Școala informală de IT

Views

View

- Defines how the data will be displayed to the user(wrapped in HTML)
- May support master views (layouts) and sub-views (partial views or controls)
- Are templates that help us fill & generate HTML code that contains data from the server
- The HTML sent back by to the browser is generated by the View engine

View

- Handle the presentation logic
- The path to the view is inferred from the name of the controller and the name of the controller action.
  - http://{hostname}/{Controller}/{Action}
- A view is a CSHTML document that can contain html and server-side code(Razor) that is returned by an action
  - @ - inline expressions
  - @{...} - code blocks

View Selection

- The convention states that a View is searched in the /Views/[controller] (the folder specific to the controller)
- If the view is not found in there it will be searched into /Shared/View
- If the view is not found there it will throw an error

Returning Views from Actions

- A view is return from an action by specifying one of the overload
- return View();
  - will search a View with the same name as the action
- return View(model);
  - will search a View with the same name as the action and pass the model
- return View(“MyCoolView”);
  - will search a View with the name “MyCoolView”
- return View(model, “MyCoolView”);
  - will search a view with the name “MyCoolView” and pass the model

Pass Data to a View

- Passing the model : View(model);
- With ViewData:
  - ViewData["message"] = "Hello World!";
  - Strongly typed ViewData:
    - ViewData.Model = model;
- With ViewBag:
  - ViewBag.Message = "Hello World!";
  - ViewBag is dynamic property of BaseController class

Razor syntax

Razor Syntax

- The Razor syntax consists of Razor markup, C#, and HTML
- 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>
```

Razor Syntax

- 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>
```

Razor Syntax

- **@(…)** – 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>
```

Razor Syntax

```
@model IEnumerable<Store.Domain.Entities.Customer>
@{
    var total = Model.Count();
    var pages = Math.Round((decimal)(total / 30));
}

<h1>@ViewData["Title"]</h1>
<h2>@ViewBag.CoolTitle</h2>
<p>Total items: @total</p>

<p>Should be @pages pages</p>

<p>
    <a asp-action="Create">Create New</a>
</p>
```

HTML Helpers

HTML Helpers

- Extension methods which generate html elements
  - Example: `Html.TextBox()`
- It is advisable to use "For" extension methods to use strong-type model
  - Example: `Html.TextBoxFor()`
- Usage is optional
- You can create your own HTML Helpers

Standard HTML Helpers

Html Helper	Strongly Typed Html Helpers	Html Control
Html.ActionLink		Anchor link
Html.TextBox	***Html.TextBoxFor***	Textbox
Html.TextArea	***Html.TextAreaFor***	TextArea
Html.CheckBox	***Html.CheckBoxFor***	Checkbox
Html.RadioButton	***Html.RadioButtonFor***	Radio button
Html.DropDownList	***Html.DropDownListFor***	Dropdown, combobox
Html.ListBox	***Html.ListBoxFor***	multi-select list box
Html.Hidden	***Html.HiddenFor***	Hidden field
Html.Password	***Html.PasswordFor***	Password textbox
Html.Display	***Html.DisplayFor***	Html text
Html.Label	***Html.LabelFor***	Label
Html.Editor	***Html.EditorFor***	Generates Html controls based on data type of specified model property e.g. textbox for string property, numeric field for int, double or other numeric type

Html.TextBox vs Html.TextBoxFor

```
@Html.TextBox("StudentName", "John", new { @class = "form-control" })
```

Renders to

```
<input class="form-control" id="StudentName" name="StudentName" type="text" value="John" />
```

and...

```
@model Student

@Html.TextBoxFor(m => m.StudentName, new { @class = "form-control" })
```

Renders to

```
<input class="form-control" id="StudentName" name="StudentName" type="text" value="John" />
```

Html.TextBox vs Html.TextBoxFor

- `@Html.TextBox()` is loosely typed method whereas
- `@Html.TextBoxFor()` is a strongly typed (generic) extension method.
- TextBox requires property name as string parameter and TextBoxFor() requires lambda expression as a parameter.
- TextBox doesn't give you compile time error if you have specified wrong property name. It will throw runtime exception.
- TextBoxFor is generic method so it will give you compile time error if you have specified wrong property name or property name changes.

Html.EditorFor

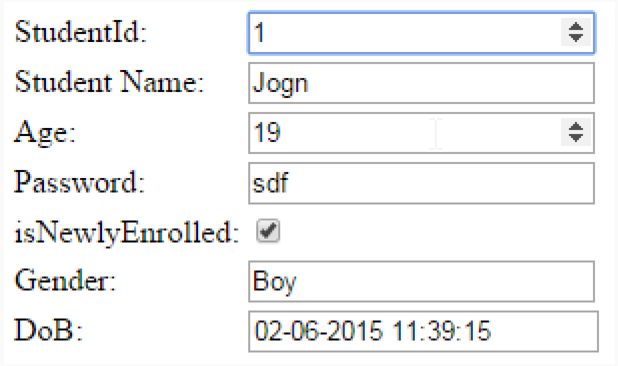
Property DataType	Html Element
string	<input type="text" >
int	<input type="number" >
decimal, float	<input type="text" >
boolean	<input type="checkbox" >
Enum	<input type="text" >
DateTime	<input type="datetime" >

Html.Editor, Html.EditorFor

```
StudentId:        @Html.Editor("StudentId")
Student Name:     @Html.Editor("StudentName")
Age:              @Html.Editor("Age")
Password:         @Html.Editor("Password")
IsNewlyEnrolled: @Html.Editor("IsNewlyEnrolled")
Gender:           @Html.Editor("Gender")
DoB:              @Html.Editor("DoB")
```

```
StudentId:        @Html.EditorFor(m => m.StudentId)
Student Name:     @Html.EditorFor(m => m.StudentName)
Age:              @Html.EditorFor(m => m.Age)
Password:         @Html.EditorFor(m => m.Password)
IsNewlyEnrolled: @Html.EditorFor(m => m.IsNewlyEnrolled)
Gender:           @Html.EditorFor(m => m.Gender)
DoB:              @Html.EditorFor(m => m.DoB)
```

Output of Editor / EditorFor helper



StudentId:	1
Student Name:	Jogn
Age:	19
Password:	sdf
isNewlyEnrolled:	✓
Gender:	Boy
DoB:	02-06-2015 11:39:15

Tag Helpers

Input TagHelper

- new feature and similar to HTML**helpers**, which help us render HTML

- server-side code to participate in creating and rendering HTML elements in Razor files

- HTML-friendly syntax

- The standard ones have distinct **asp-for** attributes that allow binding to a Model's Properties

Input TagHelper

- **Html Element**

```
<input type="text" id="Age" name="Age" value="@Model.Age" class="form-control" />
```

- **Html Helper**

```
@Html.TextBoxFor(m => m.Age, new { @class = "form-control" })
```

- **TagHelper**

```
<input asp-for="Age" class="form-control" />
```

- **To make available the tag helpers add next lines in _ViewImports.cshtml**

```
@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers
```

TagHelpers

- **Anchor tag**

```
<a asp-action="ActionName" asp-controller="ControllerName" asp-route="RouteName">...</a>
```

- **Form tag**

```
<form asp-action="ActionName" asp-controller="ControllerName" method="post"></form>
```

- **Input tag**

```
<input asp-for="FieldName" class="form-control" />
```

- **Validation tag and validation summary**

```
<span asp-validation-for="FieldName"></span>

<div asp-validation-summary="ModelOnly"></div>
```