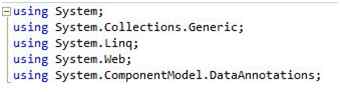
Model Validation in ASP.NET MVC

In web applications, domain validation plays a core part. Data entered from the client’s end may not always be correct. Therefore you need to ensure that the data entered by the client is not only validated but is also correct application-logic wise.  
  
This article is about the ASP.NET Model validations. The following three type of validations we can do in ASP.NET MVC web applications:

1. HTML validation / JavaScript validation
2. ASP.NET MVC Model validation
3. Database validation

But the most secure validation is the ASP.NET MVC model validation. In HTML/JavaScript, the validation can break easily, but the model validation can't. In ASP.NET MVC model validations are done using Data Annotation, its inherited *System.ComponentModel.DataAnnotations* assembly. In ASP.NET MVC 4 provides a different way to build custom validation.  
  
Before you do the ASP.NET MVC model validations you need to add the following reference:  
  
“*using System.ComponentModel.DataAnnotations;*”  
  
Now you are ready to do the model validations.  
  
In the example, create a new class called “company” under the Model add the preceding references:  
  
  
  
Now add the properties. In the following example, I have added various properties that are not directly related to the “company.”  
  
  
  


1. [Display(Name = "Company Id")]
2. **public** **int** CompanyRegisterId { **get**; **set**; }

In the preceding example you need to only add the “Required” attribute without using an error message. The default message will appear as in the following:  
  


[Display(Name = "Company Id")]

[Required (ErrorMessage= "Company Id is Required")]

**public** **int** CompanyRegisterId { **get**; **set**; }

But if you add an error message then a custom error message will appear as in the following:  
  
  
  
In HTML view you do not even need to add a label. Using ASP.NET.MVC Model we can do that easily.  
  
The only thing is, you need to call the relevant attribute in the HTML view using Razor syntax.  
  
The “ComponentModel.DataAnnotations” assembly has many built-in validation attributes, for example:

* Required
* Range,
* RegularExpression ,
* Compare
* StringLength
* Data type

There are many data types the user can select to validate the input. Using this, you can validate for the exact data type as in the following:

* Credit Card number
* Currency
* Custom
* Date
* DateTime
* Duration
* Email Address
* HTML
* Image URL
* Multiline text
* Password
* Phone number
* Postal Code
* Text
* Tine
* Upload

1. [Required (ErrorMessage="Company Email address Required")]
2. [DataType(DataType.EmailAddress)]
3. **public** **string** CompanyEmailAddress { **get**; **set**; }
5. [Required (ErrorMessage="Company Phone number is Required")]
6. [DataType(DataType.PhoneNumber)]
7. **public** **int** CompanyPhoneNumber { **get**; **set**; }
9. [Range (1,100)]
10. [DataType (DataType.Currency)]
11. **public** Decimal MinimumSalaryPerEmp { **get**; **set**; }
13. [Required]
14. [DataType (DataType.MultilineText)]
15. **public** **string** CompanyDescription { **get**; **set**; }
17. [Required]
18. [DataType (DataType.PostalCode , ErrorMessage = " Please Enter Valid Postal Code")]
19. **public** String PostalCode { **get**; **set**; }

Also, you can validate the range as in the following examples. The minimum value you can enter here is 2, whereas the maximum value you can enter is 10.

1. [Required (ErrorMessage="No.of working Hours Required")]
2. [Range(2, 10, ErrorMessage = "Please Provide correct range. It should be minimum 2 and not more than 10 ")]
3. **public** **int** WorkingHours { **get**; **set**; }

Also you can validate the string length using model validation.

1. [Required]
2. [StringLength (10 , MinimumLength =5)]
3. [Display(Name = "User name")]
4. [RegularExpression (@"(\S\D)+", ErrorMessage =" Space and numbers not allowed")]
5. **public** **string** UserName { **get**; **set**; }

In the preceding, you cannot exceed 10 letters, although it should be a minimum of 5 letters. It is also important that a regular expressions be used to do some validations. Using regular expressions I ignore empty spaces and digits that the user may enter in the text box.  
  
You can compare a user-entered password and re-enter password that are matched using this compare attribute.

1. [Required]
2. [DataType (DataType.Password)]
3. [Display (Name = "Enter Password")]
4. **public** **string** CurrentPassword { **get**; **set**; }
6. [Required]
7. [Display (Name = "Re-enter Password")]
8. [Compare("CurrentPassword" , ErrorMessage = "Please Re-enter Password Again")]
9. **public** **string** ComparedPassword { **get**; **set**; }

Using “*DisplayFormat*” you can specify the data format that should be displayed. For example, display date, currency, etc.

1. [Display(Name = "Company Founder Name")]
2. [DisplayFormat(NullDisplayText = "anonymous")]
3. **public** **string** CompanyFounder { **get**; **set**; }

Here, instead of displaying **null**, you can display **anonymous.**  
  
Now create a controller and a view to apply those validations. In this example I am using a default controller and view. The Index action returns a view that contains a form through that a user can fill in the details of the company and submit it.  
  
Sometimes a user needs to implement validation scenarios that are not provided in the MVC framework.  
  
In these kinds of scenarios you need to implement an “*IValidatableObject*” interface. Using this, you can conduct an in-depth inspection of the model.

1. **public** IEnumerable<ValidationResult> Validate(ValidationContext validationContext)
2. {
3. **if** (MinimumSalaryPerEmp < 10 && WorkingHours > 5)
4. {
5. yield **return** **new** ValidationResult("Provide Fair Amount for Workers");
6. }
7. }

In the preceding example, the method used will allow you to access all the properties in the model and you can easily implement the custom validation logic also.  
  
(Important: in the view “ValidationSummary” needs to be true.)

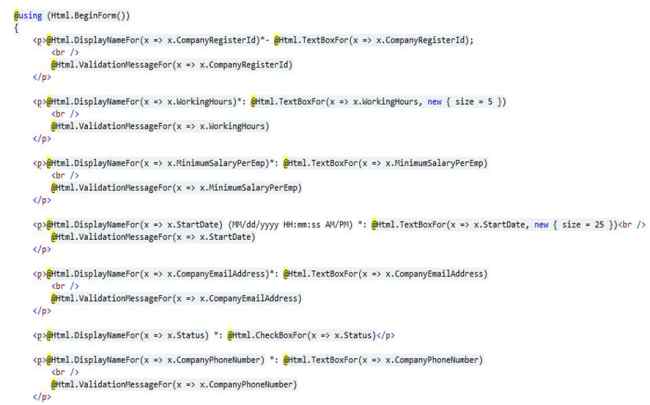
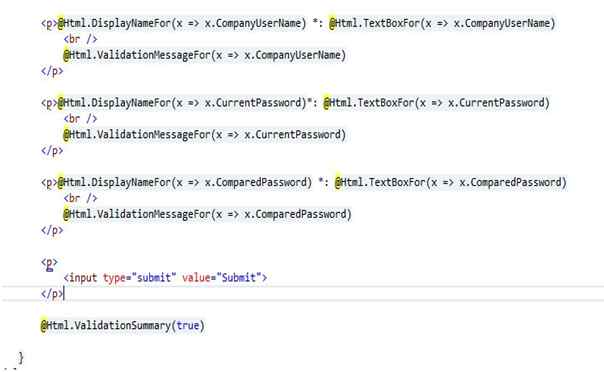
1. @Html.ValidationSummary(**true**))

When the user submits the form view, the controller will send a POST request. To respond to this request, we need to create one more ActionResult in our controller.

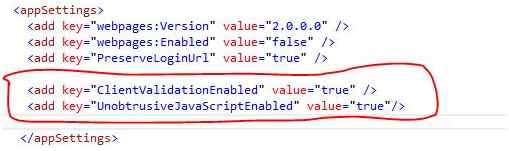
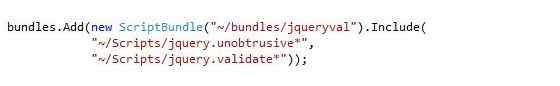
1. [HttpPost]
2. **public** ActionResult Index(Company company)
3. {
4. **if** (ModelState.IsValid)
5. {
6. // TO DO
7. **return** View();
8. }
9. **return** View();
10. }

Considering the view, you may first need to add the model reference as in the following:

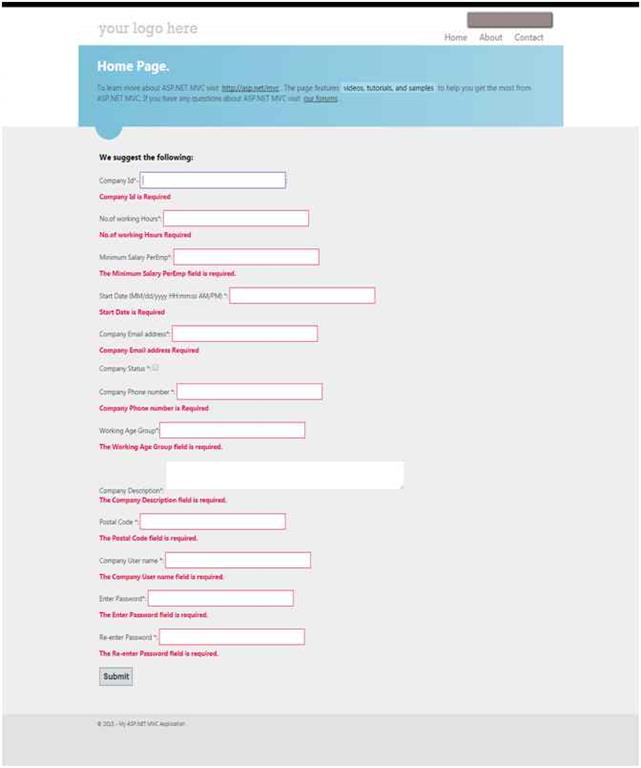
1. @model SampleModelValidation.Models.Company

Thereafter you add the form inputs and validation that you implemented in the model. It is important that the form inputs should be inside the BeginForm as shown below:  
  
  
  
  
  
  
  
Now your web application will be ready to run. Run the website, if incorrect input values are submitted you will then see invalid input error messages.  
  
This validation only applies to the server side. But there is also a need to do validation from the client side as well. When considering client-side validation, reducing the server load will be a great benefit to the user. The ASP.NET MVC Framework will support client-side validations as well.  
  
To do client-side validation, a few JavaScripts can be used. They are:

1. jQuery
2. jQuery validation library
3. jQuery unobtrusive validation library (developed by Microsoft)

When you open a new ASP.NET.MVC project you will get all the JavaScripts in the scripts folder that you want in the client-side validation. First we should set the configuration inside the <*appSettings*> tag in the *web.config* file.  
  
  
  
Now we need to add the JavaScript references inside the “*BundleConfig.cs*” file, creating a new script bundle as shown below.  
  
  
  
Then render these scripts in the “\_Layout.chtml” file as shown below.

1. @Scripts.Render("~/bundles/jqueryval")

Now you are good to go with client-side validation and server-side validation as well.  
  


# ASP.NET MVC Input Validation

Validation of user input is necessary task for the application programmer. An application should allow only valid user input so that we get only desired information.

ASP.NET MVC framework provides built-in annotations that we can apply on Model properties. It validates input and display appropriate message to the user.

## **Commonly used Validation Annotations**

|  |  |
| --- | --- |
| **Annotations** | **Description** |
| Required | It is used to make a required field. |
| DisplayName | It is used to define the text we want to display for the fields. |
| StringLength | It defines a maximum length for a string field. |
| Range | It is used to set a maximum and minimum value for a numeric field. |
| Bind | It lists fields to exclude or include when binding parameter or form values to model properties. |
| ScaffoldColumn | It allows hiding fields from editor forms. |
| MaxLength | It is used to set max length for a field. |
| EmailAddress | It is used to validate email address. |
| DataType | It is used to specify data type for the field. |
| RegularExpression | It is used to associate regular expression for a field. |

**Example**

Let's create an example that will validate input by using the annotations. To create the example, first we are creating a **StudentsController** and then a **Student** Model.

## **Controller**

### **// StudentsController.cs**

1. **using** System;
2. **using** System.Collections.Generic;
3. **using** System.Linq;
4. **using** System.Web;
5. **using** System.Web.Mvc;
6. **namespace** MvcApplicationDemo.Controllers
7. {
8. **public** **class** StudentsController : Controller
9. {
10. // GET: Students
11. **public** ActionResult Index()
12. {
13. **return** View();
14. }
15. }
16. }

## **Model**

### **// Student.cs**

1. **using** System.ComponentModel.DataAnnotations;
3. **namespace** MvcApplicationDemo.Models
4. {
5. **public** **class** Student
6. {

**public** **int** ID { **get**; **set**; }

// -- Validating Student Name

[Required(ErrorMessage ="Name is required")]

[MaxLength(12)]

**public** **string** Name { **get**; **set**; }

        // -- Validating Email Address

        [Required(ErrorMessage ="Email is required")]

        [EmailAddress(ErrorMessage = "Invalid Email Address")]

**public** **string** Email { **get**; **set**; }

        // -- Validating Contact Number

        [Required(ErrorMessage = "Contact is required")]

        [DataType(DataType.PhoneNumber)]

        [RegularExpression(@"^\(?([0-9]{3})\)?[-. ]?([0-9]{3})[-. ]?([0-9]{4})$", ErrorMessage = "Not a valid Phone number")]

**public** **string** Contact { **get**; **set**; }

1. }
2. }

## **View**

### **// Index.cshtml**

1. @model MvcApplicationDemo.Models.Student
2. @{
3. ViewBag.Title = "Index";
4. }
5. **<h2>**Index**</h2>**

@using (Html.BeginForm())

{

    @Html.AntiForgeryToken()

**<div** class="form-horizontal"**>**

**<h4>**Student**</h4>**

**<hr** **/>**

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

**<div** class="form-group"**>**

            @Html.LabelFor(model =**>** model.Name, htmlAttributes: new { @class = "control-label col-md-2" })

**<div** class="col-md-10"**>**

                @Html.EditorFor(model =**>** model.Name, new { htmlAttributes = new { @class = "form-control" } })

                @Html.ValidationMessageFor(model =**>** model.Name, "", new { @class = "text-danger" })

**</div>**

**</div>**

1. **<div** class="form-group"**>**
2. @Html.LabelFor(model =**>** model.Email, htmlAttributes: new { @class = "control-label col-md-2" })
3. **<div** class="col-md-10"**>**
4. @Html.EditorFor(model =**>** model.Email, new { htmlAttributes = new { @class = "form-control" } })
5. @Html.ValidationMessageFor(model =**>** model.Email, "", new { @class = "text-danger" })
6. **</div>**
7. **</div>**
8. **<div** class="form-group"**>**
9. @Html.LabelFor(model =**>** model.Contact, htmlAttributes: new { @class = "control-label col-md-2" })
10. **<div** class="col-md-10"**>**
11. @Html.EditorFor(model =**>** model.Contact, new { htmlAttributes = new { @class = "form-control" } })
12. @Html.ValidationMessageFor(model =**>** model.Contact, "", new { @class = "text-danger" })
13. **</div>**
14. **</div>**
15. **<div** class="form-group"**>**
16. **<div** class="col-md-offset-2 col-md-10"**>**
17. **<input** type="submit" value="Create" class="btn btn-default" **/>**
18. **</div>**
19. **</div>**
20. **</div>**

}

**<div>**

    @Html.ActionLink("Back to List", "Index")

**</div>**

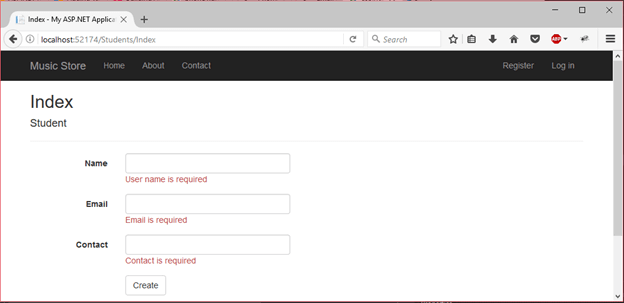
@section Scripts {

    @Scripts.Render("~/bundles/jqueryval")

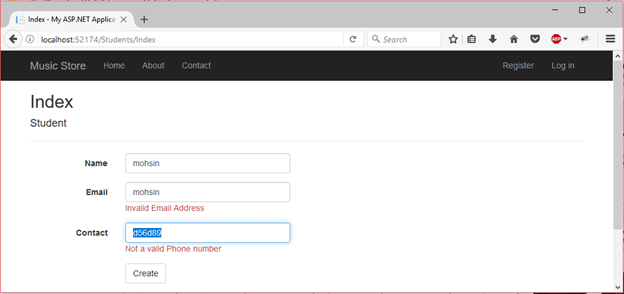
}

Output:

To see the output, right click on the **Index.cshtml** file and select view in browser. This will produce the following result.



As we can see that it validates form fields and display error message to the user. In the below screenshot, we are validating that the entered data is as expected.



Implement Data Validation in MVC

Here, you will learn how to implement the data validation and display validation messages on the violation of business rules in an ASP.NET MVC application.

The following image shows how the validation messages will be displayed if Name or Age fields are blank while creating or editing data.

Validation using Data Annotation Attributes

ASP.NET MVC includes built-in attribute classes in the [System.ComponentModel.DataAnnotations](https://docs.microsoft.com/en-us/dotnet/api/system.componentmodel.dataannotations?view=netframework-4.8" \t "_blank) namespace. These attributes are used to define metadata for ASP.NET MVC and ASP.NET data controls. You can apply these attributes to the properties of the model class to display appropriate validation messages to the users.

The following table lists all the data annotation attributes which can be used for validation.

| Attribute | Usage |
| --- | --- |
| Required | Specifies that a property value is required. |
| StringLength | Specifies the minimum and maximum length of characters that are allowed in a string type property. |
| Range | Specifies the numeric range constraints for the value of a property. |
| RegularExpression | Specifies that a property value must match the specified regular expression. |
| CreditCard | Specifies that a property value is a credit card number. |
| CustomValidation | Specifies a custom validation method that is used to validate a property. |
| EmailAddress | Validates an email address. |
| FileExtension | Validates file name extensions. |
| MaxLength | Specifies the maximum length of array or string data allowed in a property. |
| MinLength | Specifies the minimum length of array or string data allowed in a property. |
| Phone | Specifies that a property value is a well-formed phone number. |

Let's see how to use these attributes to display validation messages on the view.

The following is the Student model class.

Example: Apply DataAnnotation Attributes

 Copy

public class Student

{

public int StudentId { get; set; }

public string StudentName { get; set; }

public int Age { get; set; }

}

We want to implement validations for StudentName and Age property values. We want to make sure that users do not save empty StudentName or Age value. Also, age should be between 10 to 20.

The Required attribute is used to specify that the value cannot be empty. The Range attribute is used to specify the range of values a property can have. We will use the Required attribute on the StudentName to make it mandatory for the user to provide value and Range attribute to make sure the user enters value between 10 to 20, as shown below.

Example: Apply DataAnnotation Attributes

 Copy

public class Student

{

public int StudentId { get; set; }

[Required]

public string StudentName { get; set; }

[Range(10, 20)]

public int Age { get; set; }

}

The above attributes define the metadata for the validations of the Student class. This alone is not enough for the validation. You need to check whether the submitted data is valid or not in the controller. In other words, you need to check the model state.

ADVERTISEMENT

Use the ModelState.IsValid to check whether the submitted model object satisfies the requirement specified by all the data annotation attributes. The following POST action method checks the model state before saving data.

Example: Edit Action methods:

 Copy

public class StudentController : Controller

{

public ActionResult Edit(int id)

{

var stud = ... get the data from the DB using Entity Framework

return View(stud);

}

[HttpPost]

public ActionResult Edit(Student std)

{

if (ModelState.IsValid) { //checking model state

//update student to db

return RedirectToAction("Index");

}

return View(std);

}

}

Now, create an edit view as shown [here](https://www.tutorialsteacher.com/mvc/create-edit-view-in-asp.net-mvc). The following is a generated edit view using the default scaffolding template.

@model MVC\_BasicTutorials.Models.Student

@{

ViewBag.Title = "Edit";

Layout = "~/Views/Shared/\_Layout.cshtml";

}

<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>

}

<div>

@Html.ActionLink("Back to List", "Index")

</div>

In the above view, it calls the HTML Helper method **ValidationMessageFor()** for every field and **ValidationSummary()** method at the top. The [ValidationMessageFor()](https://www.tutorialsteacher.com/mvc/htmlhelper-validationmessagefor) displays an error message for the specified field. The [ValidationSummary()](https://www.tutorialsteacher.com/mvc/htmlhelper-validationsummary) displays a list of all the error messages for all the fields.

In this way, you can display the default validation message when you submit a form without entering StudentName or Age, as shown below.