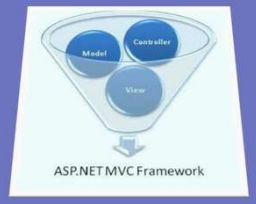


Microsoft ASP.net MVC

ASP.NET MVC Essentials

ASP.NET MVC Routing

Table of Contents

2

ASP.NET MVC Routing

- Mapping between patterns and a combination of controller + action + parameters
- Routes are defined as a global list of routes
  - System.Web.Routing.RouteTable.Routes
- Something similar to Apache mod_rewrite
- Greedy algorithm
  - the first match wins

4

Register routes

- In Global.asax in the Application_Start() there is RouteConfig.RegisterRoutes(RouteTable.Routes);
- RoutesConfig class is located in /App_Start/ in internet applications template by default

```
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 to ignore
The [*] means all left

Route name

Route pattern

Default parameters

Routing Examples

```
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
        }
    );
}
```

http://localhost/Products/ById/3

- Controller: Products
- Action: ById
- Id: 3

Routing Examples (2)

```
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
        }
    );
}
```

http://localhost/Products/ById

- Controller: Products
- Action: ById
- Id: 0 (optional parameter)

Routing Examples (3)

```
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
        }
    );
}
```

http://localhost/Products

- Controller: Products
- Action: Index
- Id: 0 (optional parameter)

Routing Examples (4)

```
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
        }
    );
}
```

http://localhost/

- Controller: Home
- Action: Index
- Id: o (optional parameter)

9

Custom Route

```
routes.MapRoute(
    name: "Users",
    url: "Users/{username}",
    defaults: new
    {
        controller = "Users",
        action = "ByUsername",
        username = "DefaultValue",
    }
);
```

http://localhost/Users/NikolayIT

- Controller: Users
- Action: ByUsername
- Username: NikolayIT

10

Custom Route (2)

```
routes.MapRoute(
    name: "Users",
    url: "Users/{username}",
    defaults: new
    {
        controller = "Users",
        action = "ByUsername",
        username = "DefaultValue",
    }
);
```

http://localhost/Users

- Controller: Users
- Action: ByUsername
- Username: DefaultValue

11

Custom Route (3)

```
routes.MapRoute(
    name: "Users",
    url: "Users/{username}",
    defaults: new
    {
        controller = "Users",
        action = "ByUsername",
    }
);
```

?

http://localhost/Users

- Result: 404 Not Found

12

Route Constraints

- Constraints are rules on the URL segments
- All the constraints are regular expression compatible with class Regex
- Defined as one of the routes.MapRoute(…) parameters

```
// 2013/01/29/Blog-title
routes.MapRoute(
    name: "Blog",
    url: "{year}/{month}/{day}",
    defaults: new { controller = "Blog", action = "ByDate" },
    constraints: new { year=@"\d{4}", month=@"\d{2}", day=@"\d{2}" }
);
```

13

Attribute Route

- You can also combine attribute routing with convention-based routing.

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

    routes.MapMvcAttributeRoutes();

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

15

Attribute Route

Enabling Attribute Routing

To enable attribute routing, call MapMvcAttributeRoutes during configuration.

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

        routes.MapMvcAttributeRoutes();
    }
}
```

14

Attribute Route

Optional URI Parameters and Default Values

- You can make a URI parameter optional by adding a question mark to the route parameter. You can also specify a default value by using the form parameter=value.

```
[Route("books/{isbn?}")]
public ActionResult View(string isbn)
,
[Route("books/lang/{lang=en}")]
public ActionResult ViewByLanguage(string lang)
{
    return View("OneBook", GetBooksByLanguage(lang));
}
    return View("AllBooks", GetBooks());
}
```

16

Custom Route Constraint

```
public class LocalhostConstraint : IRouteConstraint
{
    public bool Match
        (
            HttpContextBase httpContext,
            Route route,
            string parameterName,
            RouteValueDictionary values,
            RouteDirection routeDirection
        )
    {
        return httpContext.Request.IsLocal;
    }
}
```

```
public static void RegisterRoutes(RouteCollection routes)
{
    routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
    routes.MapRoute(
        "Admin",
        "Admin/{action}",
        new {controller="Admin"},
        new {isLocal=new LocalhostConstraint()}
    );
```

Demo: Routes

ASP.NET MVC Routing

Debugging Routes

- In actions we have access to a data structure called RouteData
  - RouteData.Values["controller"]
  - RouteData.Values["action"]
  - RouteData.Values["id"]
- We can use NuGet package RouteDebugger
  - Install-Package RouteDebugger
    - Web.config: <add key="RouteDebugger:Enabled" value="true" />
- We can also use Glimpse for debugging routes

18

Controllers and Actions

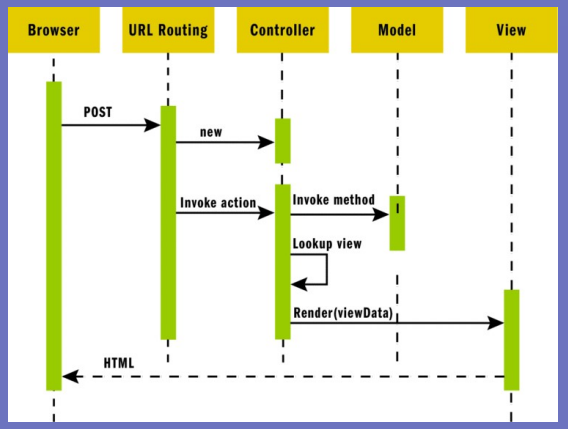
The brain of the application

Controllers

- The core component of the MVC pattern
- All the controllers should be available in a folder by name Controllers
- Controller naming standard should be "nameController" (convention)
- Routers instantiate controllers in every request
  - All requests are mapped to a specific action
- Every controller should inherit Controller class
  - Access to Request (context) and HttpContext

21

ASP.NET MVC Request



23

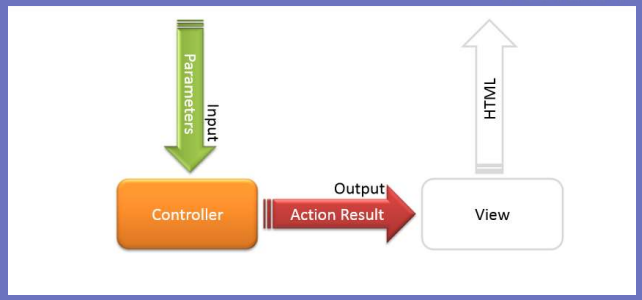
Actions

- Actions are the ultimate request destination
  - Public controller methods
  - Non-static
  - No return value restrictions
- Actions typically return an ActionResult

```
public ActionResult Contact()
{
    ViewBag.Message = "Your contact page.";
    return View();
}
```

22

Action IO



- **Input:** parameters? **get request parameters?**
- **Output: select View**

24

Action Parameters

- ASP.NET MVC maps the data from the HTTP request to action parameters in few ways:
  - Routing engine can pass parameters to actions
    - http://localhost/Users/NikolayIT
    - Routing pattern: Users/{username}
  - URL query string can contain parameters
    - /Users/ByUsername?username=NikolayIT
  - HTTP post data can also contain parameters

```
public ActionResult ByUsername(string username)
{
    return Content(username);
}
```

25

Action Selectors

- ActionName(string name)
- AcceptVerbs
  - HttpPost
  - HttpGet
  - HttpDelete
  - HttpOptions
  - …
- NonAction
- RequireHttps
- ChildActionOnly – Only for Html.Action()

```
public class UsersController : Controller
{
    [ActionName("UserLogin")]
    [HttpPost]
    [RequireHttps]
    public ActionResult Login(string pass)
    {
        return Content(pass);
    }
}
```

27

Action Parameters

Query String

```
<a href="/Student/Register?Id=SV01&Name=Tuấn&Marks=7">Tuấn</a>
<a href="/Student/Register/SV02?Name=Phương&Marks=8">Phương</a>
```

Form field

```
<form action="/Student/Register" method="post">
    <div>Id</div> <input name="Id" />
    <div>Name</div> <input name="Name" />
    <div>Marks</div> <input name="Marks" />
    <hr />
    <input type="submit" value="Register" />
</form>
```

26

Action Selectors

- Action can be invoked in both POST and GET

```
public ActionResult MyAction()
```

- use POST or GET, Action have to be mark with [HttpPost] or [HttpGet]

```
[HttpGet]
public ActionResult MyAction()

[HttpPost]
public ActionResult MyAction(MyModel model)
```

28

Action Selectors

- You can also apply multiple http verbs using AcceptVerbs attribute. GetAndPostAction method supports both, GET and POST ActionVerbs in the following example:

```
[AcceptVerbs(HttpVerbs.Post | HttpVerbs.Get)]
public ActionResult GetAndPostAction()
{
    return RedirectToAction("Index");
}
```

29

Action Selectors

- Rename transaction name of Action :

```
[ActionName("OtherName")]
public ActionResult MyAction()
```

- use @ Html.Action (), not allow call direct

```
[ChildActionOnly]
public ActionResult MyAction()
```

30

Get request parameters

In MVC has 4 ways to get request parameters

- Request object
- Argument of Action
- FormCollection
- Model

31

Get request parameters

- Request object: syntax:

```
String value = Request ["<parameter>"];
String value1 = Request.QueryString["<parameter>"];
String value2 = Request.Form["<parameter>"];
String value3 = Request.Params["<parameter>"];
```

- Example:

```
string Id = Request["Id"];
string Name = Request["Name"];
double Marks = Convert.ToDouble(Request["Marks"]);
```

32

Get request parameters

- Argument of Action:

```
public ActionResult UseArgument
        (string Id, string Name, double Marks=0){…}
```

```
<form action="/Student/Register" method="post">
    <input name="Id" />
    <input name="Name" />
    <input name="Marks" />
    <input type="submit" value="Register" />
</form>
```

33

Get request parameters

- FormCollection:

```
public ActionResult UseFormCollection(FormCollection Fields)
{
    string Id = Fields["Id"];
    string Name = Fields["Name"];
    double Marks = Convert.ToDouble(Fields["Marks"]);
    return View();
}
```

34

Get request parameters

- Model:

```
public class StudentInfo
{
    public string Id { get; set; }
    public string Name { get; set; }
    public double Marks { get; set; }
}
```

```
public ActionResult UseModel(StudentInfo model){…}
```

35

Action Results

- Controller action response to a browser request
- Inherits from the base ActionResult class
- Different results types:

Name	Framework Behavior	Producing Method
ContentResult	Returns a string literal	Content
EmptyResult	No response	
FileContentResult FilePathResult FileStreamResult	Return the contents of a file	File

36

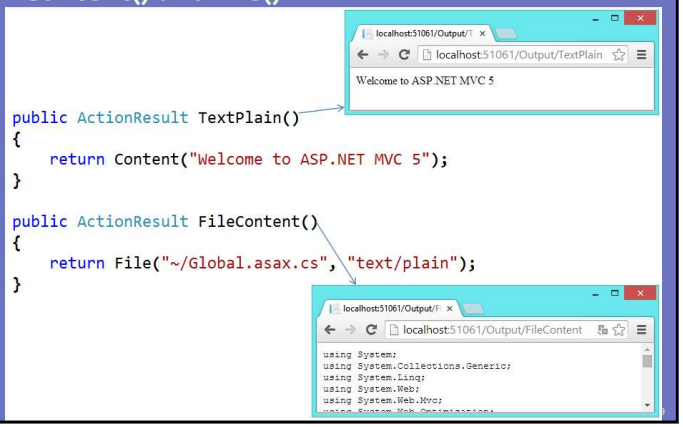
Action Results (2)

Name	Framework Behavior	Producing Method
HttpUnauthorizedResult	Returns an HTTP 403 status	
JavaScriptResult	Returns a script to execute	JavaScript
JsonResult	Returns data in JSON format	Json
RedirectResult	Redirects the client to a new URL	Redirect / RedirectPermanent
RedirectToRouteResult	Redirect to another action, or another controller's action	RedirectToRoute / RedirectToAction
ViewResult PartialViewResult	Response is the responsibility of a view engine	View / PartialView

37

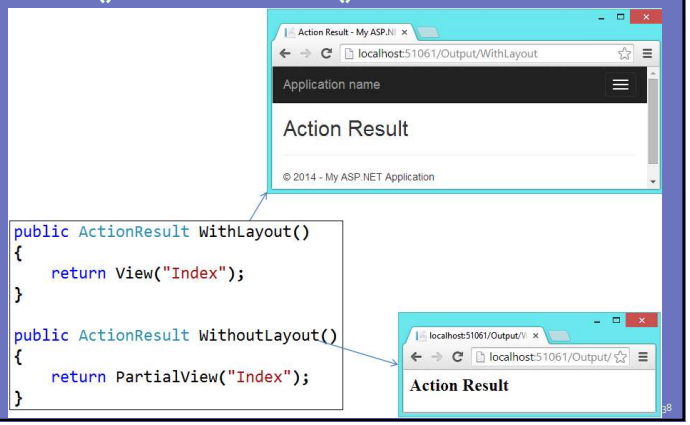
Action Results Example

- Content() and File()



Action Results Example

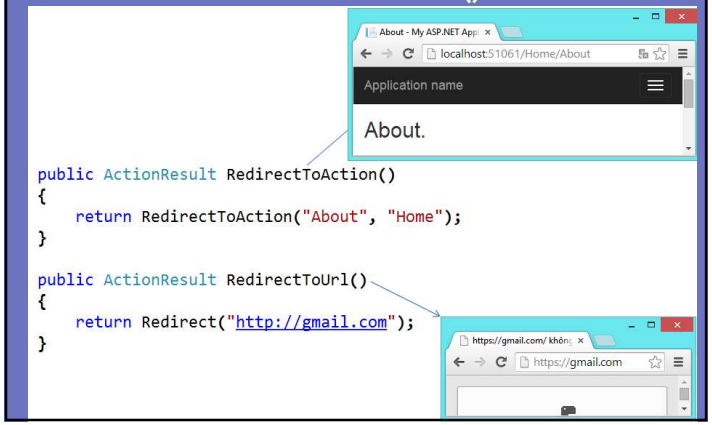
- View() and PartialView()



38

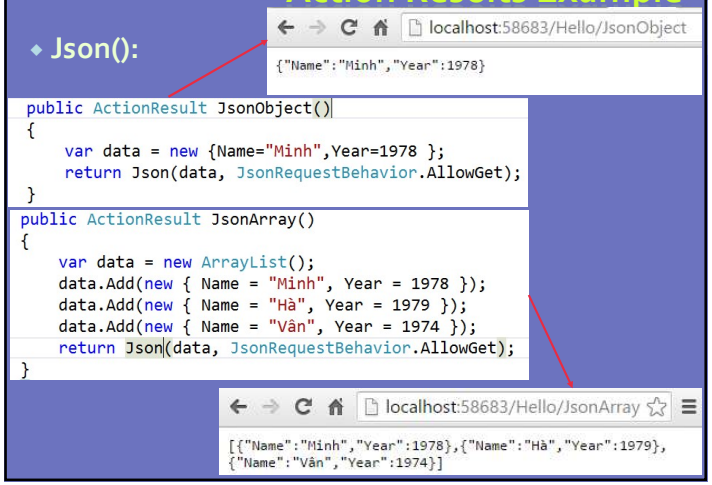
Action Results Example

- RedirectToAction and Redirect():



Action Results Example

- Json():

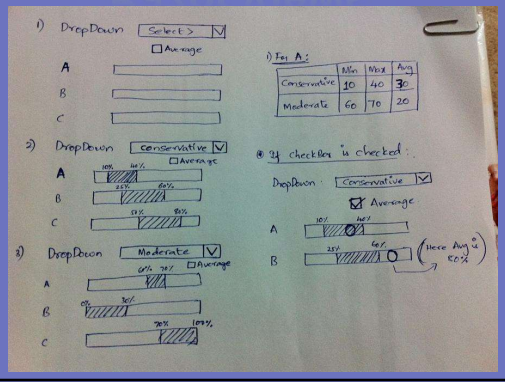


Views

- HTML templates of the application
- A lot of view engines available
  - View engines execute code and provide HTML
  - Provide a lot of helpers to easily generate HTML
  - The most popular is Razor and WebForms
- We can pass data to views through ViewBag, ViewData and Model (strongly-typed views)
- Views support master pages (layout views)
- Other views can be rendered (partial views)

43

Razor Views



Razor

- Template markup syntax
- Simple-syntax view engine

- Based on the C# programming language
- Enables the programmer to use an HTML construction workflow
- Code-focused templating approach, with minimal transition between HTML and code
  - Razor syntax starts code blocks with a @ character and does not require explicit closing of the code-block

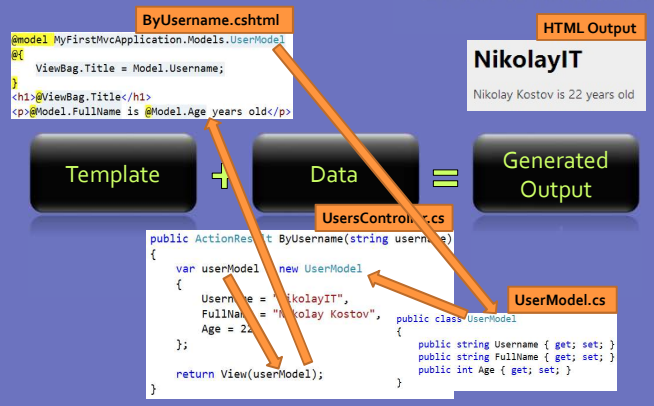
44

Design Goals

- Compact, Expressive, and Fluid
  - Smart enough to differ HTML from code
- Easy to Learn
- Is not a new language
- Works with any Text Editor
- Has great Intellisense
  - Built in Visual Studio
- Unit Testable
  - Without requiring a controller or web-server

45

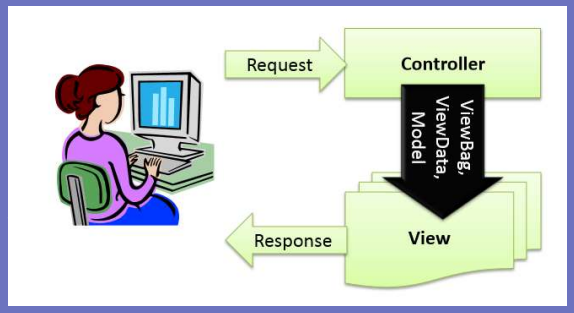
How it works?



47

Pass Data to a View

- With ViewBag (dynamic type):
  - Action: ViewBag.Message = "Hello World!";
- With ViewData (dictionary type):
  - Action: ViewData["message"] = "Hello World!";
- With Model (strongly typed):
  - Action: return View(Student);
- View: @ViewData["message"]



46

Views

- ViewBag , ViewData example:

```
public ActionResult Detail()
{
    ViewBag.Id = "SV001";
    ViewBag.Name = "Nguyễn Anh Tuấn";
    ViewData["Marks"] = 9.5;
    return View();
}
```

ViewBag.Id ~ ViewData["Id"]

```
<h2>Student Detail</h2>
<ul>
    <li>Id: @ViewBag.Id</li>
    <li>Name: @ViewData["Name"]</li>
    <li>Marks: @ViewBag.Marks</li>
</ul>
```

48

Views

- Model example:

```
public ActionResult Detail()
{
    // Tạo đối tượng
    var model = new StudentInfo
    {
        Id = "SV001",
        Name = "Nguyễn Anh Tuấn",
        Marks = 9.5
    };
    // Truyền đối tượng model cho view
    return View(model);
}
```

Action

```
public class StudentInfo
{
    public string Id { get; set; }
    public string Name { get; set; }
    public double Marks { get; set; }
}
```

Model

```
@model Mvc5.Models.StudentInfo

<h2>Student Detail</h2>
<ul>
    <li>Id: @Model.Id</li>
    <li>Name: @Model.Name</li>
    <li>Marks: @Model.Marks</li>
</ul>
```

View

Student Detail

- Id: SV001
- Name: Nguyễn Anh Tuấn
- Marks: 9.5

Razor Syntax

- @ – For values (HTML encoded)

```
<p>
   Current time is: @DateTime.Now!!!
   Not HTML encoded value: @Html.Raw(someVar)
</p>
```

- @{ … } – For code blocks (keep the view simple!)

```
@{
   var productName = "Energy drink";
   if (Model != null)
   {
      productName = Model.ProductName;
   }
   else if (ViewBag.ProductName != null)
   {
      productName = ViewBag.ProductName;
   }
}
<p>Product "@productName" has been added in your shopping cart</p>
```

51

Views

```
public ActionResult Browse()
{
    ViewBag.Title = "List of your mails";

    Mail mail1 = new Mail
    {
        From = "receiver1@gmail.com",
        To = "sender1@gmail.com",
        Subject = "Mail subject 1",
        Body = "Mail content 1"
    };
    Mail mail2 = new Mail
    {
        From = "receiver2@gmail.com",
        To = "sender2@gmail.com",
        Subject = "Mail subject 2",
        Body = "Mail content 2"
    };

    List<Mail> mails = new List<Mail>();
    mails.Add(mail1);
    mails.Add(mail2);

    return View(mails);
}
```

```
@model IEnumerable<BasicDemo.Models.Mail>

<h2>@ViewBag.Title</h2>

<table border="1" style="width:100%">
    <tr style="background:yellow;">
        <th>FROM</th>
        <th>TO</th>
        <th>SUBJECT</th>
        <th>BODY</th>
        <th> </th>
    </tr>
@foreach (var item in Model) {
    <tr>
        <td>@item.From</td>
        <td>@item.To</td>
        <td>@item.Subject</td>
        <td>@item.Body</td>
        <td><a href="Home/Single">Single</a></td>
    </tr>
}
</table>
```

50

Razor Syntax (2)

- If, else, for, foreach, etc. C# statements
  - HTML markup lines can be included at any part
  - @: – For plain text line to be rendered

```
<div class="products-list">
@if (Model.Products.Count() == 0)
{
   <p>Sorry, no products found!</p>
}
else
{
   @:List of the products found:
   foreach(var product in Model.Products)
   {
      <b>@product.Name, </b>
   }
}
</div>
```

52

Razor Syntax (3)

- Comments

```
@*
  A Razor Comment
*@
@{
  //A C# comment

  /* A Multi
     line C# comment
  */
}
```

- What about "@" and emails?

```
<p>
  This is the sign that separates email names from domains: @@<br />
  And this is how smart Razor is: spam_me@gmail.com
</p>
```

53

Razor Syntax (4)

- @(…) – Explicit code expression

```
<p>
  Current rating(0-10): @Model.Rating / 10.0      @* 6 / 10.0 *@
  Current rating(0-1): @(Model.Rating / 10.0)     @* 0.6 *@
  spam_me@Model.Rating                            @* spam_me@Model.Rating *@
  spam_me@(Model.Rating)                          @* spam_me6 *@
</p>
```

- @using – for including namespace into view
- @model – for defining the model for the view

```
@using MyFirstMvcApplication.Models;
@model UserModel
<p>@Model.Username</p>
```

54

View Helpers

- Each view inherits WebViewPage
  - ViewPage has a property named Html
- Html property has methods that return string and can be used to generate HTML
  - Create inputs
  - Create links
  - Create forms

```
@using (Html.BeginForm("Search", "Users",
                       FormMethod.Post))
{
    @Html.TextBox("username")
    <input type="submit" />
}
@Html.Raw(htmlContent)
```

- Other helper properties are also available
  - Ajax, Url, custom helpers

55

HTML Helpers

Method	Type	Description
BeginForm, BeginRouteForm	Form	Returns an internal object that represents an HTML form that the system uses to render the *<form>* tag
EndForm	Form	A void method, closes the pending *</form>* tag
CheckBox, CheckBoxFor	Input	Returns the HTML string for a check box input element
Hidden, HiddenFor	Input	Returns the HTML string for a hidden input element
Password, PasswordFor	Input	Returns the HTML string for a password input element
RadioButton, RadioButtonFor	Input	Returns the HTML string for a radio button input element
TextBox, TextBoxFor	Input	Returns the HTML string for a text input element
Label, LabelFor	Label	Returns the HTML string for an HTML label element

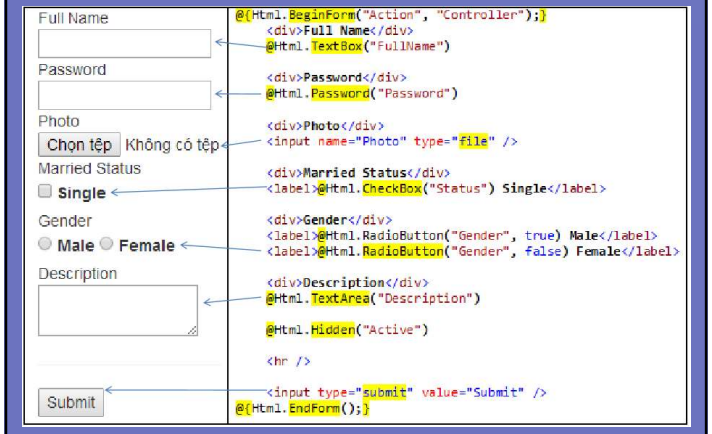
56

HTML Helpers (2)

Method	Type	Description
ActionLink, RouteLink	Link	Returns the HTML string for an HTML link
DropDownList, DropDownListFor	List	Returns the HTML string for a drop-down list
ListBox, ListBoxFor	List	Returns the HTML string for a list box
TextArea, TextAreaFor	TextArea	Returns the HTML string for a text area
Partial	Partial	Returns the HTML string incorporated in the specified user control
RenderPartial	Partial	Writes the HTML string incorporated in the specified user control to the output stream
ValidationMessage, ValidationMessageFor	Validation	Returns the HTML string for a validation message
ValidationSummary	Validation	Returns the HTML string for a validation summary message

57

Form Helper Example



59

@Html.ActionLink()

- Parameters:
  - linkText
  - actionName
  - routeValues
  - controllerName
  - htmlAttributes

```
@Html.ActionLink("Giới thiệu", "About" )
<a href="/Home/About">Giới thiệu</a>
```

```
@Html.ActionLink("Edit Record", "Edit", new {Id=3})
<a href="/Store/Edit/3">Edit Record</a>
```

Image link:

```
<a href="@Url.Action("Delete")">
    <img src="/Content/Delete.png"/></a>
```

58

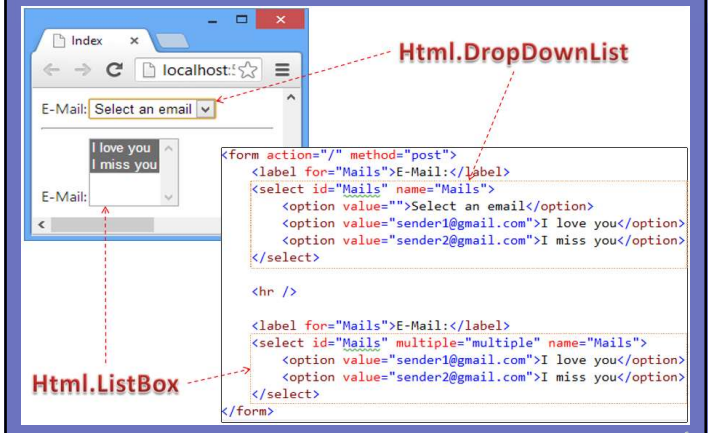
DropdownList & ListBox

```
List<Mail> Mails = new List<Mail>{
    new Mail {
        To = "sender1@gmail.com",
        Subject = "I love you"
    },
    new Mail {
        To = "sender2@gmail.com",
        Subject = "I miss you"
    }
};
ViewBag.Mails = new SelectList(Mails, "To", "Subject");
```

```
@using (Html.BeginForm()){
    @Html.Label("Mails", "E-Mail:");
    @Html.DropDownList("Mails", "Select an email")
    <hr />
    @Html.Label("Mails", "E-Mail:");
    @Html.ListBox("Mails")
}
```

60

Code HTML for DropdownList & ListBox




```
<form action="/" method="post">
    <label for="Mails">E-Mail:</label>
    <select id="Mails" name="Mails">
        <option value="">Select an email</option>
        <option value="sender1@gmail.com">I love you</option>
        <option value="sender2@gmail.com">I miss you</option>
    </select>

    <hr />

    <label for="Mails">E-Mail:</label>
    <select id="Mails" multiple="multiple" name="Mails">
        <option value="sender1@gmail.com">I love you</option>
        <option value="sender2@gmail.com">I miss you</option>
    </select>
</form>
```

61

Code HTML for Form

```
@using (Html.BeginForm("Register", "Member")) {
    ... nội dung form ...
}
```

```
<form action="/Member/Register" method="post">
    ... nội dung ...
</form>
```

62

Format Helper

Helper
@Html.FormatValue (value, format)
@String.Format(format, value1, value2...)
@Html.Raw (html)

- Format for Number, DateTime
- Format for String, ...
- decode HTML for encoded string

63

Format Helper

```
@{
    var number1 = 12345.8765;
    var number2 = 0.72;
}
<ul>
    <li>Số bình thường: @number1</li>
    <li>Phân nhóm: @Html.FormatValue(number1, "{0:#,###.##0}")</li>
    <li>Tiền tệ: @Html.FormatValue(number1, "{0:c}")</li>
    <li>Phần trăm: @Html.FormatValue(number2, "{0:p}")</li>
</ul>
```

- Số bình thường: 12345.8765
- Phân nhóm: 12,345.877
- Tiền tệ: $12,345.88
- Phần trăm: 72.00 %

64

Format Helper

{0:D}	Date – theo ngôn ngữ được chọn
{0:MMMM-dd-yyyy hh:mm:ss tt}	✓ M,MM,MMM,MMMM: tháng 1, 2 ký tự số, 3 ký tự viết tắt, tên tháng đầy đủ ✓ d,dd: ngày 1, 2 ký tự ✓ yy,yyyy: năm 2, 4 ký tự số ✓ H, HH, h, hh: 1,2 ký tự giờ 24 hoặc 12 giờ mỗi ngày ✓ m,mm: 1,2 ký tự số phút ✓ s,ss: 1,2 ký tự số giây ✓ tt: 2 ký tự sáng/chiều

- Ngày bình thường: 5/27/2014 9:26:09 PM
- Định dạng D: Tuesday, May 27, 2014
- Định dạng ISO: 2014-05-27
- Định dạng English: 05/27/2014
- Định dạng 24 giờ: 21:26:09
- Định dạng 12 giờ: 09:26:09 PM

```
@{
    var now = DateTime.Now;
}
<ul>
    <li>Ngày bình thường: @now</li>
    <li>Định dạng D: @Html.FormatValue(now, "{0:D}")</li>
    <li>Định dạng ISO: @Html.FormatValue(now, "{0:yyyy-MM-dd}")</li>
    <li>Định dạng English: @Html.FormatValue(now, "{0:MM/dd/yyyy}")</li>
    <li>Định dạng 24 giờ: @Html.FormatValue(now, "{0:HH:mm:ss}")</li>
    <li>Định dạng 12 giờ: @Html.FormatValue(now, "{0:hh:mm:ss tt}")</li>
</ul>
```

65

View with Model

- Attribute

```
public class Student
{
    [DisplayName("Mã sinh viên")]
    public String Id { get; set; }
    [DisplayName("Mật khẩu")]
    public String Password { get; set; }
    [DisplayName("Họ và tên")]
    public String FullName { get; set; }
    [DisplayName("Giới tính")]
    public bool Gender { get; set; }
    [DisplayName("Ngày sinh")]
    public DateTime Birthday { get; set; }
    [DisplayName("Ghi chú")]
    public String Notes { get; set; }
}
```

67

Format Helper

- Có mã hóa HTML: Hello
- Không mã hóa HTML : **Hello**

```
@{
    var chuoi = "<strong>Hello</strong>";
}
<ul>
    <li>Có mã hóa HTML: @chuoi</li>
    <li>Không mã hóa HTML : @Html.Raw(chuoi)</li>
</ul>
```

66

View with Model

- Explicit UI

```
@model Mvc5CodeDemo.Models.Student
<h2>Đăng ký thành viên</h2>
@using (Html.BeginForm())
{
  <table><tr>
      <td>@Html.LabelFor(m => m.Id)</td>
      <td>@Html.TextBoxFor(m => m.Id)</td>
  </tr><tr>
      <td>@Html.LabelFor(m => m.Password)</td>
      <td>@Html.PasswordFor(m => m.Password)</td>
  </tr><tr>
      <td>@Html.LabelFor(m => m.FullName)</td>
      <td>@Html.TextBoxFor(m => m.FullName)</td>
  </tr><tr>
      <td>@Html.LabelFor(m => m.Gender)</td>
      <td>
          <label>@Html.RadioButtonFor(m => m.Gender, true) Nam</label>
          <label>@Html.RadioButtonFor(m => m.Gender, false) Nữ</label>
      </td>
  </tr><tr>
      <td>@Html.LabelFor(m => m.Birthday)</td>
      <td>@Html.TextBoxFor(m => m.Birthday)</td>
  </tr><tr>
      <td>@Html.LabelFor(m => m.Notes)</td>
      <td>@Html.TextAreaFor(m => m.Notes)</td>
  </tr><tr>
      <td> </td>
      <td><input type="submit" value="Register" /></td>
  </tr></table>
}
```

Sinh <label for="Id">Mã sinh viên</label>

Kiểu của Model

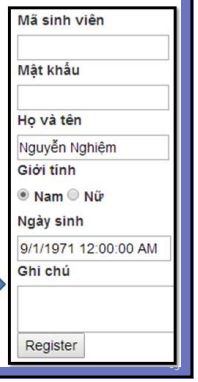
Sinh <input type="text" name="Id" id="Id"> từ thuộc tính Id của Model

View with Model

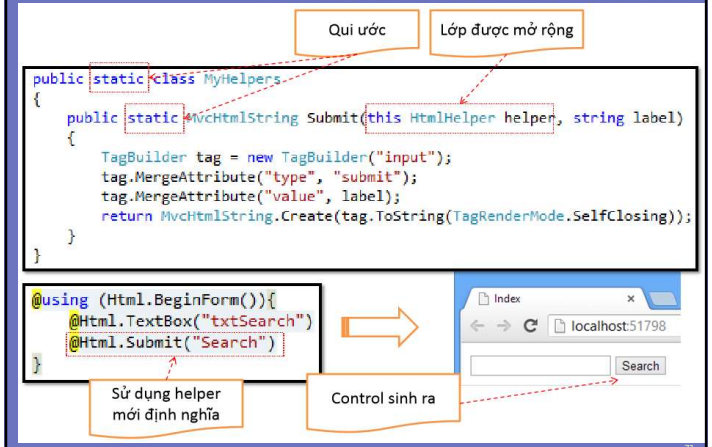
- Implicit UI

```
public class Student
{
    [DisplayName("Mã sinh viên")]
    public String Id { get; set; }
    [DisplayName("Mật khẩu"), DataType(DataType.Password)]
    public String Password { get; set; }
    [DisplayName("Họ và tên")]
    public String FullName { get; set; }
    [DisplayName("Giới tính")]
    public bool Gender { get; set; }
    [DisplayName("Ngày sinh")]
    public DateTime Birthday { get; set; }
    [DisplayName("Ghi chú"), DataType(DataType.MultilineText)]
    public String Notes { get; set; }
}
```

```
@model Mvc5CodeDemo.Models.Student
@using (Html.BeginForm())
{
    @Html.EditorForModel()
    <input type="submit" value="Register" />
}
```



Custom Helpers example



Custom Helpers

- Write extension methods for the HtmlHelper
  - Return string or override ToString method
  - TagBuilder manages closing tags and attributes
  - Add namespace in web.config (if needed)

```
public static class HtmlhelperExtensions
{
    public static TagBuilder Image(this HtmlHelper helper,
                                   string imageUrl, string alt)
    {
        TagBuilder imageTag = new TagBuilder("img");
        imageTag.MergeAttribute("src", imageUrl);
        imageTag.MergeAttribute("alt", alt);
        return imageTag;
    }
}
```

```
@Html.Image("image.jpg", "Just image");
```

Extension method

```
public static class XHtmlHelper
{
    public static MvcHtmlString Submit(this HtmlHelper helper, String label,
                                       String name = null, object htmlAttributes = null)
    {
        var tag = new TagBuilder("input");
        tag.Attributes["type"] = "submit";
        tag.Attributes["value"] = label;
        if (name != null)
        {
            tag.Attributes["name"] = name;
        }
        if (htmlAttributes != null)
        {
            var attributes = htmlAttributes.GetType().GetProperties();
            foreach (var a in attributes)
            {
                tag.Attributes[a.Name] = a.GetValue(htmlAttributes).ToString();
            }
        }
        return MvcHtmlString.Create(tag.ToString(TagRenderMode.SelfClosing));
    }
}
```

Use Helper

```
@Html.Submit("Save")
```

Sinh mã HTML: `<input type="submit" value="Save" />`

```
@Html.Submit("Save", "Command")
```

Sinh mã HTML: `<input name="Command" type="submit" value="Save" />`

```
@Html.Submit("Save", "Command", new { @class = "btn", id="save" })
```

Sinh mã HTML: `<input class="btn" id="save" name="Command" type="submit" value="Save" />`

73

Layout

- Define a common site template
- Similar to ASP.NET master pages (but better!)
- Razor view engine renders content inside-out
  - First view is rendered, then layout
- @RenderBody() – indicate where we want the views based on this layout to "fill in" their core content at that location in the HTML

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="utf-8" />
    </head>
    <body>
        <nav>@* Menu *@</nav>
        <div id="body">
            @RenderBody()
        </div>
        <footer>@* Footer *@</footer>
    </body>
</html>
```

74

Views and Layouts

- Views don't need to specify layout since their default layout is set in their _ViewStart file:
  - ~/Views/_ViewStart.cshtml (code for all views)
- Each view can specify custom layout pages

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

- Views without layout:

```
@{
  Layout = null;
}
```

75

Sections

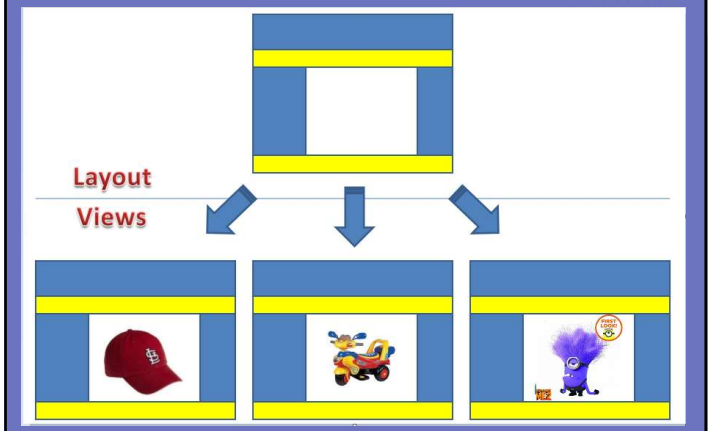
- You can have one or more "sections" (optional)
- They are defined in the views:

```
@section SideBar {
    <aside>
        Some side information
    </aside>
}
```

- And may be rendered anywhere in the layout page using the method RenderSection()
  - @RenderSection(string name, bool required)
  - If the section is required and not defined, an exception will be thrown (IsSectionDefined())

76

Layout



77

Layouts

@RenderBody()

Layout (_layout.cshtml)

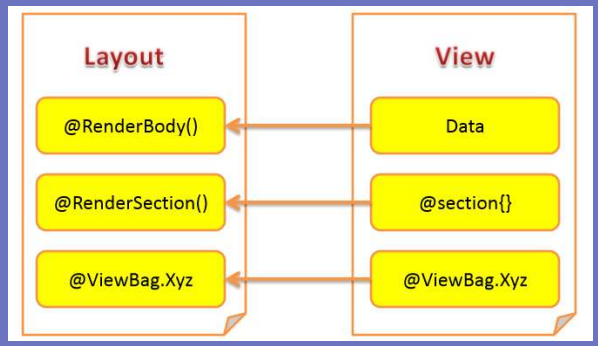
```
<html>
<head>
...
@RenderBody()
</body>
</html>
```

View (Index.cshtml)

```
<p>
Info from Index view
</p>
```

79

Layouts



78

Layouts

@RenderSection()

Layout (_layout.cshtml)

```
<html>
<head></head>
<body>
@RenderSection("fred")
@RenderSection("bob")
</body>
</html>
```

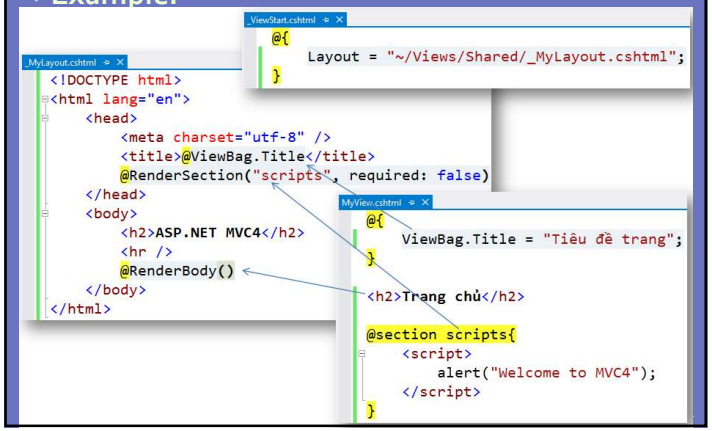
View (Index.cshtml)

```
@section fred
{
<p>I am Fred</p>
}
@section bob
{
<p>I am Bob</p>
}
```

80

Layouts

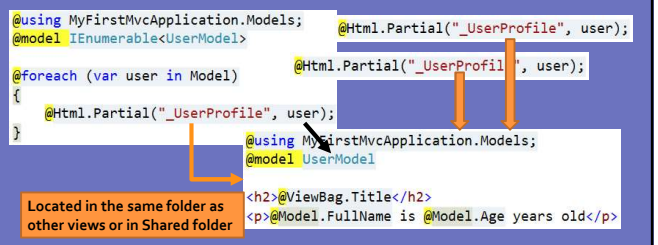
- Example:



Partial Views

- Partial views render portions of a page
  - Reuse pieces of a view
  - Html helpers – Partial, RenderPartial and Action
- Razor partial views are still .cshtml files

Sub-request



Resource package

BundleConfig.cs

```
bundles.Add(new ScriptBundle("~/bundles/jquery").Include(
            "~/Scripts/jquery-{version}.js"));

bundles.Add(new ScriptBundle("~/bundles/jqueryval").Include(
            "~/Scripts/jquery.validate*"));

bundles.Add(new ScriptBundle("~/bundles/modernizr").Include(
            "~/Scripts/modernizr-*"));

bundles.Add(new ScriptBundle("~/bundles/bootstrap").Include(
          "~/Scripts/bootstrap.js",
          "~/Scripts/respond.js"));

bundles.Add(new StyleBundle("~/Content/css").Include(
          "~/Content/bootstrap.css",
          "~/Content/site.css"));
```

Layout/view

```
@Styles.Render("~/Content/css")
@Scripts.Render("~/bundles/modernizr")
```

Partial Views

- @Html.Partial():

_Layout.cshtml

- @Html.Partial("_LoginPartial")

_LoginPartial.cshtml

```
@if (Request.IsAuthenticated) {
    <text>
        Hello, @Html.ActionLink(User.Identity.Name,
        @using (Html.BeginForm("LogOff", "Account",
            @Html.AntiForgeryToken()
            <a href="javascript:document.getElementB
        }
    </text>
} else {
    <ul>
        <li>@Html.ActionLink("Register", "Register",
        <li>@Html.ActionLink("Log in", "Login", "Acc
    </ul>
}
```

Partial Views

- @Html.Action():

```
<!--vmenu::chủng loại-->
@Html.Action("Category","Layout")
<!--/vmenu::chủng loại-->
```

LayoutController.cs

```
[ChildActionOnly]
public ActionResult Category()
{
    return PartialView("_Category", db.Categories);
}
```

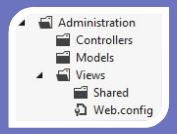
_Category.cshtml

```
@model IEnumerable<Category>

<ul class="art-vmenu">
@foreach(var p in Model){
    <li>
        <a href="/Product/Search?CategoryId=@p.Id">
            @p.Name</a>
    </li>
}
</ul>
```

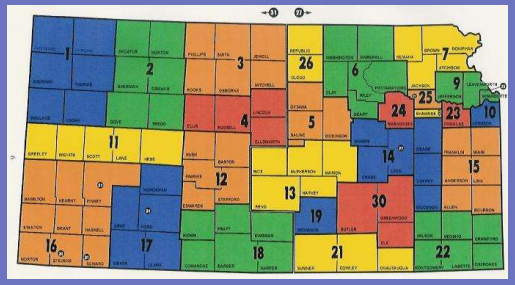
Areas

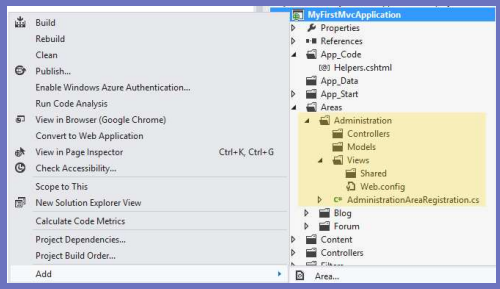
- Some applications can have a large number of controllers
- ASP.NET MVC lets us partition Web applications into smaller units (areas)
- An area is effectively an MVC structure inside an application
- Example: large e-commerce application
  - Main store, users
  - Blog, forum
  - Administration



87

Areas





Demo: Areas

ASP.NET MVC structures (areas)

Summary

- Routes maps URLs to controllers and actions
- Controllers are the brain of our application
- Actions are the ultimate request destination
- Razor is a powerful engine for combining models and templates into HTML code
  - Layout, sections, partials views and helpers help us to divide our views into pieces
- Our project can be divided into smaller parts containing controllers (areas)

89

Homework

1. Write down in a text file all the major similarities and differences you can find between ASP.NET Web Forms and ASP.NET MVC
2. Using ASP.NET MVC write the same web calculator as: http://www.gwebtools.com/bit-calculator
3. Create a simple informational ASP.NET MVC application by your choice with at least 3 controllers, 1 area, 1 custom route, 5 views (at least 1 partial view and 1 section). Using data is not required.
4. * Create a custom route constraint that allows requests only if the controller name starts with the string "Admin"

91