# MVC

Christen Zarif Foad

## OutLine

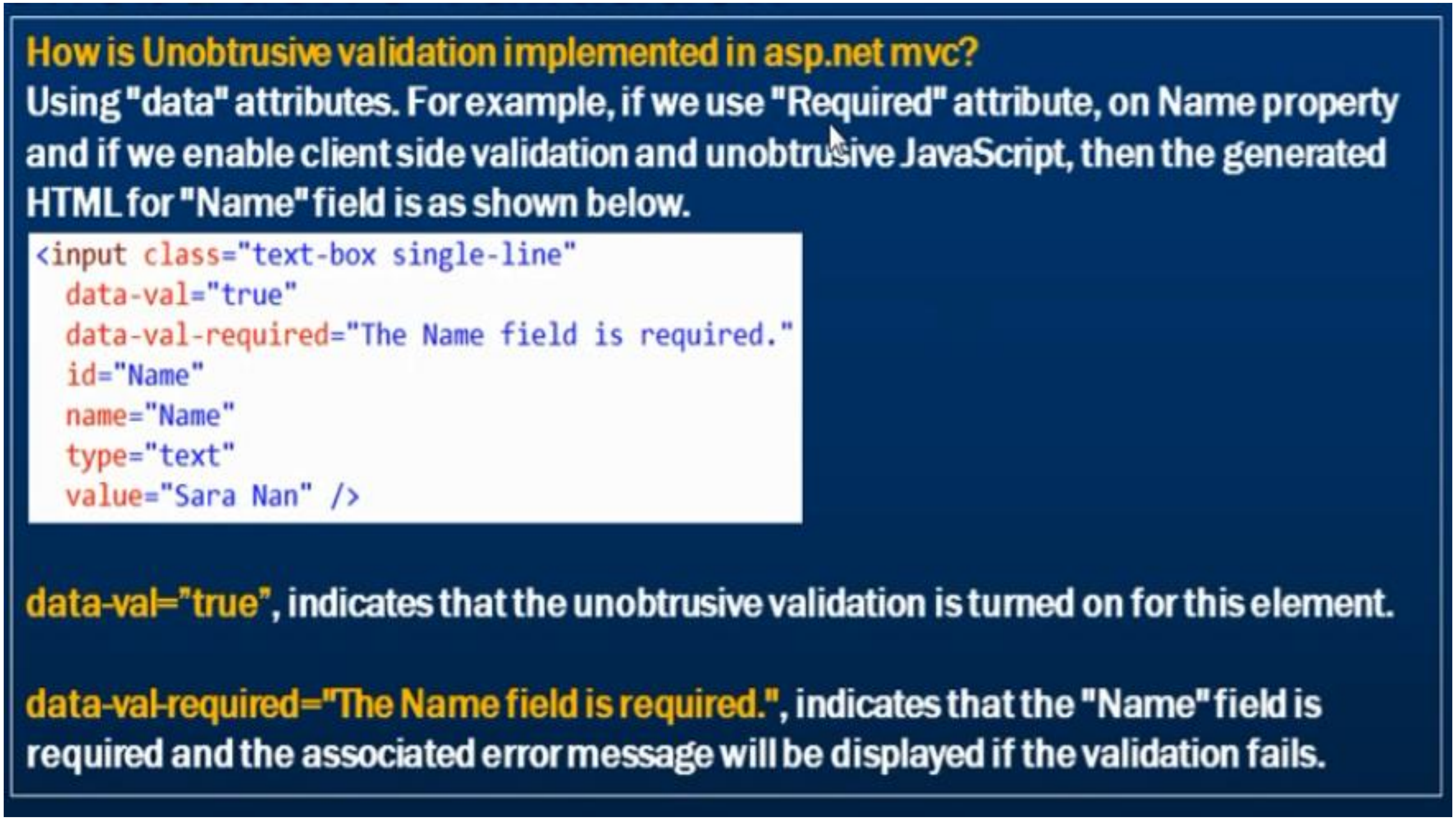
- Validation

## VALIDATION

## Unobtrusive javascript

- What is Unobtrusive Javascript?
  - Is a JavaScript that is separated from the web site's html markup.
- There are several benefits of using Unobtrusive Javascript
  - **Separation of Concerns**: i.e the HTML mark-up is now clean without any traces of javascript .page load time is faster
  - **It is also easy to update the code** : as all th e Javascript logic is present in a separate file
  - **We also get, better cache support**: as all our Javascript is now present in separate file , it can be cached and accessed much faster

# Unobtrusive Validation (Con.)



**How is Unobtrusive validation implemented in asp.net mvc?**

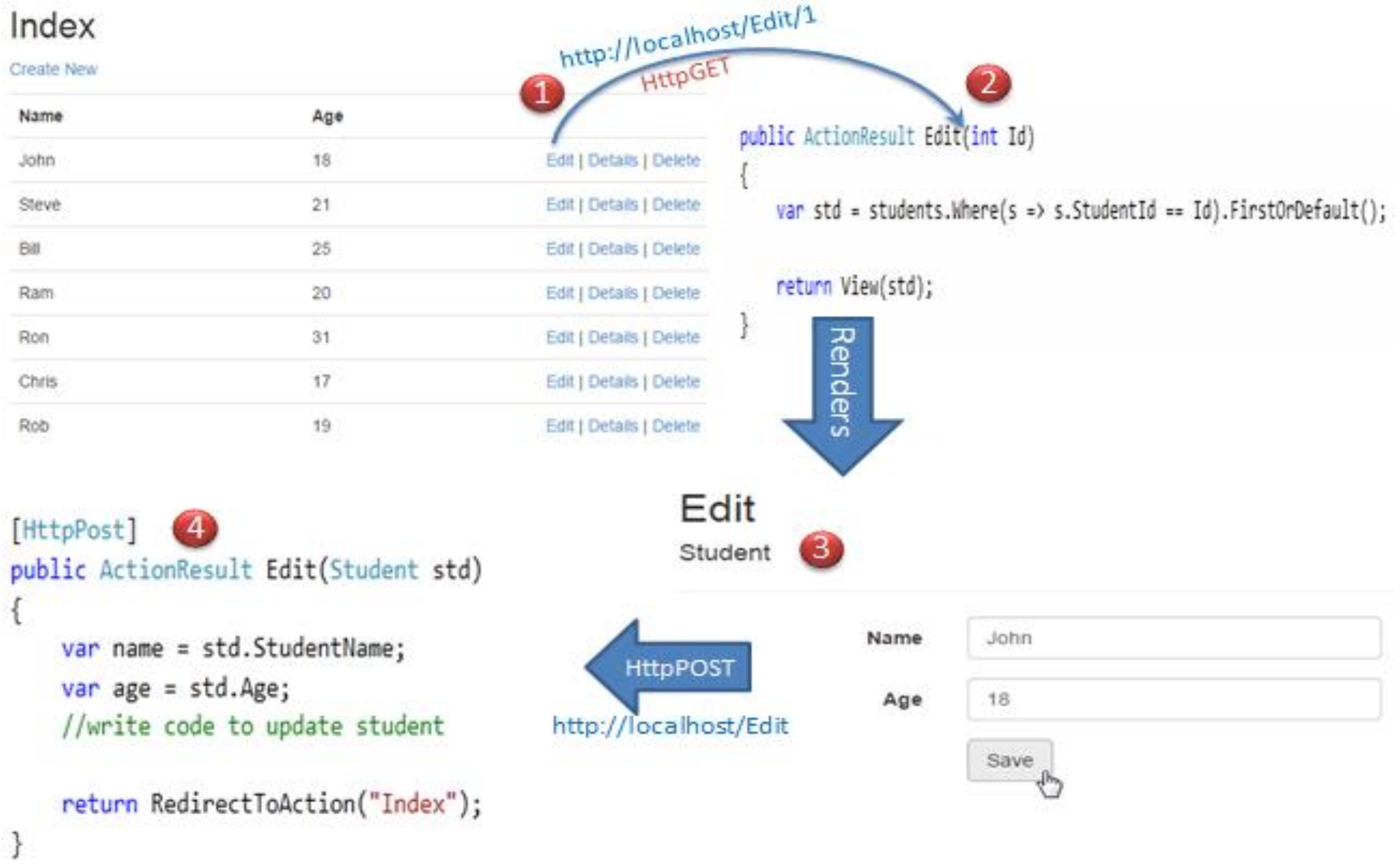
Using "data" attributes. For example, if we use "Required" attribute, on Name property and if we enable client side validation and unobtrusive JavaScript, then the generated HTML for "Name" field is as shown below.

```
<input class="text-box single-line"
  data-val="true"
  data-val-required="The Name field is required."
  id="Name"
  name="Name"
  type="text"
  value="Sara Nan" />
```

data-val="true", indicates that the unobtrusive validation is turned on for this element.

data-val-required="The Name field is required.", indicates that the "Name" field is required and the associated error message will be displayed if the validation fails.

# Edit View



## Index

Create New

| Name | Age | |
|---|---|---|
| John | 18 | Edit \| Details \| Delete |
| Steve | 21 | Edit \| Details \| Delete |
| Bill | 25 | Edit \| Details \| Delete |
| Ram | 20 | Edit \| Details \| Delete |
| Ron | 31 | Edit \| Details \| Delete |
| Chris | 17 | Edit \| Details \| Delete |
| Rob | 19 | Edit \| Details \| Delete |

```
public ActionResult Edit(int Id)
{
    var std = students.Where(s => s.StudentId == Id).FirstOrDefault();

    return View(std);
}
```

## Edit

Student

| | |
|---|---|
| Name | John |
| Age | 18 |

```
[HttpPost]
public ActionResult Edit(Student std)
{
    var name = std.StudentName;
    var age = std.Age;
    //write code to update student

    return RedirectToAction("Index");
}
```

## Data Validation

- In this Edit form implement data validation, which will display validation messages on the click of Save button , if Student Name or Age is blank.
- ASP.NET MVC uses **DataAnnotations** attributes for validation.
- **DataAnnotations** attributes can be applied to the properties of the model class to indicate the kind of value the property will hold.
- You can apply multiple DataAnnotations validation attributes to a single property if required.
- Each validation attribute assign to default message error.

# Validation

- The following validation attributes available by default

| Attribute | Description |
|---|---|
| **Required** | Indicates that the property is a required field |
| **StringLength** | Defines a maximum length for string field |
| **Range** | Defines a maximum and minimum value for a numeric field |
| **RegularExpression** | Specifies that the field value must match with specified Regular Expression |
| **CreditCard** | Specifies that the specified field is a credit card number |
| **CustomValidation** | Specified custom validation method to validate the field |
| **EmailAddress** | Validates with email address format |
| **FileExtension** | Validates with file extension |
| **MaxLength** | Specifies maximum length for a string field |
| **MinLength** | Specifies minimum length for a string field |
| **Phone** | Specifies that the field is a phone number using regular expression for phone numbers |

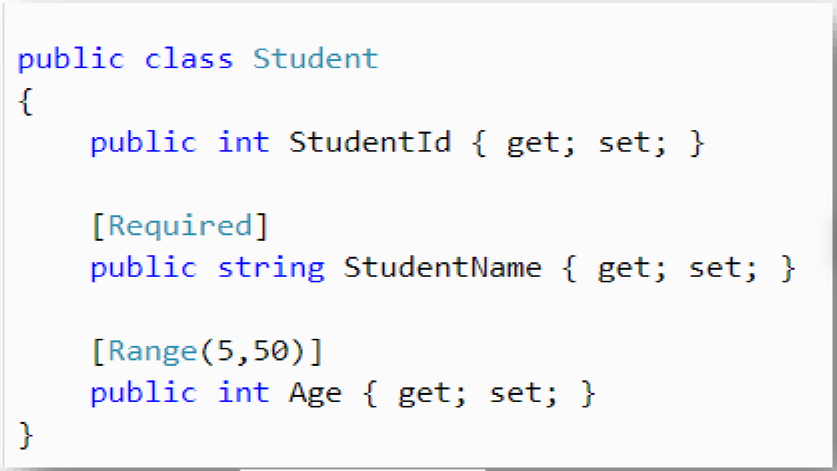
## Validation

- The following validation attributes available by default

| Attribute | Description |
|---|---|
| **ValidateNever** | indicates that a property or parameter should be excluded from validation |
| **Compare** | Validates that two properties in a model match |
| **Url** | Validates that the property has a URL format. |
| **Remote** | Validates input on the client by calling an action method on the server |

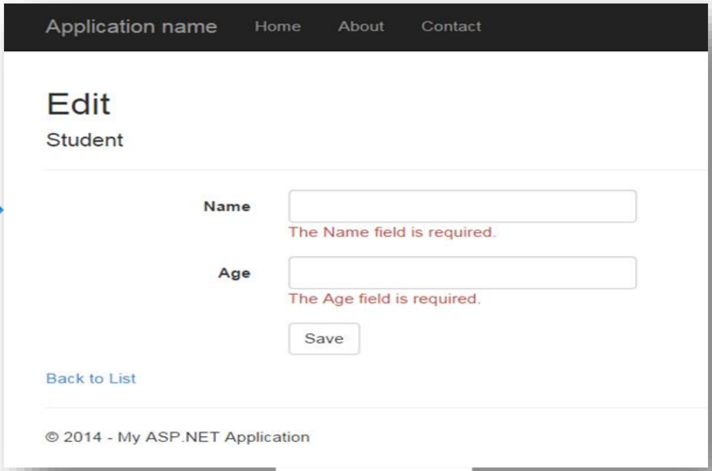
# Implement validation in edit view

- **Step 1**: First of all, apply **DataAnnotation** attribute on the properties of Student **model** class.
  - We want to validate
    1. **StudentName** and **Age** is not blank.
    2. **Age** should be between 5 and 50.
  - the MVC framework will **automatically display the default error message**, if the user tries to save the Edit form without entering the Student Name,the same with Range attribute.



```
public class Student
{
    public int StudentId { get; set; }

    [Required]
    public string StudentName { get; set; }

    [Range(5,50)]
    public int Age { get; set; }
}
```

Model



Edit

Student

Name

The Name field is required.

Age

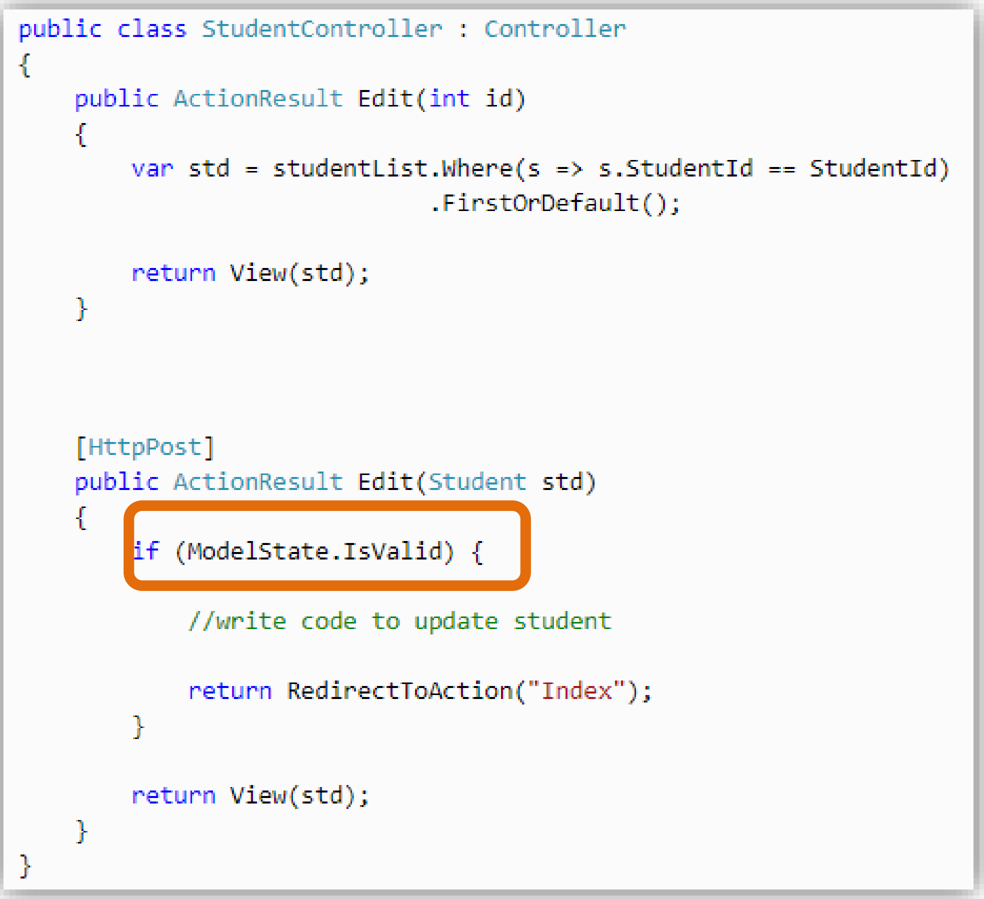
The Age field is required.

© 2014 - My ASP.NET Application

View

# Implement validation (Con.)

- **Step 2:** Create the GET and POST Edit Action method.
  - ❑ The **GET** action method will render Edit view to edit the selected student
  - ❑ The **POST** Edit method will save edited student.
  - ❑ **ModelState.IsValid**: determines that whether submitted values satisfy All the DataAnnotation Validation attributes applied to model properties.



```
public class StudentController : Controller
{
    public ActionResult Edit(int id)
    {
        var std = studentList.Where(s => s.StudentId == StudentId)
                             .FirstOrDefault();

        return View(std);
    }



    [HttpPost]
    public ActionResult Edit(Student std)
    {
        if (ModelState.IsValid) {

            //write code to update student

            return RedirectToAction("Index");
        }

        return View(std);
    }
}
```

## ModelState

- Model state represents errors that come from two subsystems: model binding and model validation.
- Errors that originate from model binding are generally data conversion errors.
  - For example, an "x" is entered in an integer field.
- Model validation occurs after model binding and reports errors where data doesn't conform to business rules.
  - For example, a 0 is entered in a field that expects a rating between 1 and 5.

## Implement Validation (Con.)

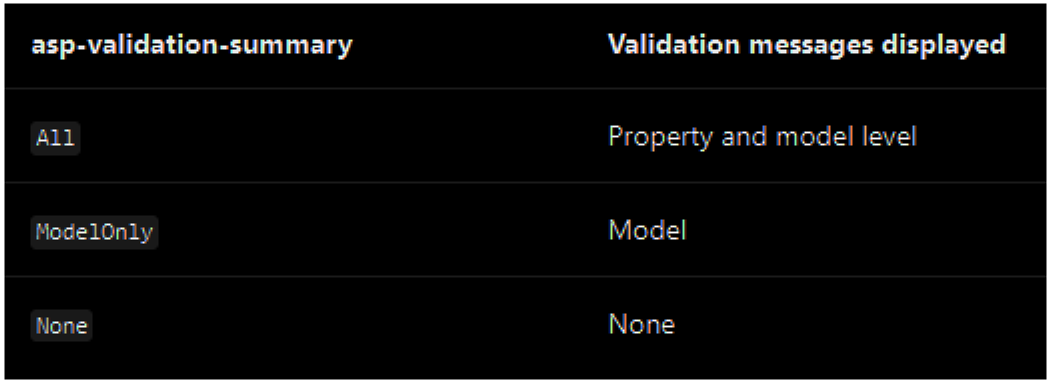
- **Step 3 :** in view used tag helper
- There are two Validation Tag Helpers.
  - The Validation Message Tag Helper (which displays a validation message for a single property on your model),
  - The Validation Summary Tag Helper (which displays a summary of validation errors).

# Implement Validation (Con.)

- **Step 3 :** in view used tag helper
  - The Input Tag Helper adds HTML5 client side validation attributes to input elements based on data **annotation** attributes on your model classes.
  - Validation is also performed on the server.
  - The Validation Tag Helper displays these error messages when a validation error occurs.

## Validation Summary Tag Helper

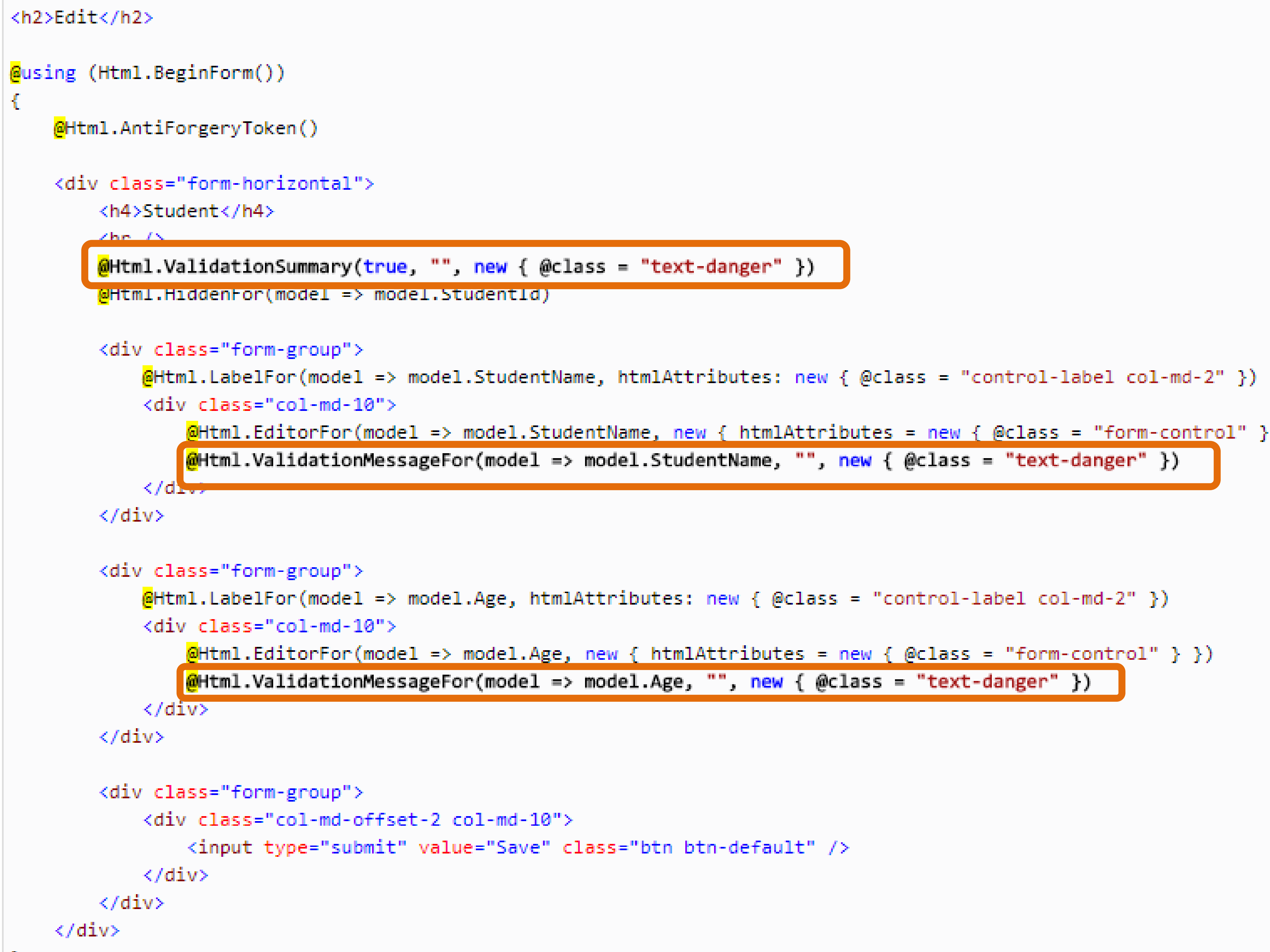
- Targets <div> elements with the asp-validation-summary attribute
- The asp-validation-summary attribute value can be any of the following:



| asp-validation-summary | Validation messages displayed |
|---|---|
| `All` | Property and model level |
| `ModelOnly` | Model |
| `None` | None |

## Implement Validation (Con.)

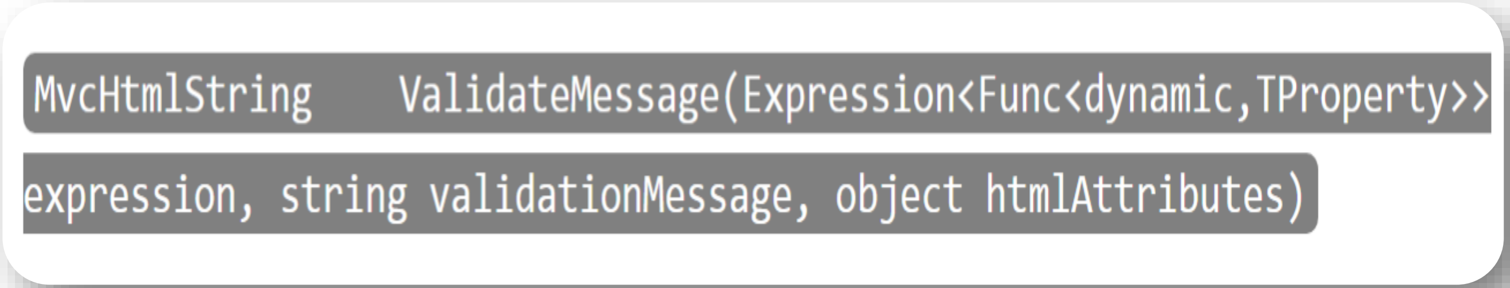
- **Step 3 :** in view used
  - Use **ValidationSummary** to display all the error messages in the view.
  - Use **ValidationMessageFor** or **ValidationMessage** helper method to display field level error messages in the view.
  - Or you can used scaffolding to create edit view using template with validation



```
<h2>Edit</h2>

@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()

    <div class="form-horizontal">
        <h4>Student</h4>
        <hr />
        @Html.ValidationSummary(true, "", new { @class = "text-danger" })
        @Html.HiddenFor(model => model.StudentId)

        <div class="form-group">
            @Html.LabelFor(model => model.StudentName, htmlAttributes: new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.StudentName, new { htmlAttributes = new { @class = "form-control" }
                @Html.ValidationMessageFor(model => model.StudentName, "", new { @class = "text-danger" })
            </div>
        </div>

        <div class="form-group">
            @Html.LabelFor(model => model.Age, htmlAttributes: new { @class = "control-label col-md-2" })
            <div class="col-md-10">
                @Html.EditorFor(model => model.Age, new { htmlAttributes = new { @class = "form-control" } })
                @Html.ValidationMessageFor(model => model.Age, "", new { @class = "text-danger" })
            </div>
        </div>

        <div class="form-group">
            <div class="col-md-offset-2 col-md-10">
                <input type="submit" value="Save" class="btn btn-default" />
            </div>
        </div>
    </div>
```

## ValidationMessageFor

- The **Html.ValidationMessageFor()** is a strongly typed extension method.
- It displays a validation message if an error exists for the specified field in the ModelStateDictionary object.
- Method Signature :

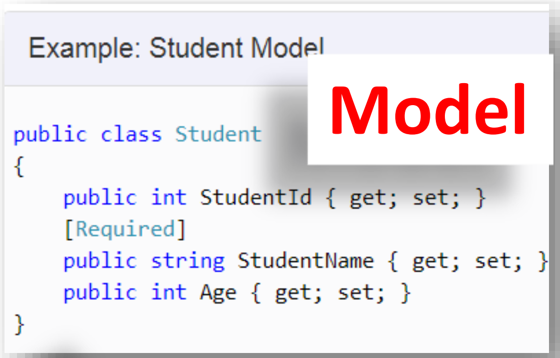


```
MvcHtmlString    ValidateMessage(Expression<Func<dynamic,TProperty>>
expression, string validationMessage, object htmlAttributes)
```

- **Method Parameters:**
  - **The first Parameters** :is a lambda expression to specify a property for which we want to show the error message.
  - **The second parameter** is for custom error message
  - **The third parameter** is for html attributes like css, style etc

# ValidationMessageFor (Con.)

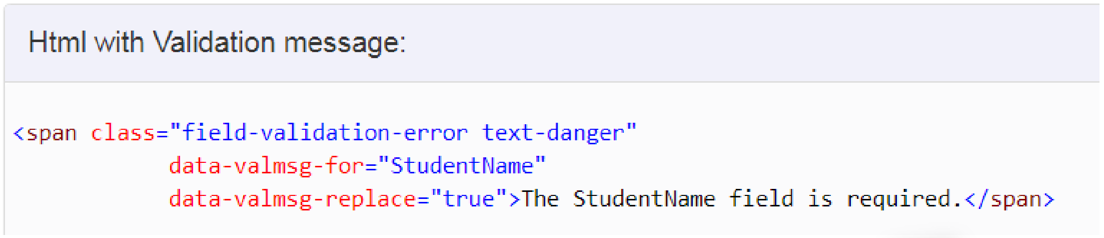
- This method will only display an error if you have configured DataAnnotations attribute to the specifed property in the model class.



Example: Student Model

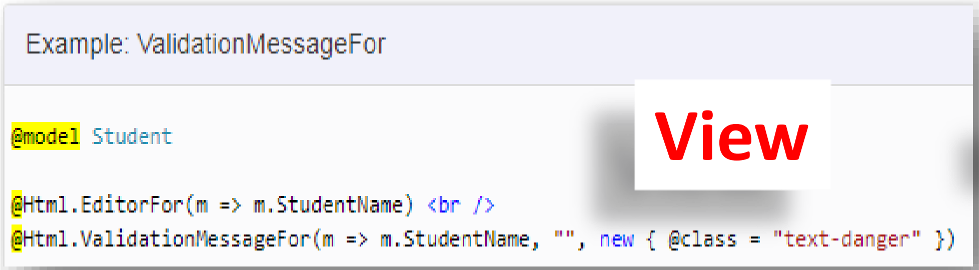
Model

```
public class Student
{
    public int StudentId { get; set; }
    [Required]
    public string StudentName { get; set; }
    public int Age { get; set; }
}
```



Html with Validation message:

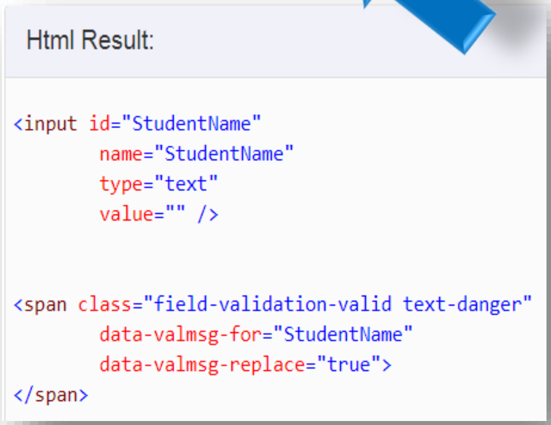
```
<span class="field-validation-error text-danger"
          data-valmsg-for="StudentName"
          data-valmsg-replace="true">The StudentName field is required.</span>
```



Example: ValidationMessageFor

View

```
@model Student

@Html.EditorFor(m => m.StudentName) <br />
@Html.ValidationMessageFor(m => m.StudentName, "", new { @class = "text-danger" })
```



Html Result:

```
<input id="StudentName"
       name="StudentName"
       type="text"
       value="" />


<span class="field-validation-valid text-danger"
       data-valmsg-for="StudentName"
       data-valmsg-replace="true">
</span>
```

# ValidationMessageFor (Con.)

- **Custom Error Message**
  - You can display your own error message instead of the default error message , using:
    - **DataAnnotations Attribute:**

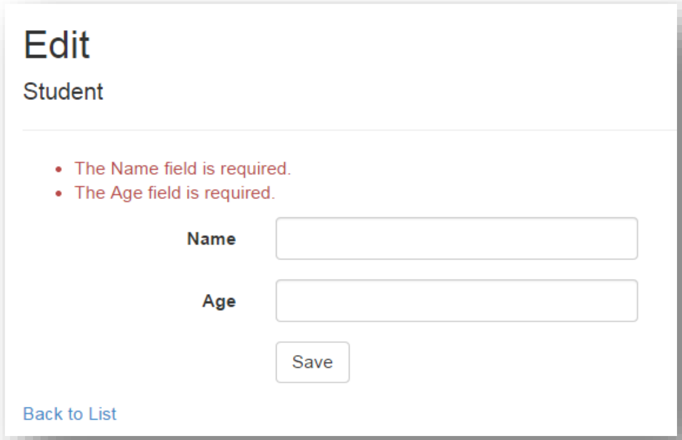
```
[Required(ErrorMessage="Please enter student name.")]
public string StudentName { get; set; }
```

- **ValidationMessageFor() Method:**

```
@Html.Editor("StudentName") <br />
@Html.ValidationMessageFor(m => m.StudentName,
"Please enter student name.", new { @class = "text-
danger" })
```

## ValidationSummary

- The ValidationSummary helper method generates an unordered list (ul element) of validation messages that are in the ModelStateDictionary object

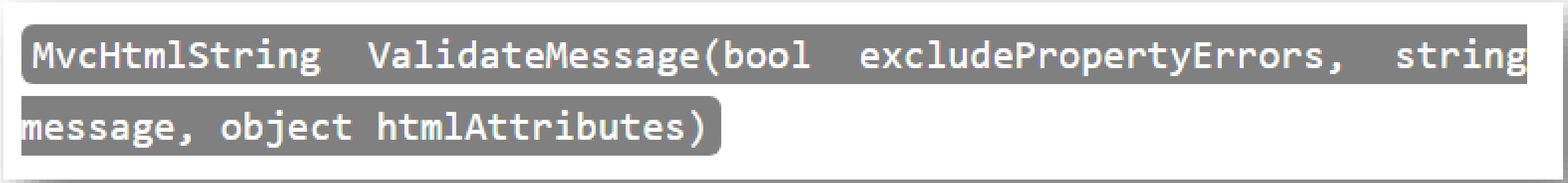


Edit

Student

- The Name field is required.
- The Age field is required.

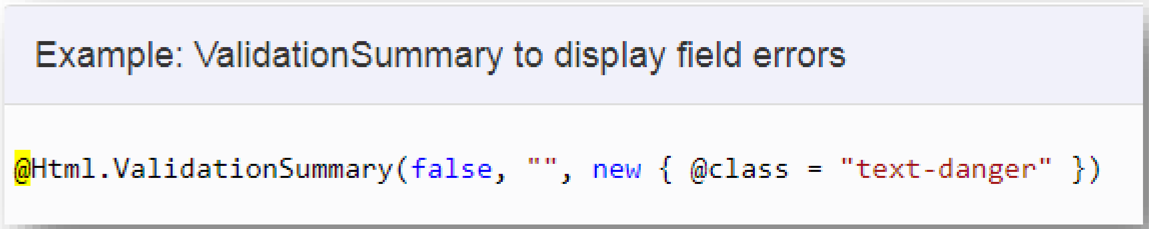
- Method Signature:



```
MvcHtmlString ValidateMessage(bool excludePropertyErrors, string
message, object htmlAttributes)
```

# ValidationSummary (Con.)

- By default, ValidationSummary filters out field level error messages. If you want to display field level error messages as a summary then specify **excludePropertyErrors = false**.



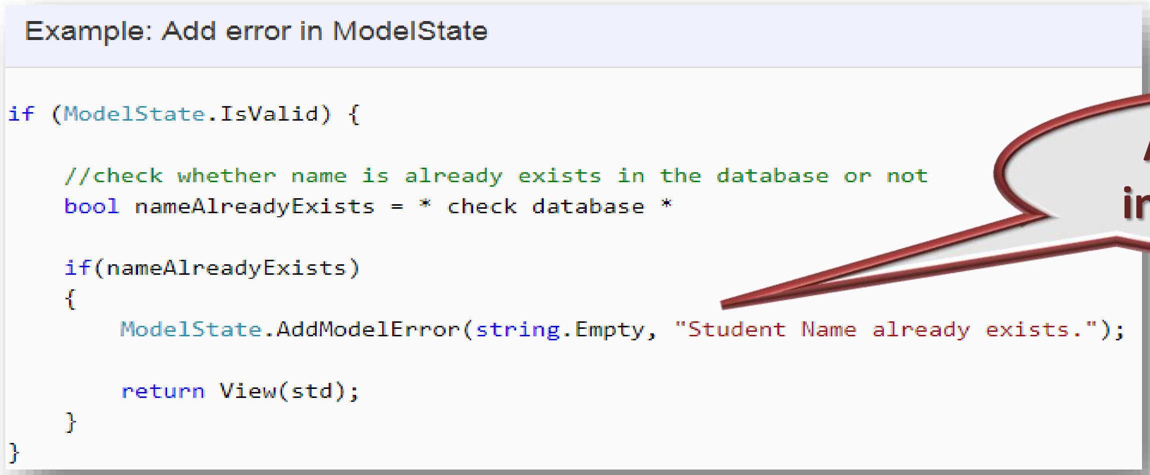
Example: ValidationSummary to display field errors

```
@Html.ValidationSummary(false, "", new { @class = "text-danger" })
```

- To display error messages as a summary at the top.
- Please make sure that you don't have a ValidationMessageFor method for each of the fields.

# ValidationSummary (Con.)

- **Display custom error messages:**
  - **For example**, we want to display a message if Student Name already exists in the database



Example: Add error in ModelState

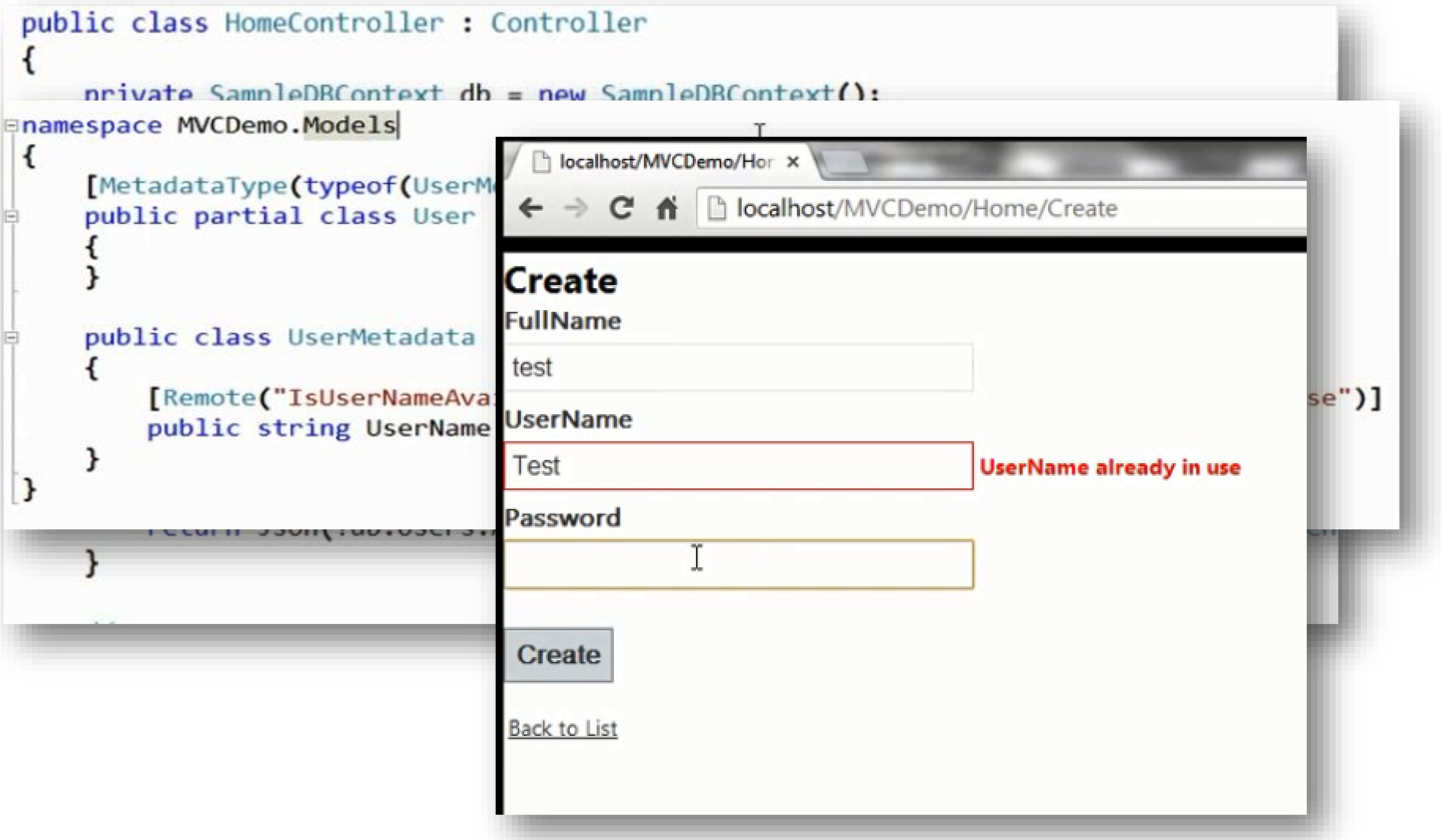
```
if (ModelState.IsValid) {

    //check whether name is already exists in the database or not
    bool nameAlreadyExists = * check database *

    if(nameAlreadyExists)
    {
        ModelState.AddModelError(string.Empty, "Student Name already exists.");

        return View(std);
    }
}
```

  - The **ValidationSummary** method will automatically display all the error messages added into **ModelState**.

## Remote Validation

- Remote Validation used to check whether the content enter in the input control is valid or not by sending an ajax request to server side to check it.
- The RemoteAttribute works by
  - making an **AJAX call** from the client to a controller action with the value of the field being validated.
  - The controller action then **returns a JsonResult** response indicating validation success or failure.

# Remote Validation (Con.)



```
public class HomeController : Controller
{
    private SampleDBContext db = new SampleDBContext();
```

```
namespace MVCDemo.Models
{
    [MetadataType(typeof(UserM
    public partial class User
    {
    }

    public class UserMetadata
    {
        [Remote("IsUserNameAva                                                    se")]
        public string UserName
    }
}
```

```
    }
```

localhost/MVCDemo/Hor

localhost/MVCDemo/Home/Create

**Create**

FullName

test

UserName

Test **UserName already in use**

Password

Create

Back to List

## Remote validation in ASP.NET MVC

This is the method which gets called to perform the remote validation. An AJAX request is issued to this method. If this method returns true, validation succeeds, else validation fails and the form is prevented from being submitted. The parameter name (UserName) must match the field name on the view. If they don't match, model binder will not be able bind the value with the parameter and validation may not work as expected.

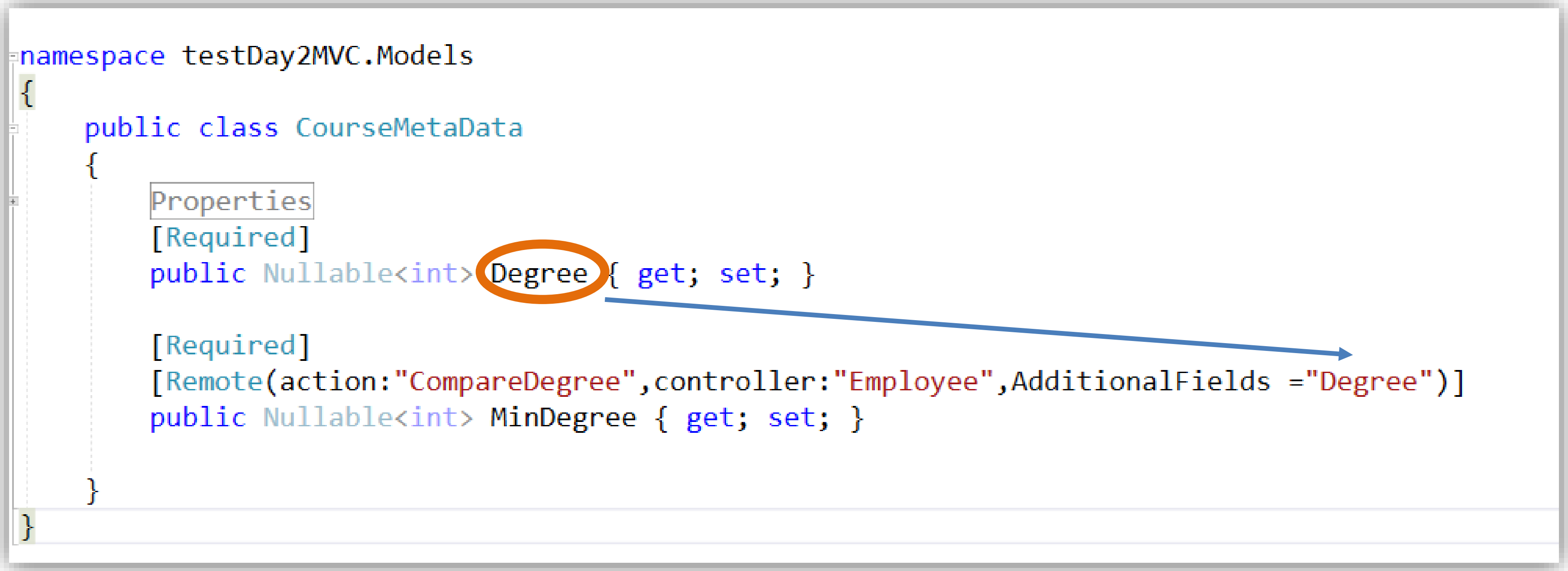
```
public JsonResult IsUserNameAvailable(string UserName)
{
    return Json(!db.Users.Any(x => x.UserName == UserName), JsonRequestBehavior.AllowGet);
}
```

```
// Notice that name of the method(IsUserNameAvailable), controller name (Home)
// & error message are passed as arguments to Remote Attribute
public class UserMetaData
{
    [Remote("IsUserNameAvailable", "Home", ErrorMessage = "UserName already in use.")]
    public string UserName { get; set; }
}
```

jQuery, jquery.validate &jquery.validate.unobtrusive script files are required for remote validation to work.

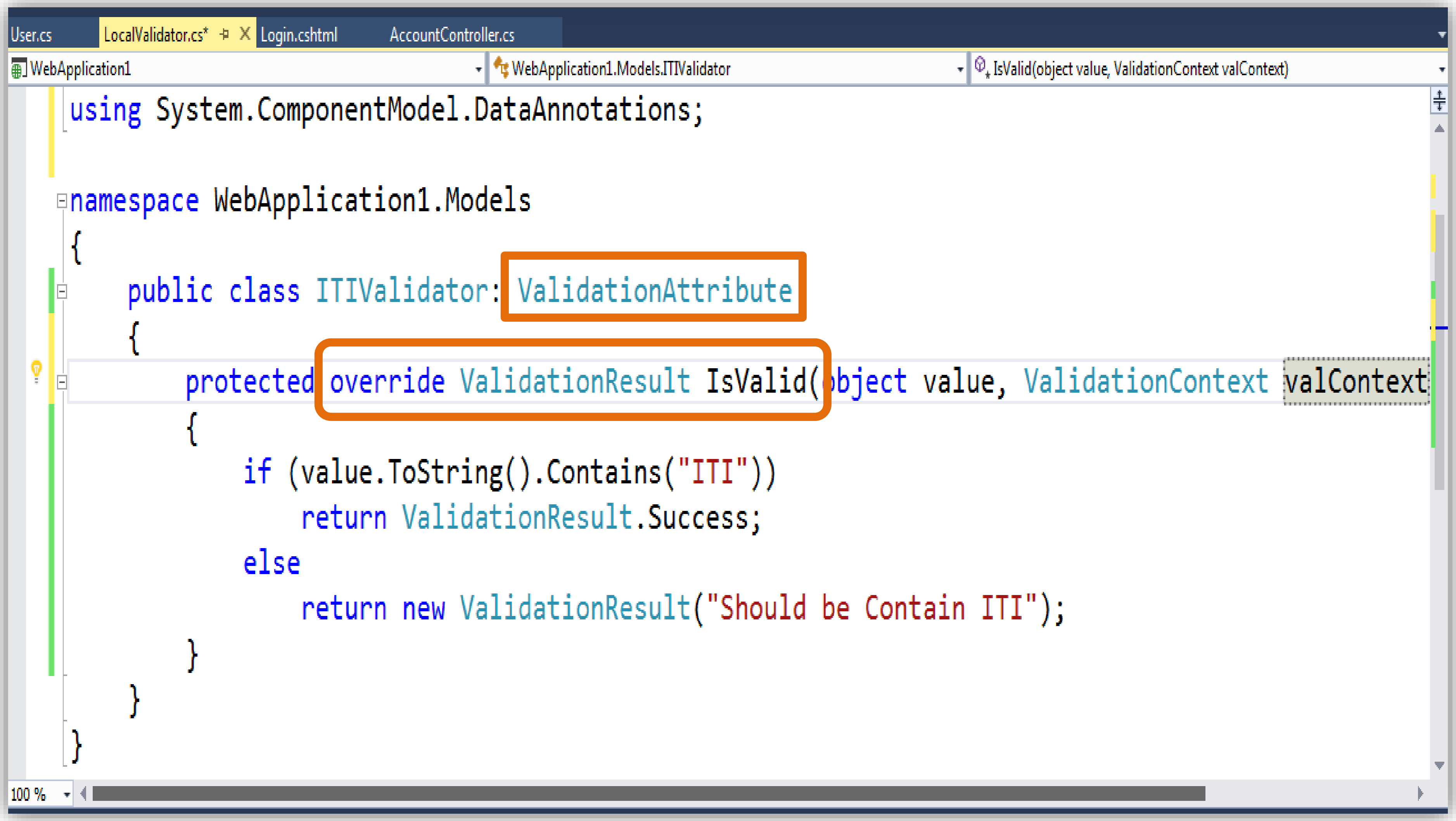
```
<script src="~/Scripts/jquery-1.7.1.min.js" type="text/javascript"></script>
<script src="~/Scripts/jquery.validate.min.js" type="text/javascript"></script>
<script src="~/Scripts/jquery.validate.unobtrusive.js" type="text/javascript"></script>
```

# Remote Attribute (Con.)



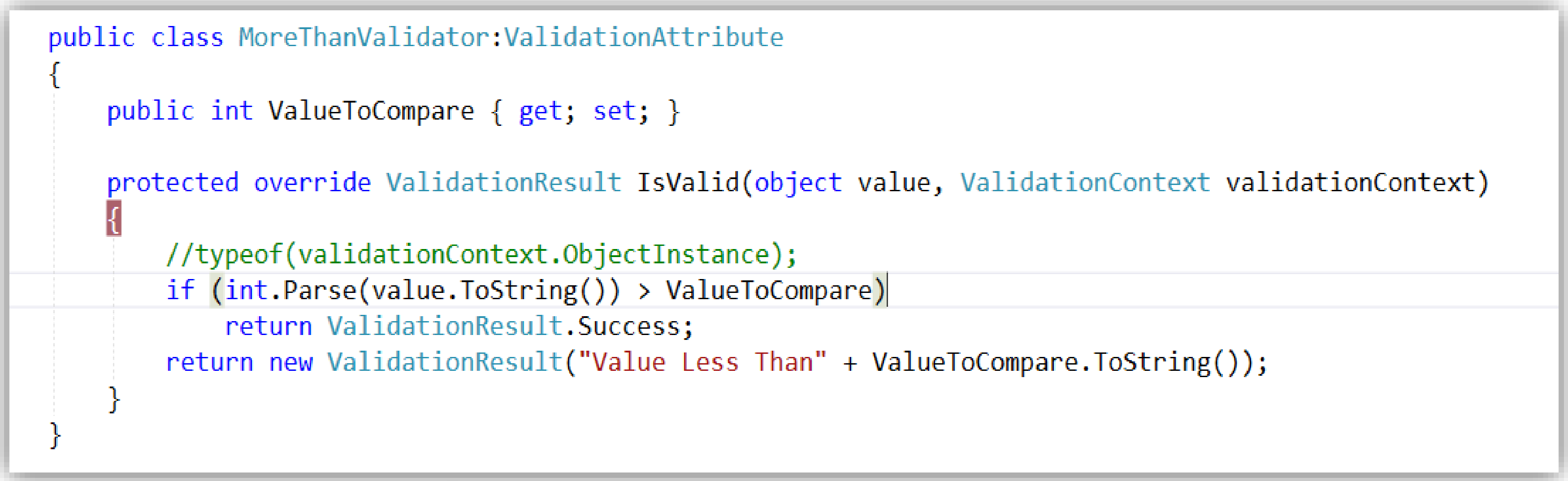
```
namespace testDay2MVC.Models
{
    public class CourseMetaData
    {
        Properties
        [Required]
        public Nullable<int> Degree { get; set; }

        [Required]
        [Remote(action:"CompareDegree",controller:"Employee",AdditionalFields ="Degree")]
        public Nullable<int> MinDegree { get; set; }

    }
}
```

# Custom Validation

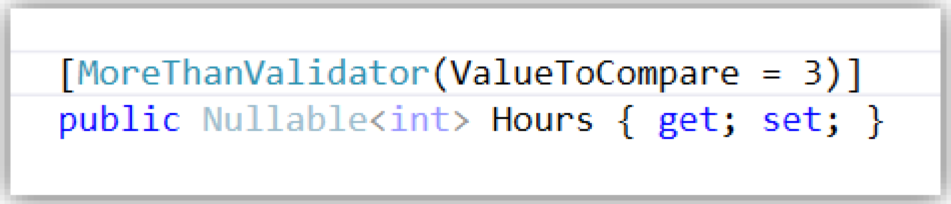

User.cs
LocalValidator.cs*
Login.cshtml
AccountController.cs
WebApplication1
WebApplication1.Models.ITIValidator
IsValid(object value, ValidationContext valContext)
100 %

```
using System.ComponentModel.DataAnnotations;

namespace WebApplication1.Models
{
    public class ITIValidator: ValidationAttribute
    {
        protected override ValidationResult IsValid(object value, ValidationContext valContext
        {
            if (value.ToString().Contains("ITI"))
                return ValidationResult.Success;
            else
                return new ValidationResult("Should be Contain ITI");
        }
    }
}
```

# Custom Validator



```
public class MoreThanValidator:ValidationAttribute
{
    public int ValueToCompare { get; set; }

    protected override ValidationResult IsValid(object value, ValidationContext validationContext)
    {
        //typeof(validationContext.ObjectInstance);
        if (int.Parse(value.ToString()) > ValueToCompare)
            return ValidationResult.Success;
        return new ValidationResult("Value Less Than" + ValueToCompare.ToString());
    }
}
```



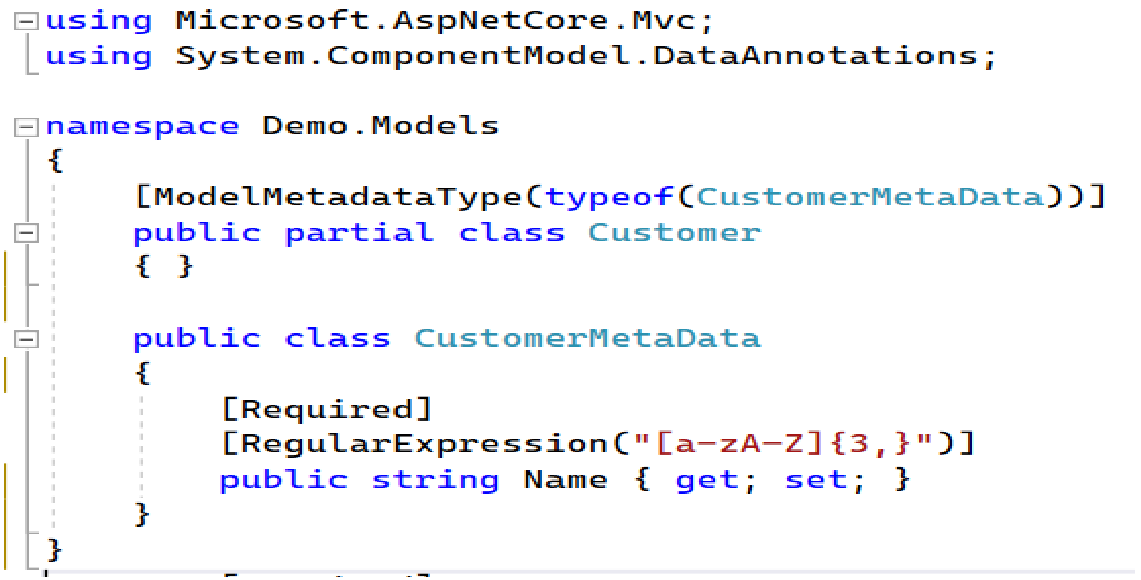
```
[MoreThanValidator(ValueToCompare = 3)]
public Nullable<int> Hours { get; set; }
```

## Validation

- How can we do validations in MVC?
  1. **Model**: By using attributes which can be applied on model properties
  2. **View** :To display the validation error message we need to use the ValidateMessageFor method which belongs to the Html helper class.
  3. **Controller**: using the ModelState.IsValid property and accordingly we can take actions
- Check whether the model is valid before updating in the action method using ModelState.IsValid.
- Enable client side validation to display error messages without postback effect in the browser by add **Unobtrusive validation script links**

## Data annotations for Database first
### (model code auto-generated)

- If you want to use the validators with the classes generated by the Entity Framework then you need to create meta data classes. You apply the validators to the meta data class instead of applying the validators to the actual class.
- Using **[ModelMetadataType (typeof(ClassName))]**



```
using Microsoft.AspNetCore.Mvc;
using System.ComponentModel.DataAnnotations;

namespace Demo.Models
{
    [ModelMetadataType(typeof(CustomerMetaData))]
    public partial class Customer
    { }

    public class CustomerMetaData
    {
        [Required]
        [RegularExpression("[a-zA-Z]{3,}")]
        public string Name { get; set; }
    }
}
```