Data Validation

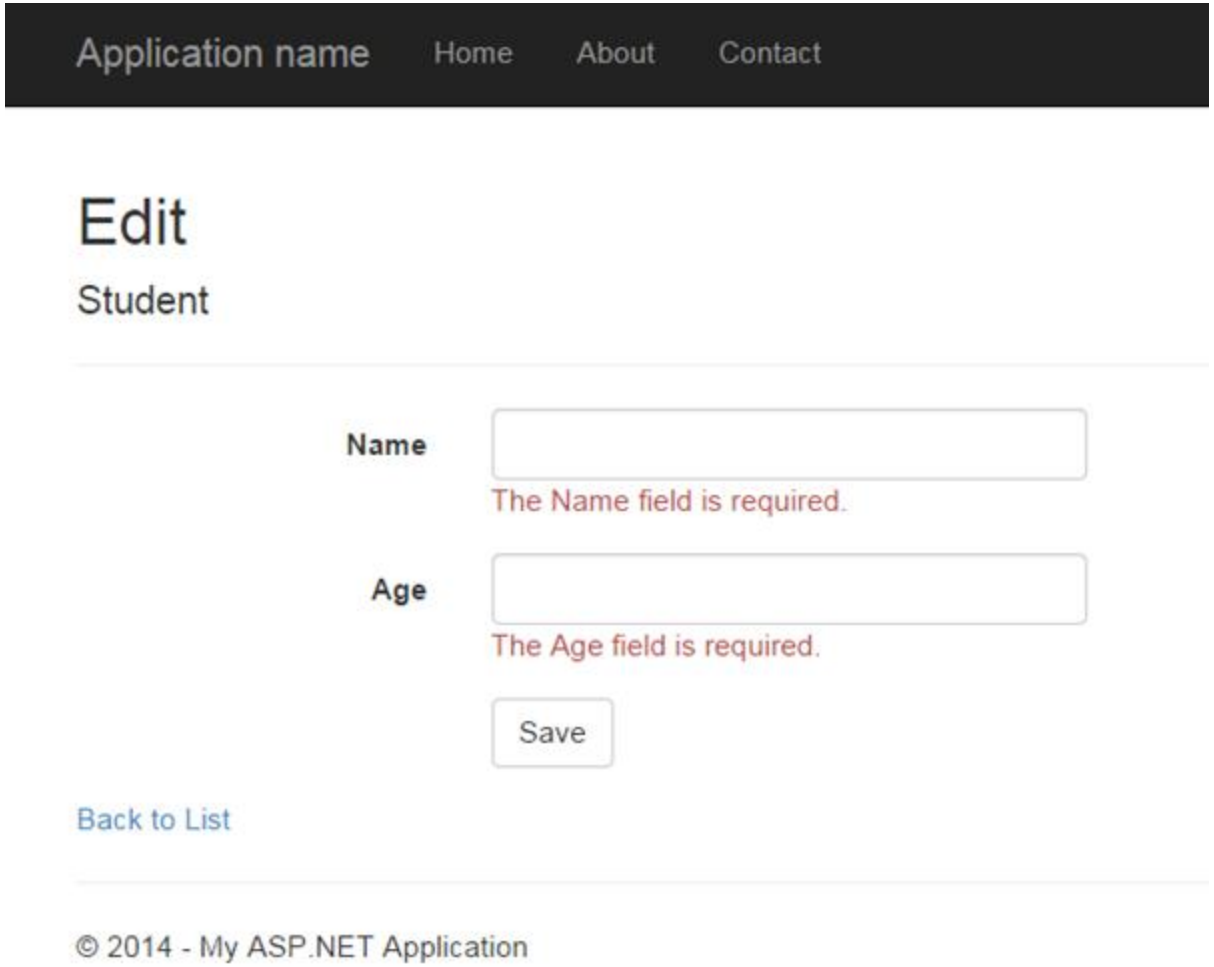
Thien Khai Tran

DataAnnotations

- ASP.NET MVC uses DataAnnotations attributes to implement validations. DataAnnotations includes built-in validation attributes for different validation rules, which can be applied to the properties of model class.
- The DataAnnotations attributes included in *System.ComponentModel.DataAnnotations* namespace

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

Apply DataAnnotation Attributes




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

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

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

Apply DataAnnotation Attributes

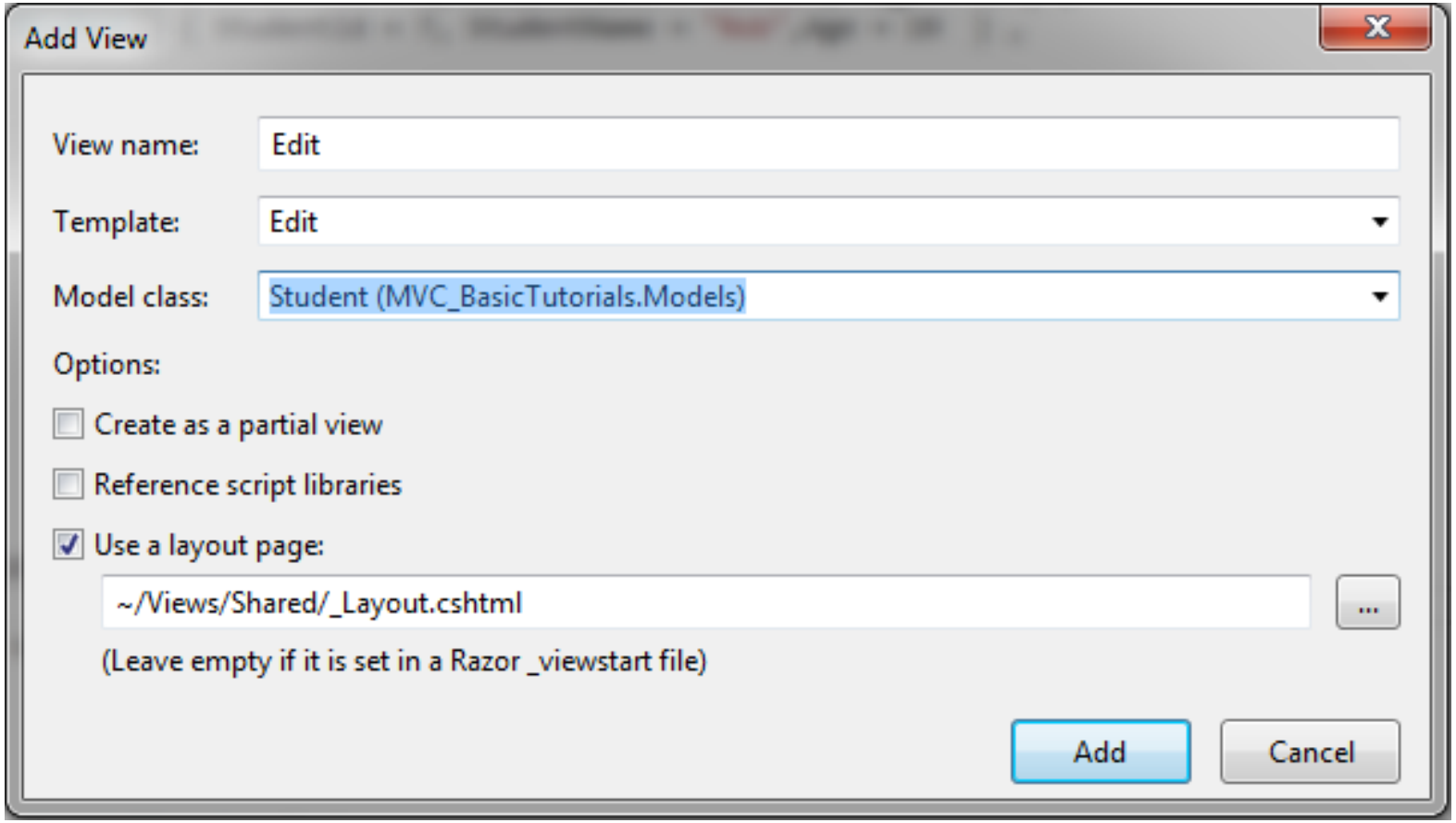
```
using MVC_BasicTutorials.Models;

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

            return View(std);
        }
```

Apply DataAnnotation Attributes

ModelState.IsValid determines that whether submitted values satisfy all the DataAnnotation validation attributes applied to model properties.

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

                //write code to update student

                return RedirectToAction("Index");
            }

            return View(std);
        }
    }
}
```

Creating an Edit view for Student.



Creating an Edit view for Student

```
@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>
```

ValidationMessage

```
@model Student

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

In the above example,

- **The first parameter** in the ValidationMessage method is a property name for which we want to show the error message e.g. StudentName.
- **The second parameter** is for custom error message and
- **The third parameter** is for html attributes like css, style etc.

ValidationMessage

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

Also, you can specify a message as a second parameter in the ValidationMessage() method as shown below.

```
@model Student

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

ValidationMessageFor

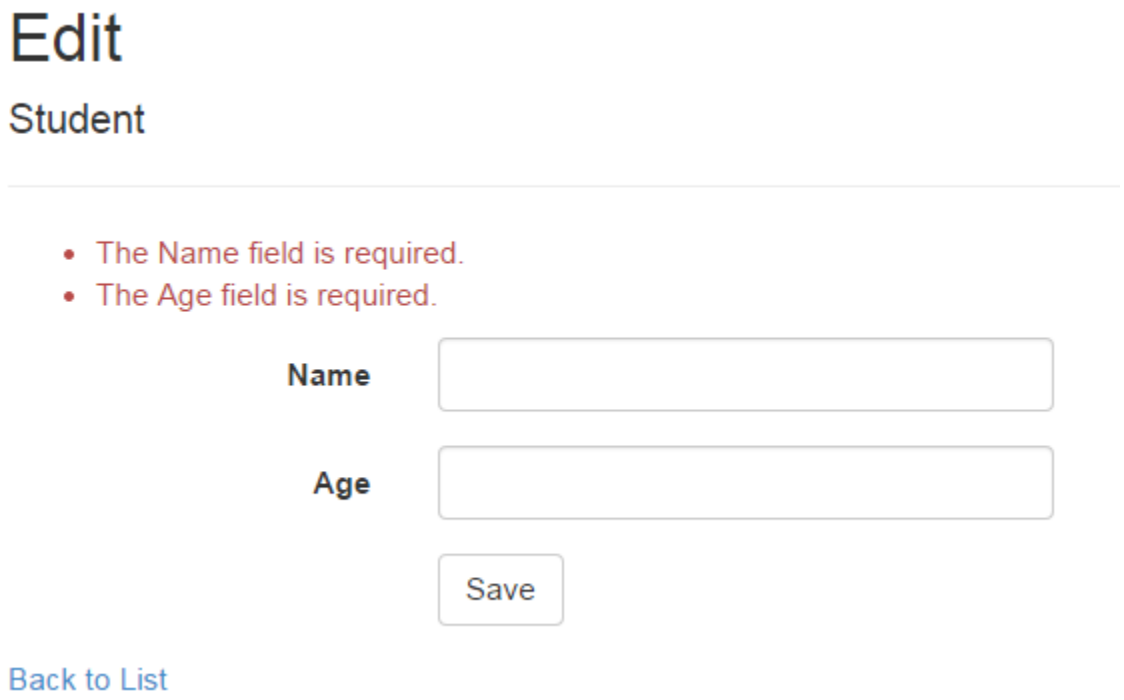
```
@model Student

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

In the above example, the **first parameter** in **ValidationMessageFor** method 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 and the **third parameter** is for html attributes like css, style etc.

ValidationSummary

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

The ValidationSummary can be used to display all the error messages for all the fields. It can also be used to display custom error messages. The following figure shows how ValidationSummary displays the error messages.




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

SignUp

UserDetails

UserName

User Name is Required

NewPassword

Password Required

Confirm new password

Date of Birth

Date Of Birth is Required

Email

Email is Required

PostalCode

550

Postal Code is Required

Phone Number

Phone Number is Required

Profile

Profile is Required

Photo

Additional Comments

City

City can only be either Hyderabad or Cyberabad

Create

Apply DataAnnotation Attributes

```
[Required(ErrorMessage = "User Name is Required")]

[StringLength(15, ErrorMessage = "User Name cannot be more than 15 characters")]

[RegularExpression("^[a-zA-Z']{1,15}$", ErrorMessage = "Invalid UserName")]

public string UserName { get; set; }


[Required(ErrorMessage = "Password Required")]

[StringLength(11, MinimumLength = 5, ErrorMessage = "Minimum Length of Password is 5 letters or Max Length is of 11 letters..")]

[DataType("password")]

public string NewPassword { get; set; }
```

Apply DataAnnotation Attributes

```
[DataType(DataType.Password)]

[Display(Name = "Confirm new password")]

[System.ComponentModel.DataAnnotations.Compare("NewPassword", ErrorMessage = "The new password and confirmation password do not match.")]

public string ConfirmPassword { get; set; }


[Required(ErrorMessage = "Date Of Birth is Required")]

[DisplayName("Date of Birth")]

[DataType(DataType.Date)]

public DateTime DateOfBirth { get; set; }
```

Apply DataAnnotation Attributes

```
[Required(ErrorMessage = "Email is Required")]
[EmailAddress(ErrorMessage = "Please enter valid Email Id")]
public string Email { get; set; }

[Required(ErrorMessage = "Postal Code is Required")]
[Range(100, 1000, ErrorMessage = "Must be between 100 and 1000")]
public int PostalCode { get; set; }

[Required(ErrorMessage = "Phone Number is Required")]
[DisplayName("Phone Number")]
public int PhoneNo { get; set; }

[Required(ErrorMessage = "Profile is Required")]
[DataType(DataType.MultilineText)]
public string Profile { get; set; }
```

Apply DataAnnotation Attributes

```
[FileExtensions(Extensions = "png,jpg,jpeg,gif")]
public string Photo { get; set; }

[AllowHtml()]
[Display(Name = "Additional Comments")]
public string AdditionalComments { get; set; }


[CustomValidation(typeof(CityValidator), "IsCityValid")]
public string City { get; set; }
```

Controller

```
[HttpGet]
public ActionResult SignUp()
{
    return View();
}
[HttpPost]
public ActionResult SignUp(UserDetails ud)
{
    if (ModelState.IsValid)
    {
        return Content("Success!");
    }
    return View(ud);
}
```

View

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

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

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

REGULAR EXPRESSION

```
Regex regex = new Regex(@"
                 ^                # anchor at the start
                (?=.*\d)          # must contain at least one numeric character
                (?=.*[a-z])       # must contain one lowercase character
                (?=.*[A-Z])       # must contain one uppercase character
                .{8,10}           # From 8 to 10 characters in length
                \s                # allows a space
                $                 # anchor at the end",
                RegexOptions.IgnorePatternWhitespace);
```

Field	Expression	Format Samples	Description
Name	^[a-zA-Z''-'\s]{1,40}$	John Doe O'Dell	Validates a name. Allows up to 40 uppercase and lowercase characters and a few special characters that are common to some names. You can modify this list.
Social Security Number	^\d{3}-\d{2}-\d{4}$	111-11-1111	Validates the format, type, and length of the supplied input field. The input must consist of 3 numeric characters followed by a dash, then 2 numeric characters followed by a dash, and then 4 numeric characters.
Phone Number	^[01]?[- .]?(\([2-9]\d{2}\)|[2-9]\d{2})[- .]?\d{3}[- .]?\d{4}$	(425) 555-0123 425-555-0123 425 555 0123 1-425-555-0123	Validates a U.S. phone number. It must consist of 3 numeric characters, optionally enclosed in parentheses, followed by a set of 3 numeric characters and then a set of 4 numeric characters.
E-mail	^(?("")(".+?""@)|(([0-9a-zA-Z]((\.(?!\.))|[-!#\$%&'*\+/=\?\^`\{\}|~\w])*)(?<=[0-9a-zA-Z])@))(?(\[)(\[(\d{1,3}\.){3}\d{1,3}\])|(([0-9a-zA-Z][-\w]*[0-9a-zA-Z]\.)+[a-zA-Z]{2,6}))$	someone@example.com	Validates an e-mail address.

URL	^(ht|f)tp(s?)\:\/\/[0-9a-zA-Z]([-.\w]*[0-9a-zA-Z])*(:(0-9)*)*(\/?)([a-zA-Z0-9\-\.\?\,\'\/\\\+&%\$#_]*)?$	http://www.mic.com	Validates a URL
ZIP Code	^(\d{5}-\d{4}|\d{5}|\d{9})$\|^([a-zA-Z]\d[a-zA-Z] \d[a-zA-Z]\d)$	12345	Validates a U.S. ZIP Code. The code must consist of 5 or 9 numeric characters.
Password	(?!^[0-9]*$)(?!^[a-zA-Z]*$)^([a-zA-Z0-9]{8,10})$		Validates a strong password. It must be between 8 and 10 characters, contain at least one digit and one alphabetic character, and must not contain special characters.
Non- negative integer	^\d+$	0 986	Validates that the field contains an integer greater than zero.
Currency (non- negative)	^\d+(\.\d\d)?$	1.00	Validates a positive currency amount. If there is a decimal point, it requires 2 numeric characters after the decimal point. For example, 3.00 is valid but 3.1 is not.
Currency (positive or negative)	^(-)?\d+(\.\d\d)?$	1.20	Validates for a positive or negative currency amount. If there is a decimal point, it requires 2 numeric characters after the decimal point.

1. ASP.NET MVC uses **DataAnnotations** attributes for validation.
2. DataAnnotations attributes can be applied to the properties of the model class to indicate the kind of value the property will hold.
3. The following validation attributes available by default
   - **Required / StringLength**
   - **Range / RegularExpression**
   - **CreditCard / CustomValidation**
   - **EmailAddress / FileExtension**
   - **MaxLength / MinLength / Phone**
4. Use **ValidationSummary** to display all the error messages in the view.
5. Use **ValidationMessageFor** or **ValidationMessage** helper method to display field level error messages in the view.
6. Check whether the model is valid before updating in the action method using **ModelState.IsValid**.

CustomValid

```
public class User
    {
        public int ID { get; set; }

        [Remote("checking","Users")]
        public string UserName { get; set; }

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


}
```

CustomValid

```
public class UsersController : Controller
    {
        private UserContext db = new UserContext();

        public JsonResult checking(string username) {
            bool has = db.Users.Any(u => u.UserName == username);
            if (has==true{
                return Json("exist",JsonRequestBehavior.AllowGet);
            }
            else{
                return Json(true, JsonRequestBehavior.AllowGet);

            }
```

?

Royal Inn and Suites

Where you're always treated like royalty

Arrival date: mm/dd/yyyy * * *

Number of nights: *

Number of adults: 3 Children: 1

Preferences

Room type: Business Suite Standard

Bed type: King Double Double

Smoking

Special requests:

Contact information

Name: *

Email: 3teo@4psoft.com

Submit Clear

Please correct the following errors:

- Arrival date is required.
- You must enter a valid date.
- Arrival date must be within 6 months of the current date.
- Number of nights is required.
- Name is required.