# Topics – Form Helpers

- Input HTML Helpers &
  Strongly Typed Input HTML Helpers
- Security when sending data to the server
- SelectList, SelectListItem
- Exercise 1-2

- Templated Helper Methods &
  Strongly Typed Templated Helper Methods
- Model Metadata
- Editor for Model
- Exercises 3-4

# Feedback

Mandatory Assignment 1

# Presenting the catalogue

# Example 1: The Controller,

```
public class CatalogueController : Controller
{
    private Repository repository;
    public ActionResult Index()
    {
        repository = new Repository();
        ViewBag.Products = repository.Products;
        return View();
    }
}
```

# Example 1: The View

```
<h2>Music</h2>
@foreach (Product product in ViewBag.Products)
{
    if (product is MusicCD)
    {
        ...
    }
}
```

# Example 2: The Controller

```
private Repository repository = new Repository();
public ActionResult Index()
{
    IList<Book> books = repository.Products.OfType<Book>().ToList();
    ViewBag.Books = books;
    IList<Movie> movies = repository.Products.OfType<Movie>().ToList();
    ViewBag.Movies = movies;
    IList<MusicCD> musicCDs = repository.Products.OfType<MusicCD>().ToList();
    ViewBag.MusicCDs = musicCDs;
    return View();
}
```

# Example 2: The View

```
<div id="musicCD">
    <h1>MusicCD's</h1>
    @foreach (MusicCD musicCD in ViewBag.MusicCDs)
    {
        @RenderCD(musicCD)
    }
</div>
```

# Sub-class controllers (example 1)

```
public Movie(string title, decimal price, string imageUrl,
string director)
{
    this.title = title;
    this.price = price;
    this.imageUrl = imageUrl;
    this.director = director;
}
```

# Sub-class controllers (example 1)

```
public Movie(string title, decimal price, string imageUrl,
string director) : base (title, price, imageUrl) // calling
the base class controller
{
    this.director = director;
}
```

# Calculating Total Price

# Calculating TotalPrice in a View

```
@foreach (OrderItem orderITEM in invoice.OrderItems)
{
    @orderITEM.Product.Price @(" DKK")<br />
    totalPrice = Decimal.Add(totalPrice, orderITEM.Product.Price);
}
<strong><u>@totalPrice DKK</u></strong>
```

# TotalPrice

## Read-only property in the **Invoice** class

```
public decimal TotalPrice{
    get {
        foreach (OrderItem item in orderItems)        {
            totalPrice += item.Product.Price * item.Quantity;
        }
        return totalPrice;
    }
}
```

# Displaying **TotalPrice** in the View

```
@foreach (OrderItem orderITEM in invoice.OrderItems)
{
    @orderITEM.Product.Price @(" DKK")<br />
}
<strong><u>@invoice.TotalPrice DKK</u></strong>
```

# Advantages of having basic calculations inside the class itself

1. Super easy to get a result by referencing a property.
2. These calculations will be accessible from all other classes, controllers and views.
3. If you want to change anything, you'll do it in one single place.

# Form Helper Method

# Html.BeginForm

Action Method

Controller

HTTP Method

```
@{Html.BeginForm("Search", "Home", FormMethod.Get);}
    <input type="text" name="q" />
    <input type="submit" value="Search" />
@{Html.EndForm();}
```

# Html.BeginForm

Action Method

Controller

HTTP Method

```
@using (Html.BeginForm("Search", "Home", FormMethod.Get))
{
    <input type="text" name="q" />
    <input type="submit" value="Search" />
}
```

The HTML **BeginForm** method returns an object
that implements **iDisposable** interface with the **Dispose** method

https://msdn.microsoft.com/en-us/library/system.web.mvc.html.formextensions.beginform(v=vs.118).aspx

# To GET or to POST?

- A GET request represents an idem potent, read-only operation .

  You can send a GET request to a server repeatedly with no ill effects, because a GET does not (or should not ) change state on the server

- A POST request generally modifies state on the server

  Repeating POST requests might produce undesirable effects (such as double billing). Many browsers help a user avoid repeating a POST request .

# Default HTTP method?

- ASP.NET MVC forms?
  - POST
- HTML form?
  - GET

# The Overloads of the **BeginForm** Helper Method

| Overload | Description |
|---|---|
| `BeginForm()` | Creates a form which posts back to the action method it originated from |
| `BeginForm(`**`action, controller`**`)` | Creates a form which posts back to the action method and controller, specified as strings |
| `BeginForm(action, controller, `**`method`**`)` | As for the previous overload, but allows you to specify the value for the method attribute using a value from the System.Web.Mvc.FormMethod enumeration |
| `BeginForm(action, controller, method, `**`attributes`**`)` | As for the previous overload, but allows you to specify attributes for the form element an object whose properties are used as the attribute names |
| `BeginForm(action, controller,`**`routeValues`**`, method, attributes)` | As for the previous overload, but allows you to specify values for the variable route segments in your application routing configuration as an object whose properties correspond to the routing variables |

https://msdn.microsoft.com/en-us/library/system.web.mvc.html.formextensions.beginform(v=vs.118).aspx

# Input HTML Helpers

Make your life as a web developer easier

| HTML Element | Example |
|---|---|
| Check box | **`Html.CheckBox("myCheckbox", true)`**<br>**Output:**<br>`<input id="myCheckbox" name="myCheckbox" type="checkbox" value="true" />`<br>`<input name="myCheckbox" type="hidden" value="false" />` |
| Radio button | **`Html.RadioButton("myRadiobutton", "val", true)`**<br>**Output:**<br>`<input checked="checked" id="myRadiobutton" name="myRadiobutton" type="radio" value="val" />` |
| Hidden field | **`Html.Hidden("myHidden", "val")`**<br>**Output:**<br>`<input id="myHidden" name="myHidden" type="hidden" value="val" />` |
| Password | **`Html.Password("myPassword", "val")`**<br>**Output:**<br>`<input id="myPassword" name="myPassword" type="password" value="val" />` |

| HTML Element | Example |
| --- | --- |
| Text area | **`Html.TextArea("myTextarea", "val", 5, 20)`**<br>**Output**:<br>`<textarea cols="20" id="myTextarea" name="myTextarea" rows="5">val</textarea>` |
| Text box | **`Html.TextBox("myTextbox", "val")`**<br>**Output**:<br>`<input id="myTextbox" name="myTextbox" type="text" value="val" />` |

# Example of Use with a **Strongly Typed View**

```
@model lesson05_examples.Models.Person
@using(Html.BeginForm()) {
   <label>PersonId</label>
   @Html.TextBox("personId", @Model.PersonId) <br/>
   <label>First Name</label>
   @Html.TextBox("firstName", @Model.FirstName)
   <br/>
   <label>Last Name</label>
   @Html.TextBox("lastName", @Model.LastName) <br/>
   <input type="submit" value="Submit" />
}
```

Home/Form01

# The Controller

- To reference the model in the View, you must send an object (an instance of the model) to the View:

```
public class Person
{
    public int PersonId { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public DateTime BirthDate { get; set; }
    public string Phone { get; set; }
    public string Email { get; set; }
    public string Comment { get; set; }
}
```

```
public ActionResult Form01() {
    Person p = new Person() {
        PersonId = 1,
        FirstName = "Peter",
        LastName = "Mikkelsen",
    };
    return View(p);
}
```

# Strongly typed View -
An **overload** which takes a single string argument

```
@model lesson05_examples.Models.Person
@using(Html.BeginForm()) {
   <label>PersonId</label>
   @Html.TextBox("personId") <br/>
   <label>First Name</label>
   @Html.TextBox("firstName") <br/>
   <label>Last Name</label>
   @Html.TextBox("lastName") <br/>
   <input type="submit" value="Submit" />
}
```

Home/Form02

# Where Are ASP.NET MVC Views looking for the variables?

1. `ViewBag.DataValue`
2. `@Model.DataValue`

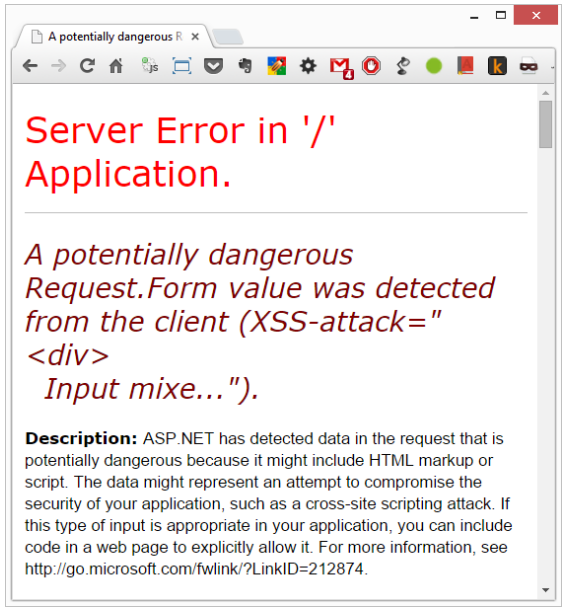
# Security – request validation

- **Request validation** is a feature in ASP.NET that examines an HTTP request and determines whether it contains potentially dangerous content. In this context, potentially dangerous content is any HTML markup or JavaScript code in the header, query string, body, or cookies of the request.

Dirty code:

```
<div>
   Input mixed with markup
</div>
```

Submit



# Allow HTML & JavaScript from input fields

- Application level (Web.config file)
- Action Method Attribute

```
[HttpPost]
[ValidateInput(false)]
public ActionResult Form03(FormCollection fc) {
      ViewBag.XSS_attack = fc["XSS-attack"];
      return View();
}
```

Form03.cshtml

# The Razor Engine encodes HTML and JavaScript tags

Dirty code:

```
<script>
  alert("bad code");
</script>
```

Submit

Source code:

```
17
18
19
20
<h2>Sanitized by the Razor View Engine</h2>
<p>&lt;script&gt;
alert(&quot;bad code&quot;);
&lt;/script&gt; </p>
```

**Sanitized by the Razor View Engine**

<script> alert("bad code"); </script>

Form03.cshtml

# Strongly Typed Input HTML Helpers

with Lambda Expressions

# Example of Strongly Typed Input HTML Helpers
## – with Lambda Expressions

```
@model lesson05_examples.Models.Person
@using(Html.BeginForm()) {
   <label>PersonId</label>
   @Html.TextBoxFor(m => m.PersonId) <br/>
   <label>First Name</label>
   @Html.TextBoxFor(m => m.FirstName) <br/>
   <label>Last Name</label>
   @Html.TextBoxFor(m => m.FirstName) <br/>
   <input type="submit" value="Submit" />
}
```

Form04.cshtml

# Strongly Typed Input HTML Helpers

| HTML Element | Example |
|---|---|
| Check box | **`Html.CheckBoxFor(x => x.IsApproved)`**<br>**Output:**<br>`<input id="IsApproved" name="IsApproved" type="checkbox" value="true" />`<br>`<input name="IsApproved" type="hidden" value="false" />` |
| Radio button | **`Html.RadioButtonFor(x => x.IsApproved, "val")`**<br>**Output:**<br>`<input id="IsApproved" name="IsApproved" type="radio" value="val" />` |
| Hidden field | **`Html.HiddenFor(x => x.PersonId)`**<br>**Output:**<br>`<input id="x.PersonId" name="x.PersonId" type="hidden" value="" />` |

| HTML Element | Example |
| --- | --- |
| Password | **`Html.Password(x => x.Password)`**<br>**Output:**<br>`<input id="Password" name="Password" type="password" />` |
| Text area | **`Html.TextAreaFor(x => x.Bio, 5, 20)`**<br>**Output:**<br>`<textarea cols="20" id="Bio" name="Bio" rows="5">`<br>`Bio value</textarea>` |
| Text box | **`Html.TextBoxFOr(x => x.FirstName)`**<br>**Output:**<br>`<input id="FirstName" name="FirstName" type="text" value="" />` |

# Advantages of using strongly typed Input Helpers

- Enables full **IntelliSense** support!

# How to add parameters to HTML elements?

- Pass an anonymously typed object

```
@Html.TextBoxFor(m => m.FirstName,
      new {style = "background-color:lightyellow", size="2"})

@Html.TextBox("LastName", "",
      new {@class = "active", maxlength="40"})
```

Form05.cshtml

# Dropdown lists

with **`SelectListItem`** elements

# Creating Select Elements

Controller

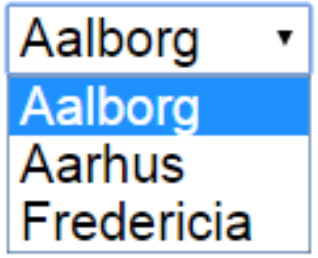
```
List<SelectListItem> postalDistricts = new List<SelectListItem>();
postalDistricts.Add(
      new SelectListItem { Text = "Aalborg", Value = "9000" }
);

ViewBag.PostalDistrict = postalDistricts;
return View();
```

```
<div>
    @Html.DropDownList("PostalDistrict")
</div>
```

Form06.cshtml

View



# Exercise 1-2

# Templated Helper Methods

# The Idea behind Templated Helper methods?

- You specify the property you want to display
- The MVC Framework figure out what HTML elements that are required

# The Model with Metadata Attribute

```
public class Person
{
    public int PersonId { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    [DataType(DataType.Date)]
    public DateTime BirthDate { get; set; }
}
```

# The View

```
<h2>Create Person</h2>
<div class="dataElem">
    @Html.Label("PersonId")
    @Html.Editor("PersonId")
</div>
<div class="dataElem">
    @Html.Label("FirstName")
    @Html.Editor("FirstName")
</div>
<div class="dataElem">
    @Html.LabelFor(m => m.LastName)
    @Html.EditorFor(m => m.LastName)
</div>
<div class="dataElem">
    @Html.LabelFor(m => m.BirthDate)
    @Html.EditorFor(m => m.BirthDate)
</div>
```

# The Output

# The HTML Source Code

```
<h2>Create Person</h2>
<div class="dataElem">
    <label for="PersonId">PersonId</label>
    <input class="text-box single-line" id="PersonId" name="PersonId" type="number" value="" />
</div>
<div class="dataElem">
    <label for="FirstName">FirstName</label>
    <input class="text-box single-line" id="FirstName" name="FirstName" type="text" value="" />
</div>
<div class="dataElem">
    <label for="LastName">LastName</label>
    <input class="text-box single-line" id="LastName" name="LastName" type="text" value="" />
</div>
<div class="dataElem">
    <label for="BirthDate">BirthDate</label>
    <input class="text-box single-line" id="BirthDate" name="BirthDate" type="date" value="" />
</div>
```

# Validation markup

- As default, the form helper methods will insert validation information as part of the html markup. Eg.:

```
<input data-val="true" data-val-number="The field PersonId must be a
number." data-val-required="The PersonId field is required."
id="personId" name="personId" type="text" value="1" />
```

- You normally want that for client side validation, but you can disable it altogether in the view file:

```
@{
    Layout = null;
    Html.EnableClientValidation(false);
}
```

# The MVC Templated HTML Helpers

| Helper | Example | Description |
|---|---|---|
| Display | Html.Display("FirstName") | Renders a read-only view of the specified model property, choosing an HTML element according to the property's type and metadata |
| DisplayFor | Html.DisplayFor(x => x.FirstName) | Strongly typed version of the previous helper |
| Editor | Html.Editor("FirstName") | Renders an editor for the specified model property, choosing an HTML element according to the property's type and metadata |
| EditorFor | Html.EditorFor(x => x.FirstName) | Strongly typed version of the previous helper |
| Label | Html.Label("FirstName") | Renders an HTML <label> element referring to the specified model property |
| LabelFor | Html.LabelFor(x => x.FirstName) | Strongly typed version of the previous helper |

# Model Metadata

# Using Metadata to Control Display, Editing and Visibility

```
public class Person {
    [HiddenInput(DisplayValue = false)]
    public int PersonId { get; set; }
    [Display(Name="First name")]
    public string FirstName { get; set; }
    [Display(Name="Last name")]
    public string LastName { get; set; }
    [Display(Name = "Birth Date")]
    [DataType(DataType.Date)]
    public string BirthDay { get; set; }
}
```

**Create Person**

First name

Last name

Birth Date dd-mm-åååå

Create

# The Values of the DataType Enumeration

| Value | Description |
|---|---|
| `DateTime` | Displays a date and time (this is the default behavior for `System.DateTime` values) |
| `Date` | Displays the date portion of a `DateTime` |
| `Time` | Displays the time portion of a `DateTime` |
| `Text` | Displays a single line of text |
| `PhoneNumber` | Displays a phone number |
| `MultilineText` | Renders the value in a `textarea` element |
| `Password` | Displays the data so that individual characters are masked from view |
| `Url` | Displays the data as a URL (using an HTML `a` element) |
| `EmailAddress` | Displays the data as an e-mail address (using an `a` element with a `mailto href`) |

# Built-In MVC Framework View Templates

| | | | | |
|---|---|---|---|---|
| Boolean | Date | MultilineText | String | Url |
| Collection | EmailAddress | Number | Text | |
| Decimal | HiddenInput | Object | Tel | |
| DateTime | Html | Password | Time | |

```
public class Person
{
    ...
    [UIHint("MultilineText")]
    public string Comment { get; set; }
}
```

Comment

Create

Takes precedence over DataType

# Model Helper Method

# Whole-Model Templated Helpers (scaffolding)

```
public class Person {
    [HiddenInput(DisplayValue = false)]
    public int PersonId { get; set; }
    [Display(Name="First")]
    public string FirstName { get; set; }
    [Display(Name="Last")]
    public string LastName { get; set; }
    [Display(Name = "Birth Date")]
    [DataType(DataType.Date)]
    public DateTime BirthDate { get; set; }
    [DataType(DataType.PhoneNumber)]
    public string Phone { get; set; }
    [DataType(DataType.EmailAddress)]
    public string Email { get; set; }
}
```

```
<h2>Create Person</h2>

@using (Html.BeginForm()) {
    @Html.EditorForModel()
    <br />
    <input type="button"
     value="Create">
}
```

Form09.cshtml

# Scaffolding View (`@Html.EditorForModel`)

## Create Person

First name

Last name

Birth Date

dd-mm-åååå

Phone

Email

Comment

Create

| Helper | Example | Description |
|---|---|---|
| `DisplayForModel` | `Html.DisplayForModel()` | Renders a read-only view |
| `EditorForModel` | `Html.EditorForModel()` | Renders editor elements |
| `LabelForModel` | `Html.LabelForModel()` | Renders an HTML `<label>` element object |

# Form helpers: An overview

| HTML Form Helpers | | |
|---|---|---|
| **Input HTML Helpers** | Weakly typed | Strongly types |
| **Templated Helpers** | Weakly typed | Strongly types |

| |
|---|
| **Editor for Model** |
| **Form Helper Method** |

# Exercise 3-4