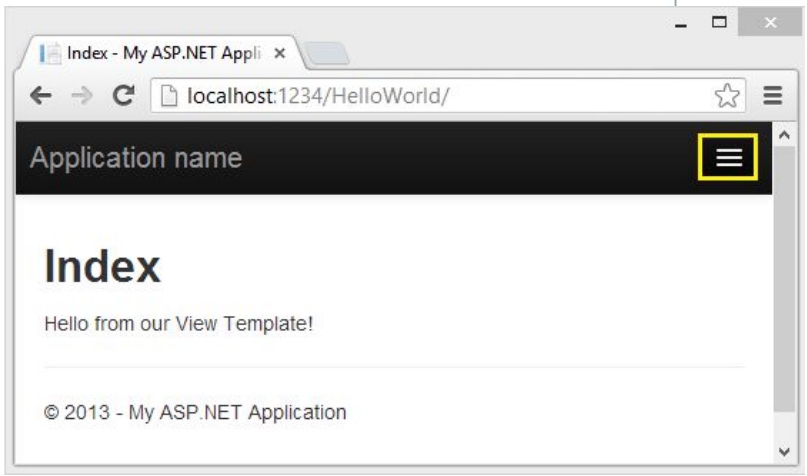

```
@{
    Layout = "~/Views/Shared/_Layout.cshtml";
}

@{
    ViewBag.Title = "Index";
}

<h2>Index</h2>

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





Layout Page

SoftServe

_Layout.cshtml

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

</head>
<body>
    <div class="navbar navbar-inverse navbar-fixed-top"
        <div class="navbar-inner">
            <div class="container">
                <button type="button" class="btn btn-na
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                    <span class="icon-bar"></span>
                </button>
                @Html.ActionLink("Application name", "I
                <div class="nav-collapse collapse">
                    <ul class="nav">
                        <li>@Html.ActionLink("Home", "I
                        <li>@Html.ActionLink("About", "
                        <li>@Html.ActionLink("Contact",
                    </ul>
                    @Html.Partial("_LoginPartial")
```

Solution Explorer

Search Solution Explorer (Ctr

- Solution 'MvcMovie' (1 projec
  - MvcMovie
    - Properties
    - References
    - App_Data
    - App_Start
    - Content
    - Controllers
    - img
    - Models
    - Scripts
    - Views
      - Account
      - HelloWorld
        - Index.cshtml
      - Home
      - Shared
        - _Layout.cshtml
        - _LoginPartial.csh
        - Error.cshtml
    - _ViewStart.cshtml

Layout Page

SoftServe

- The layout has access to the same properties the Razor view has, including:
  - AjaxHelper (through the Ajax property)
  - HtmlHelper (through the Html property)
  - ViewData and model
  - UrlHelper (through the Url property)
  - TempData and ViewContext
- To specify a layout inside a view, we can specify the layout to use with the Layout property:

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

Razor View Engine

SoftServe

- an alternative to the Web Forms view engine
- is responsible for rendering views in the Razor format (either .cshtml files or .vbhtml files)
  - The Web Form view engine is used to support the older-format Web Form views (.aspx and .ascx files)

Web Forms view engine example:

```
<%@ Page Language="C#"
Inherits="System.Web.Mvc.ViewPage<Product[]>" %>
<ul>
<% foreach(var product in Model) { %>
    <li><%: product.Name %></li>
<% } %>
</ul>
```

Razor view engine example

```
@model Product[]
<ul>
@foreach(var product in Model) {
    <li>@product.Name</li>
}
</ul>
```

- '@' is the magic character that precedes code instructions in the following contexts
  - '@' For a single code line/values

```
<p>
   Current time is: @DateTime.Now
</p>
```

  - '@{ … }' For code blocks with multiple lines

```
@{
     var name = "John";
     var nameMessage = "Hello, my name is " + name + " Smith";
}
```

  - '@:' For single plain text to be rendered in the page

```
@{
     @:The day is: @DateTime.Now.DayOfWeek. It is a <b>great</b> day!
}
```

SoftServe

- HTML markup lines can be included at any part of the code:

```
@if(IsPost){
    <p>Hello, the time is @DateTime.Now and this
    page is a postback!</p>
} else {
    <p>Hello, today is: </p> @DateTime.Now
}
```

- Razor uses code syntax to infer indent:

```
// This won't work in Razor. Content has to be
// wrapped between { }
if( i < 1 ) int myVar=0;
```

Passing Data to the View

SoftServe

- There are three different ways to pass data to a view:
  - by using the ViewDataDictionary,
  - by using the ViewBag,
  - by using strongly typed views.

ViewDataDictionary

- It isn't recommended to use ViewDataDictionary
  - You have to perform type casts whenever you want to retrieve something from the dictionary.

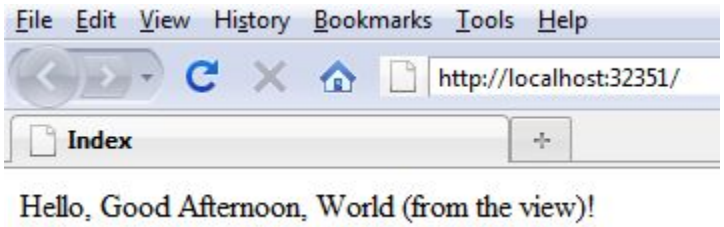
```
public class HomeController : Controller
{
    public ViewResult Index()
    {
        int hour = DateTime.Now.Hour;
        ViewData["greeting"] = (hour < 12 ? "Good Morning" : "Good Afternoon");
        return View();
    }

}
```

```
@{
    Layout = null;
}

<!DOCTYPE html>

<html>
<head>
    <title>Index</title>
</head>
<body>
    <div>
        Hello,
        @ViewData["greeting"], World (from the view)!
    </div>
</body>
</html>
```



ViewBag

- It isn't recommended to use ViewBag
- The ViewBag provides a way to pass data from the controller to the view
  - It makes use of the dynamic language features of C# 4
- Set properties on the dynamic ViewBag property within your controller:
- A ViewBag property is also available in the view:

```
public ActionResult About()
{
    ViewBag.Message = "Your app description page.";

    return View();
}
```

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

<hgroup class="title">
    <h1>@ViewBag.Title.</h1>
    <h2>@ViewBag.Message</h2>
</hgroup>
```

Strongly Typed Views

SoftServe

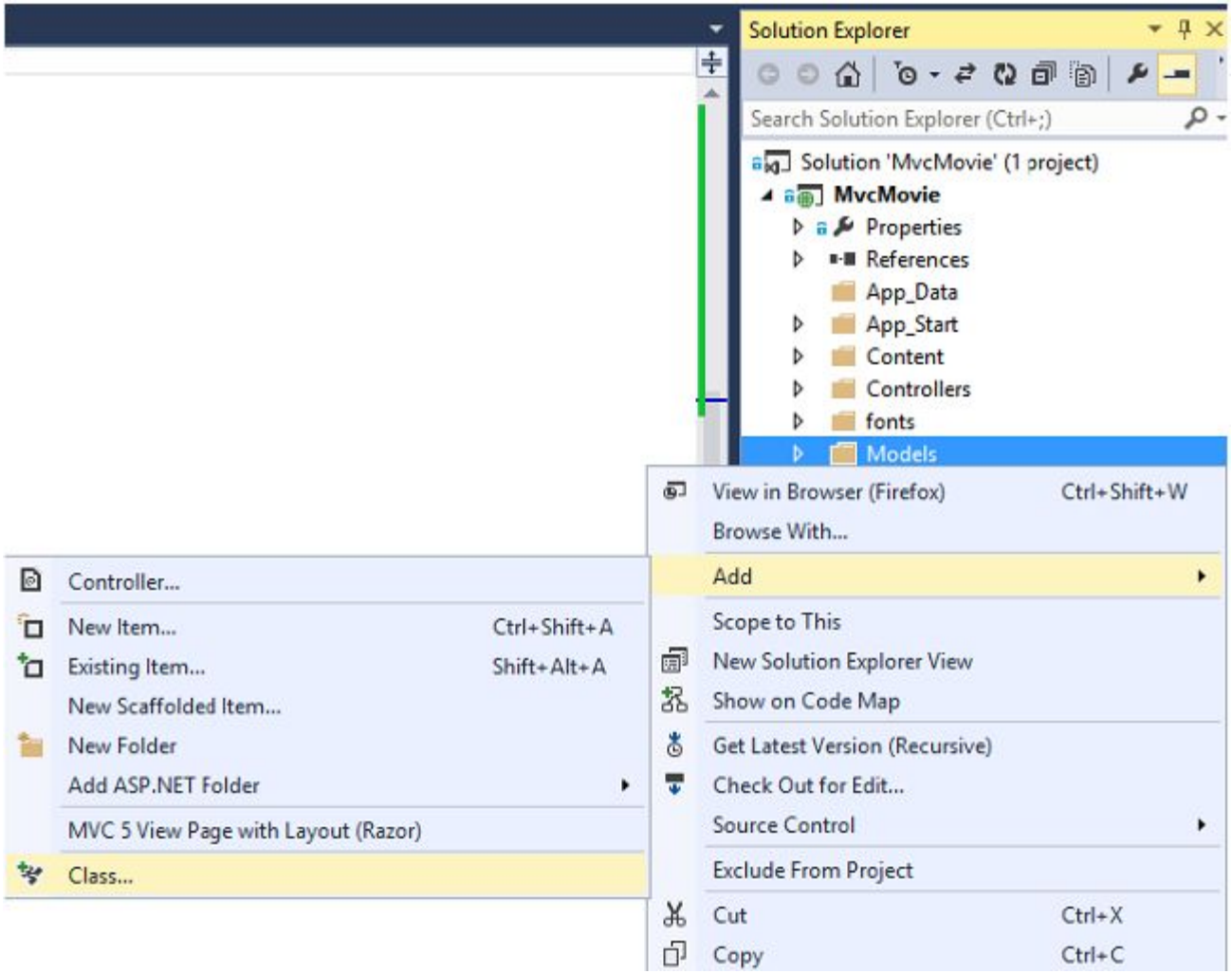
- Views can inherit from two types by default:
  - System.Web.Mvc.WebViewPage or
  - System.Web.Mvc.WebViewPage<T>
- Class WebViewPage<T> provides a strongly typed wrapper over ViewData.Model through the Model property and provides access to strongly typed versions of the associated view helper objects - AjaxHelper and HtmlHelper

Skeleton definition of WebViewPage<T>

```
public class WebViewPage<TModel> : WebViewPage
{
    public new AjaxHelper<TModel> Ajax { get; set; }
    public new HtmlHelper<TModel> Html { get; set; }
    public new TModel Model { get; }
    public new ViewDataDictionary<TModel> ViewData { get; set; }
}
```

1 Strongly typed view model

Adding a Model



Passing Model to the View

SoftServe

- By specifying the model type using the @model keyword, view will inherit from WebViewPage<T> instead of WebViewPage, and we will have a strongly typed view

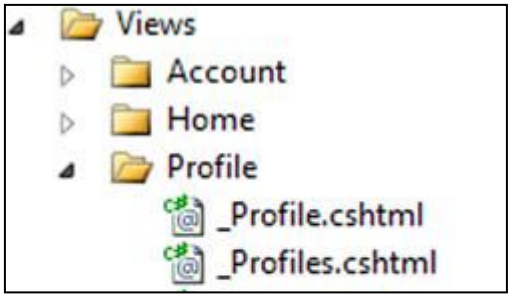
```
public ActionResult Index()
{
     //…
     SomeModel model = new SomeModel();
     return View(model);
}
```

```
<dl>
<dt>Name:</dt>
<dd>@Model.Name</dd>
<dt>Date Added:</dt>
<dd>@Model.DateAdded</dd>
<dt>Message:</dt>
<dd>@Model.Message</dd>
</dl>
```

- Partials are intended to render snippets of content
- If you find yourself copying and pasting one snippet of HTML from one view to the next, that snippet is a great candidate for a partial
- To render a partial we can use the RenderPartial method or the Partial method in a parent view

Rendering a partial from a parent view

```
@model IEnumerable<Profile>
<h2>Profiles</h2>
<table>
    <tr>
        <th>Username</th>
        <th>First name</th>
        <th>Last name</th>
        <th>Email</th>
    </tr>
    @foreach (var profile in Model) {
        @Html.Partial("_Profile", profile)
    }
</table>
```



A partial to display a row for a `Profile` model

```
@model AccountProfile.Models.Profile
<tr>
    <td>@Model.FirstName</td>
    <td>@Model.LastName</td>
    <td>@Model.Email</td>
</tr>
```

Partial Views

SoftServe

- The partial name is used to locate the partial markup in the locations:
  - <Area>\<Controller>\<PartialName>.cshtml
  - <Area>\Shared\<PartialName>.cshtml
  - \<Controller>\<PartialName>.cshtml
  - \Shared\<PartialName>.cshtml
- In order to prevent accidentally using a partial view from an action, we prefix the view name with an underscore
- Html.RenderPartial(...) renders the partial immediately to the response stream
- Html.Partial(...) returns a string
  - In Razor, Html.RenderPartial must be in a code block

Questions ?

SoftServe

?