Data Validation and Annotation

Nội dung

- Define and describe how to validate data
- Explain how to use data annotation
- Explain and describe how to use ModelState

Validation

- In an ASP.NET MVC application, users interact with the application in the following ways:
  - Typing a URL on the browser.
  - Clicking a link on the application.
  - Submitting a form provided by the application as a view.
- In all the preceding interactions:
  - You as a developer need to ensure that any data sent by a user to the application is valid.
  - You can validate user data by including validation logic in your application.
  - The validation logic should first analyze whether or not the user specified data is valid and as expected by the application.
  - The validation logic should accordingly provide feedback to the user so that the user can identify and rectify any invalid input before they resubmit the data.

Data validation workflow

- The MVC Framework implements a validation workflow to validate user data.
- The validation workflow starts when a user specified data arrives in the server.
- The validation process:
  - First checks whether the request is some form of attack, such as Cross Site Scripting (CSS).
  - If the process identifies a request as an attack, it throws an exception.
  - Else, the process checks the request data against the validation requirements of a application.
  - If all data meets the validation requirements, the process forwards the request to the application for further processing.
  - If one or more data does not meet the validation requirements, the process sends back a validation error message as a response.

Kiểm soát dữ liệu thủ công

- ♦ In an ASP.NET MVC application, you can manually validate data in the controller action that accepts user data for some kind of processing.
- ♦ To manually validate data, you can implement simple validation routines, such as a routine to check that the length of a password submitted through a registration form exceeds more than seven characters.

Kiểm soát dữ liệu thủ công

```
public class HomeController : Controller {
Public ActionResult Index() {
        return View();
}
[HttpPost]
Public ActionResult Index(User model) {
        String modelPassword = model.password;
        if (modelPassword.Length< 7) {
                return View();
        }
        else {
        /*Implement registration process*/
                return Content("You have successfully registered");
        }
}
}
```

Kiểm soát dữ liệu thủ công

- The preceding code:
  - Creates a HomeControllercontroller class with two Index()action methods.
  - The first Index() method returns the corresponding view.
  - When a user submits a form displayed by the view the HttpPostversion of the action method receives a Usermodel object that represents the user submitted data.
  - This action method validates whether or not the length of the password property is greater than seven.
  - If the password length is less than seven, the action method returns back the view, else the action method returns a success message.
  - You can also use regular expression while performing manual validations.
  - For example, you can use a regular expression to check whether the user submitted e-mail id is in the correct format, such as abc@abc.com.

Kiểm soát dữ liệu thủ công

```
public class HomeController : Controller {
public ActionResult Index() {
        return View();
}
[HttpPost]
public ActionResult Index(User model) {
        string modelEmailId = model.email;
        string regexPattern=@"[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,4}";
        if (System.Text.RegularExpressions.Regex.IsMatch(modelEmailId,
                regexPattern)) {
                /*Implement registration process*/
                return Content("You have successfully registered"); }
        else {
                return View(); }
}
}
```

Kiểm soát dữ liệu thủ công

- The preceding code:
  - Uses the regexPattern variable to store a regular expression that specifies a valid email id format.
  - Then, uses the IsMatch() method of the System.Text.RegularExpressions.Regexclass to validate whether or not the email property is in valid format. If the value of the email property is valid, the action method returns back the view else the action method returns a success message.

Annotation

- The MVC Framework provides several data annotations that you can apply as attributes to a model.
- These data annotations implement tasks that are commonly required across applications.
- Some of the important annotations that you can use in the models of an ASP.NET MVC application are as follows:
  - Required
  - StringLength
  - RegularExpression
  - Range
  - Compare
  - DisplayName
  - ReadOnly
  - DataType
  - ScaffoldColumn

Required annotation

- The Required data annotation specifies that the property, with which this annotation is associated, is a required property.
- This attribute raises a validation error if the property value is null or empty.
- Following is the syntax of the Required data annotation:

```
[Required]
public string <property_name>;
```

- where,
  - property_name: Is the name of a model property.

Required annotation

- Following code snippet shows using the Required annotation in the properties of a User model:

```
public class User {
        public long Id { get; set; }
        [Required]
        public string Name { get; set; }
        [Required]
        public string Password { get; set; }
        [Required]
        public string ReenterPassword { get; set; }
        [Required]
        public int Age { get; set; }
        [Required]
        public string Email { get; set; } }
```

- In this code, the Required attribute is specified for the Name, Password, ReenterPassword, Age, and Email properties of the Usermodel.

Required annotation

- Once you use the Required attributes in the model properties, you should:
- Use the Html.ValidationSummary() helper method passing true as the parameter.
- This method is used to display validation messages on the page.
- Use the Html.ValidationMessageFor() helper method passing the lambda expression to associate the method with a model property for each field.
- This method returns the validation error message for the associated property as HTML.

Required annotation

♦ Following code shows the view that uses helper methods to display validation messages:

```
@model MVCModelDemo.Models.User
@{ViewBag.Title = “User Form Validation”;}
<h2>User Form</h2>
@using (Html.BeginForm()) {
	@Html.ValidationSummary(true)
	<div>@Html.LabelFor(model =>model.Name)</div>
	<div>@Html.EditorFor(model =>model.Name)
		@Html.ValidationMessageFor(model =>model.Name)</div>
	<div>@Html.LabelFor(model =>model.Password)</div>
	<div>@Html.EditorFor(model =>model.Password)
		@Html.ValidationMessageFor(model =>model.Password)</div>
	<div>@Html.LabelFor(model =>model.ReenterPassword)</div>
	<div>@Html.EditorFor(model =>model.ReenterPassword)
	@Html.ValidationMessageFor(model =>model.ReenterPassword)</div>
```

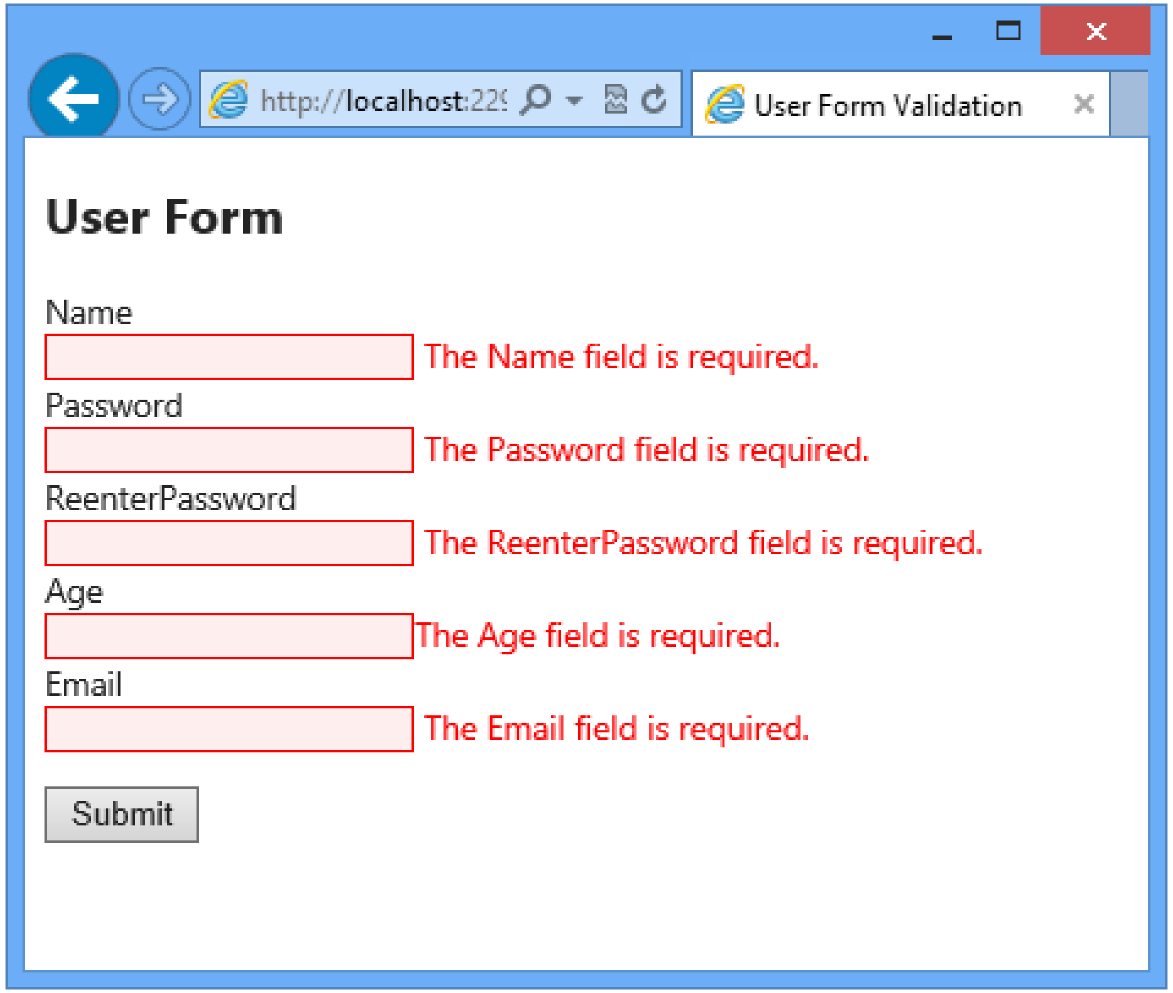
Required annotation

```
    <div>@Html.LabelFor(model =>model.Age)</div>
    <div>@Html.EditorFor(model =>model.Age)
        @Html.ValidationMessageFor(model =>model.Age)</div>
    <div>@Html.LabelFor(model =>model.Email)</div>
    <div>@Html.EditorFor(model =>model.Email)
        @Html.ValidationMessageFor(model =>model.Email)</div>
    <p>
        <input type="submit" value="Submit" />
    </p>
}
```

- In the preceding code:
  - The Html.ValidationSummary(true) helper method displays validation messages as a list.
  - Then, for each UI field, the Html.ValidationMessageFor() helper method is used to validate the corresponding property of the model.

Required annotation

♦ Following figure shows the default validation error messages when a user clicks the Submitbutton without specifying any values in the fields:



User Form

Name

The Name field is required.

Password

The Password field is required.

ReenterPassword

The ReenterPassword field is required.

Age

The Age field is required.

Email

The Email field is required.

Required annotation

- You can also specify a custom validation message for the Requiredannotation.
- The syntax to specify a custom validation message for the Requiredannotation is as follows:

```
[Required(ErrorMessage = <error-message>)]
```

- where,
  - error-message: Is the custom error message that you want to display

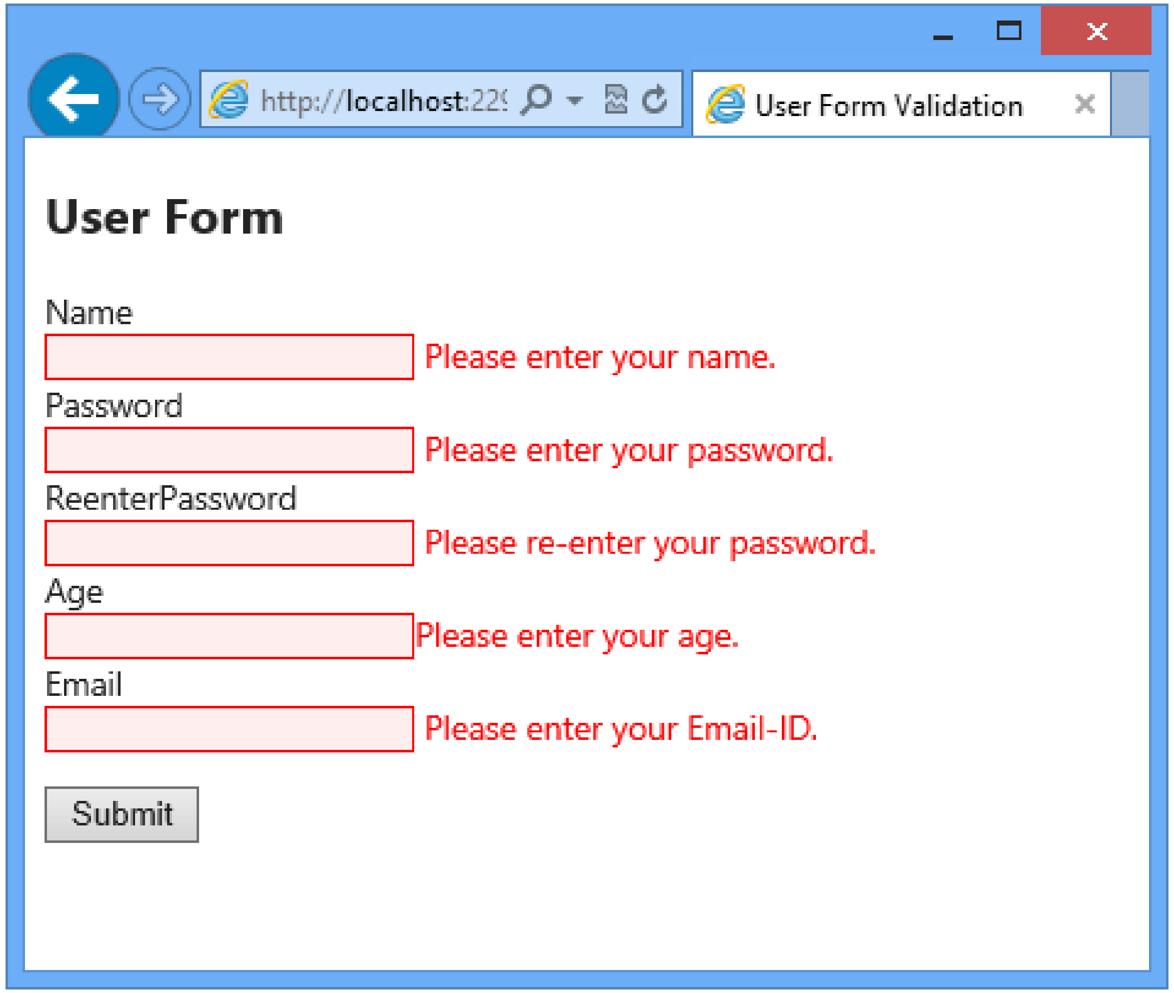
Required annotation

♦ Following code snippet shows a User model with the Required annotations with custom validation messages applied to its properties:

```
public class User {
        public long Id { get; set; }
        [Required(ErrorMessage = "Please enter your name.")]
        public string Name { get; set; }
        [Required(ErrorMessage = "Please enter your password.")]
        public string Password { get; set; }
        [Required(ErrorMessage = "Please re-enter your password.")]
        public string ReenterPassword { get; set; }
        [Required (ErrorMessage = "Please enter your age.")]
        public int Age { get; set; }
        [Required(ErrorMessage = "Please enter your Email-ID.")]
        public string Email { get; set; }
}
```

Required annotation

- Following figure shows the custom validation messages when a user tries to submit the form without specifying any values:



User Form

Name

Please enter your name.

Password

Please enter your password.

ReenterPassword

Please re-enter your password.

Age

Please enter your age.

Email

Please enter your Email-ID.

StringLength annotation

- You can use the StringLength annotation to specify the minimum and maximum lengths of a string field.
- Following is the syntax for StringLength annotation:

```
[StringLength(<max_length>, MinimumLength = <min_length>)]
```

- where,
  - max_length: Is an integer value that specifies the maximum allowed length.
  - min_length: Is an integer value that specifies the minimum allowed length.

StringLength annotation

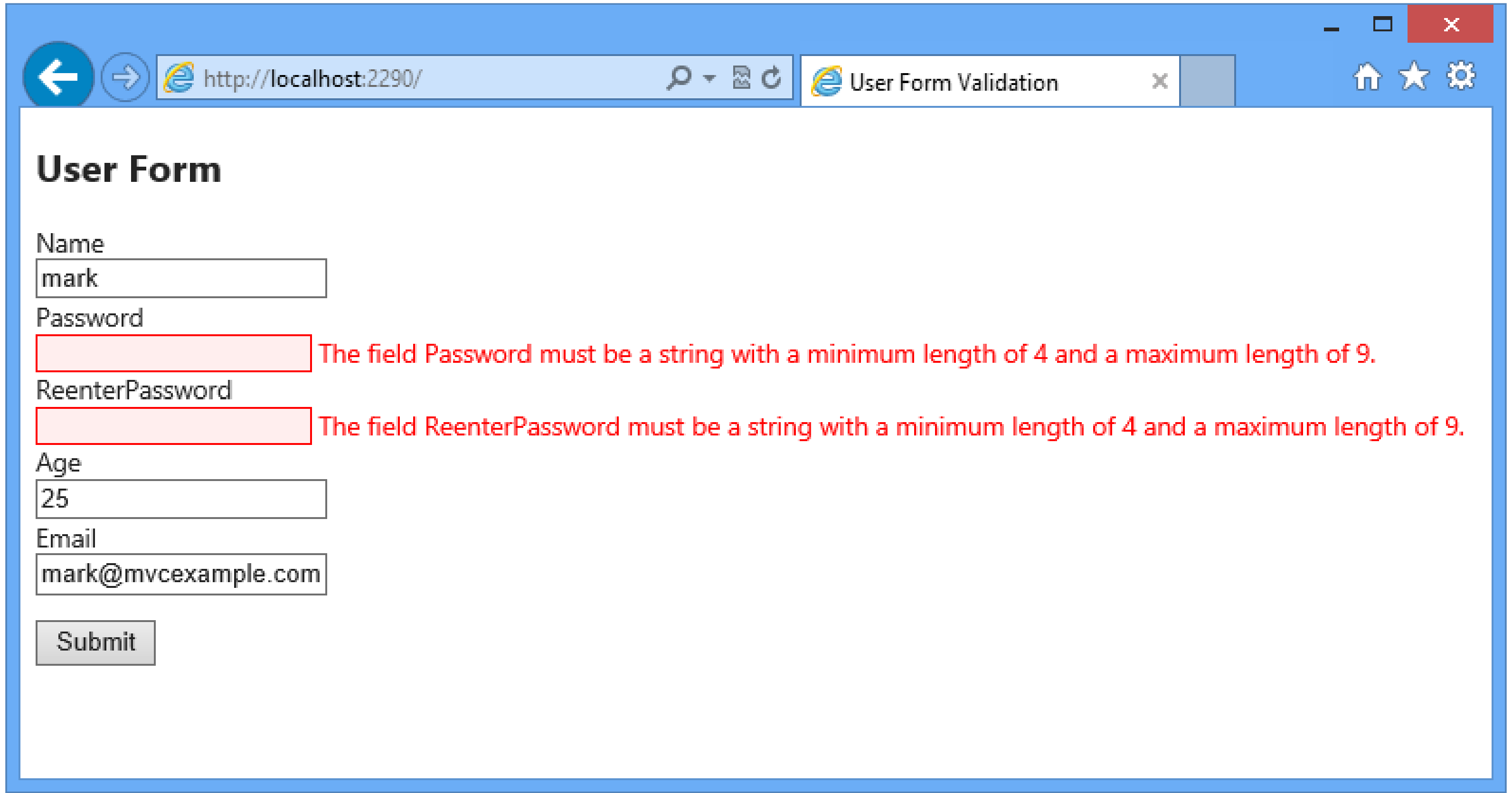
- Following code shows a User model with the StringLength annotations applied to its Password and ReenterPassword properties:

```
public class User { public long Id { get; set; }
        [Required(ErrorMessage = "Please enter your name.")]
        public string Name { get; set; }
        [StringLength(9, MinimumLength = 4)]
        [DataType(DataType.Password)]
        public string Password { get; set; }
        [StringLength(9, MinimumLength = 4)]
        [DataType(DataType.Password)]
        public string ReenterPassword { get; set; }
        [Required (ErrorMessage = "Please enter your age.")]
        public int Age { get; set; }
        [Required(ErrorMessage = "Please enter your Email-ID.")]
        public string Email { get; set; } }
```

- The [StringLength] annotation specifies that the maximum length of the Password and ReenterPassword properties is 9 and the minimum length is 4.

StringLength annotation

- Whenever a user specified values for these fields are out of this specified range a validation error message is displayed
- Following figure shows the validation messages when a user specified value in the Passwordand ReenterPasswordfields is out of the specified range:



User Form

Name
mark
Password
The field Password must be a string with a minimum length of 4 and a maximum length of 9.
ReenterPassword
The field ReenterPassword must be a string with a minimum length of 4 and a maximum length of 9.
Age
25
Email
mark@mvcexample.com

RegularExpression Annotation

- You can use the RegularExpression annotation to accept user input in a specific format.
- This annotation allows you to match a text string with a search pattern that contains one or more character literals, operators, or constructs.
- Following is the syntax for RegularExpression annotation:

```
[RegularExpression(<pattern>)]
```

- where,
  - <pattern>: is the specified format according to which you want user input.

RegularExpression Annotation

♦ Following code shows a User model with the RegularExpression annotations applied to its Email property:

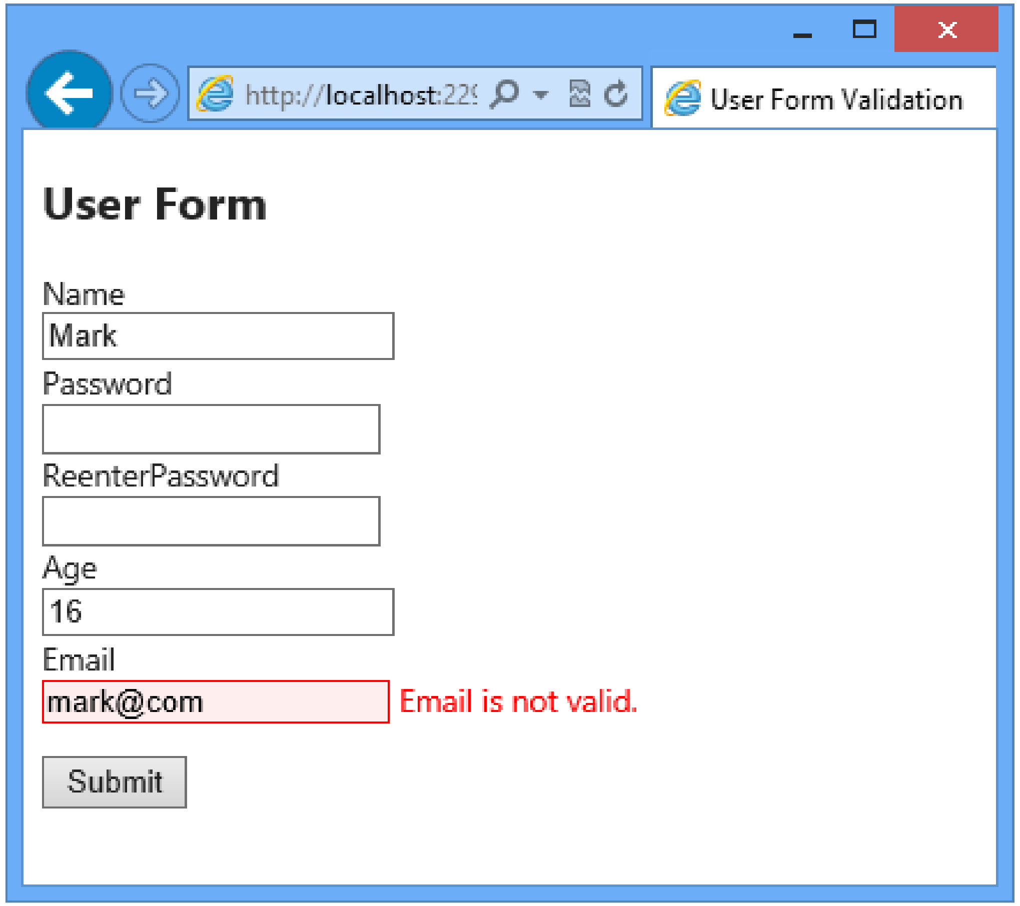
```
public class User {
        public long Id { get; set; }
        [Required(ErrorMessage = "Please enter your name.")]
        public string Name { get; set; }
        [StringLength(9, MinimumLength = 4)]
        public string Password { get; set; }
        [StringLength(9, MinimumLength = 4)]
        public string ReenterPassword { get; set; }
        [Required (ErrorMessage = "Please enter your age.")]
        public int Age { get; set; }
        [RegularExpression(@"[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,4}",
        ErrorMessage = "Email is not valid.")]
        public string Email { get; set; }
}
```

RegularExpression Annotation

- In the preceding code:
  - The regular expression A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,4} defines the format of an e-mail address. It is divided in three parts where:
  - Part 1:[A-Za-z0-9._%+-]+
  - Part 2:[A-Za-z0-9.-]+
  - Part 3:[A-Za-z]{2,4}
- The first part of the regular expression specifies the characters and it ranges within square brackets that can appear.
- Then, the + sign between the first and second part indicates that these parts can consist of one or more characters of the types specified within the square brackets preceding the + sign.
- The third part contains {2,4}at the end indicating that this part can include 2-4 characters

RegularExpression Annotation

- Following figure shows the validation message that is displayed when a user specified value in the Email field is not in a valid format as specified using regular expression:



http://localhost:22

User Form Validation

User Form

Name

Mark

Password

ReenterPassword

Age

16

Email

mark@com Email is not valid.

Submit

Range Annotation

- You can use the Range annotation to specify the minimum and maximum constraints for a numeric value.
- Following is the syntax of the Range annotation:

```
[Range (<minimum_range>, <maximum_range>)]
```

- where,
  - minimum_range: Is a numeric value that specifies the minimum value for the range.
  - maximum_range: Is a numeric value that specifies the minimum value for the range..

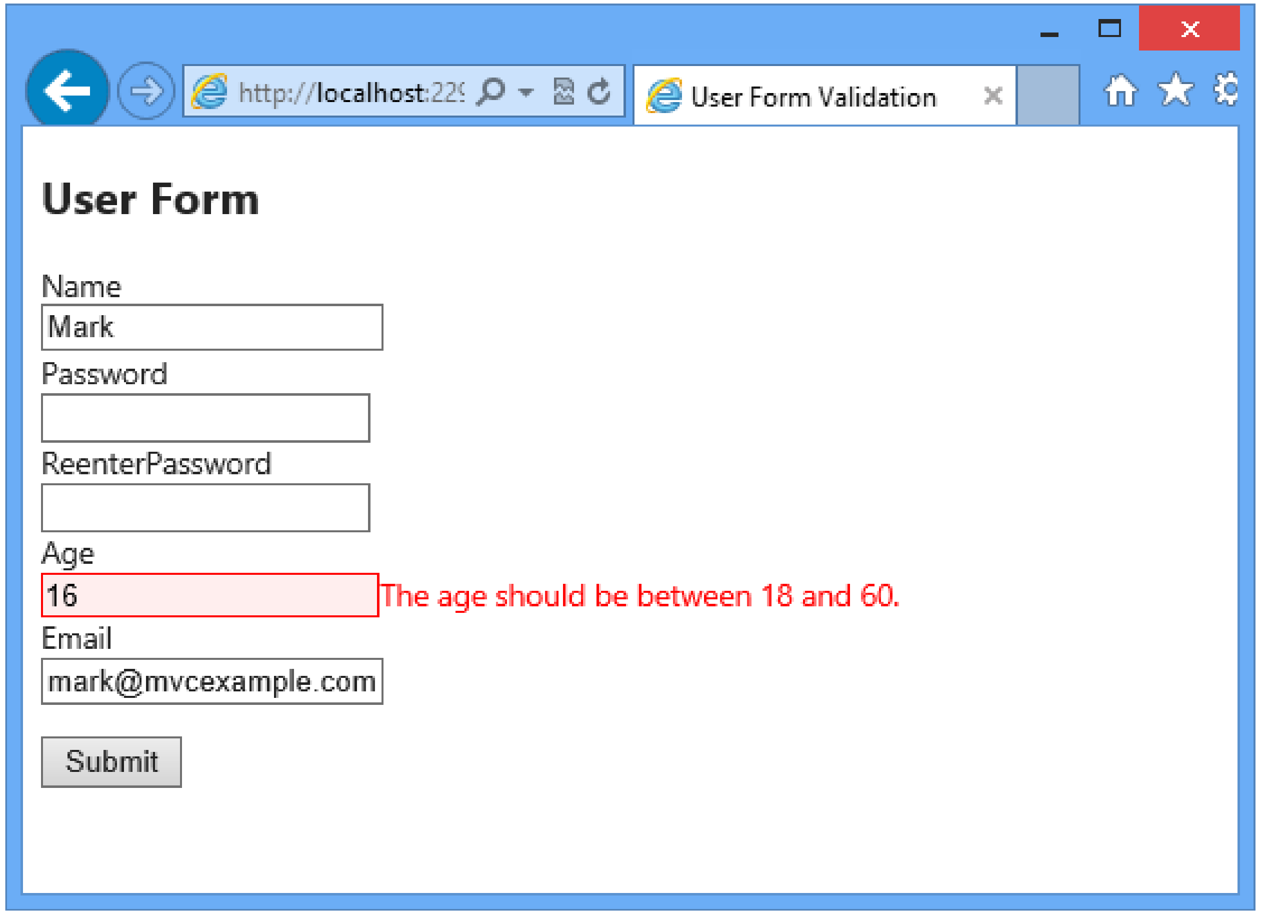
Range Annotation

- Following code shows a User model with the Range annotations applied to its Age property:

```
public class User {
        public long Id { get; set; }
        [Required(ErrorMessage = "Please enter your name.")]
        public string Name { get; set; }
        [StringLength(9, MinimumLength = 4)]
        public string Password { get; set; }
        [StringLength(9, MinimumLength = 4)]
        public string ReenterPassword { get; set; }
        [Range(18, 60, ErrorMessage = "The age should be between 18 and 60.")]
        public int Age { get; set; }
        [RegularExpression(@"[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,4}",
        ErrorMessage = "Email is not valid.")]
        public string Email { get; set; }
}
```

Range Annotation

- Following figure shows the validation error message that is displayed when a user specified age is not in the specified range:



User Form

Name
Mark
Password

ReenterPassword

Age
16 The age should be between 18 and 60.
Email
mark@mvcexample.com

Compare Annotation

- You can use the Compare annotation to compare values in two fields.
- The Compare annotation ensures that the two properties on a model object have the same value.
- Following code snippet shows a User model with the Compare annotation applied to its ReenterPassword property:

```
[Compare("Password", ErrorMessage = "The specified passwords do not
match with the Password field.")]
```

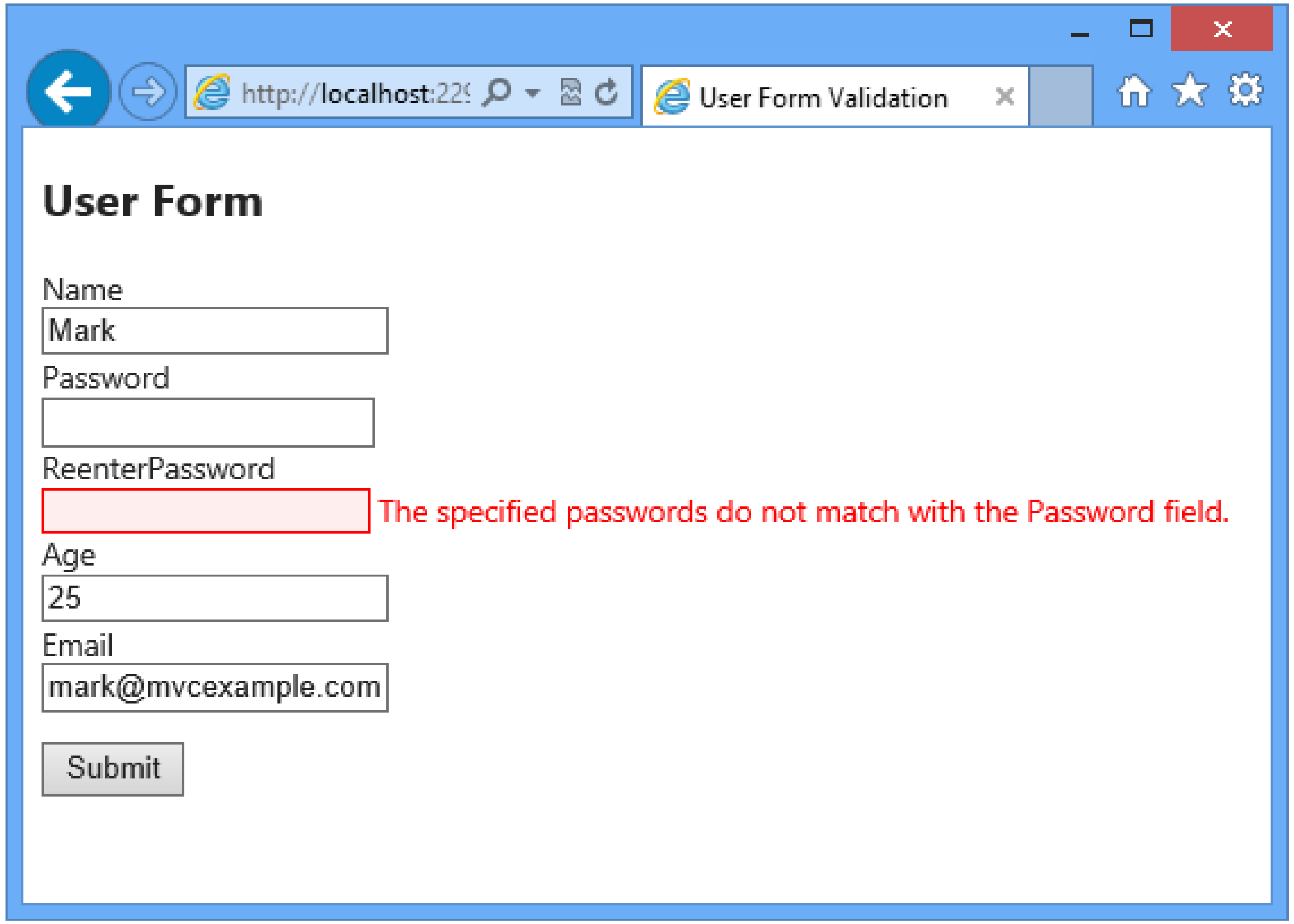
Compare Annotation

- This code uses the Compare annotation to check that the ReenterPassword field contains the same values as that of the Passwordfield.:

```
public class User {
        public long Id { get; set; }
        [Required(ErrorMessage = "Please enter your name.")]
        public string Name { get; set; }
        [StringLength(9, MinimumLength = 4)]
        public string Password { get; set; }
        [StringLength(9, MinimumLength = 4)]
        [Compare("Password", ErrorMessage = "The specified passwords do not
        match with the Password field.")]
        public string ReenterPassword { get; set; }
        [Range(18, 60, ErrorMessage = "The age should be between 18 and 60.")]
        public int Age { get; set; }
        [RegularExpression(@"[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,4}",
        ErrorMessage = "Email is not valid.")]
        public string Email { get; set; }
}
```

Compare Annotation

- Following figure shows the validation error message that is displayed when a user does not enter the same value in both the Password and ReenterPassword fields:



http://localhost:22

User Form Validation

User Form

Name

Mark

Password

ReenterPassword

The specified passwords do not match with the Password field.

Age

25

Email

mark@mvcexample.com

Submit

DisplayName Annotation

- When you use the @Html.LabelFor() helper method in a strongly typed view, the method displays a label with a text whose value is the corresponding property name.
- You can also explicitly state the text that the @Html.LabelFor() method should display using the DisplayName annotation on the model property.
- Following is the syntax for DisplayName annotation:

```
[DisplayName(<text>)]
```

- where,
  - text: Is the text to be displayed for the property.

DisplayName Annotation

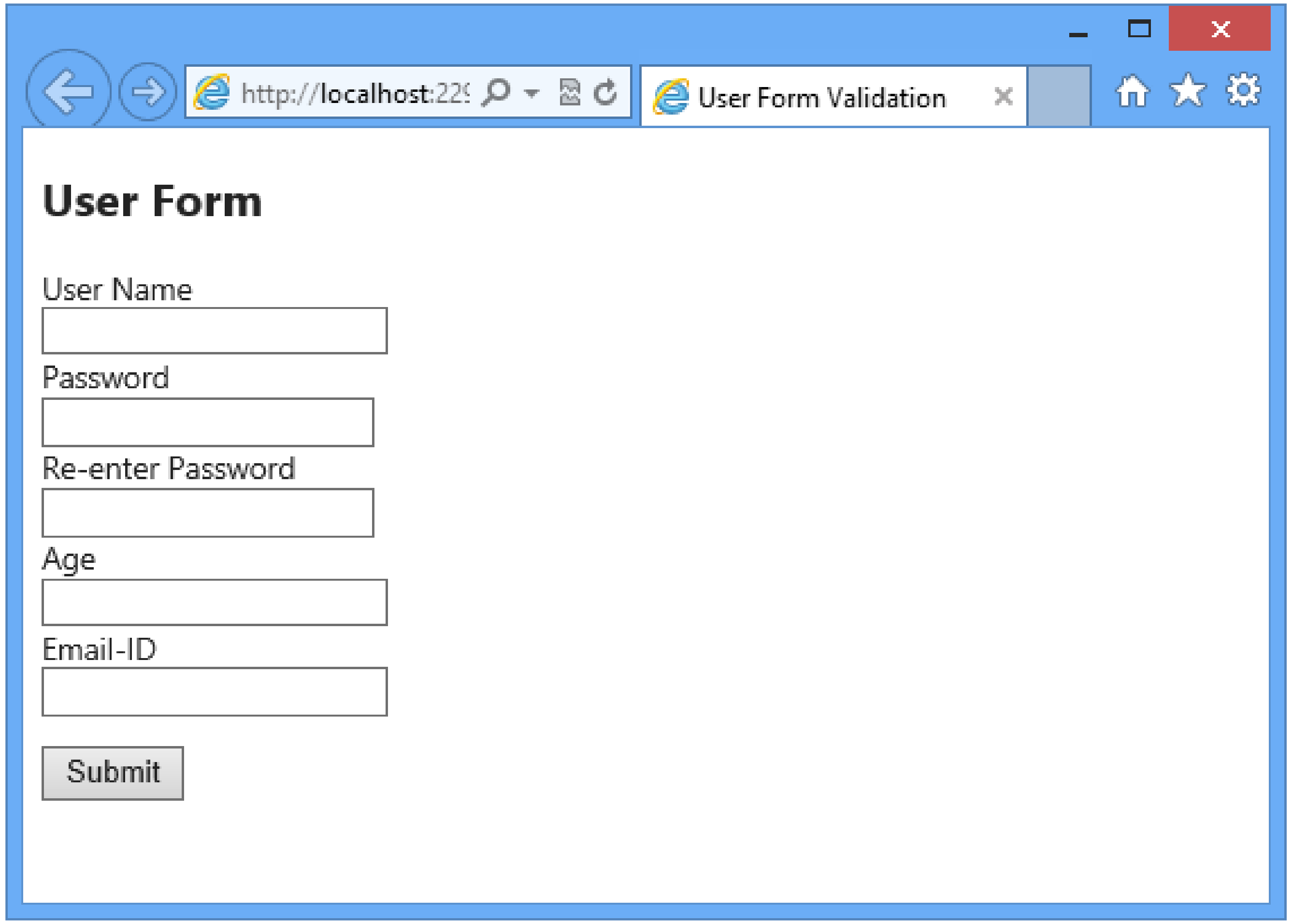
- Following code shows a User model with the DisplayName annotation applied to its Name, ReenterPassword, and Email properties:

```
public class User {
        public long Id { get; set; }
        [DisplayName("User Name")]
        public string Name { get; set; }
        public string Password { get; set; }
        [DisplayName("Re-enter Password")]
        public string ReenterPassword { get; set; }
        public int Age { get; set; }
        [DisplayName("Email-ID")]
        public string Email { get; set; }
}
```

- This code applies the DisplayName annotation to the Name, ReenterPassword, and Email properties.

DisplayName Annotation

- Following figure shows the user friendly display name for the Name, ReenterPassword, and Email model properties:


http://localhost:229

User Form Validation

User Form

User Name

Password

Re-enter Password

Age

Email-ID

Submit

ReadOnly Annotation

- You can use the ReadOnly annotation to display read-only fields on a form.
- You can use this annotation to instruct the default model binder not to set the DiscountedAmount property with a new value from the request.
- Following is the syntax for ReadOnly annotation:

```
[ReadOnly(<boolean_value>)]
```

- where,
    - boolean_value: Is a boolean value which can be either true or false. A true value indicates that the default model binder should not set a new value for the property. A false value indicates that the default model binder should set a new value for the property based on the request.

ReadOnly Annotation

- Following code snippet shows how to use the ReadOnly annotation:

```
        [ReadOnly(true)]
        public intDiscountedAmount { get; set; }
}
```

- This code uses the ReadOnly annotation passing true as the value for the DiscountedAmount property.

DataType Annotation

- You can use the DataType annotation to provide information about the specific purpose of a property at run time.
- Following is the syntax for DataType annotation:

```
[DataType(DataType.<value>)]
```

- where,
  - value: Is a member of DataType enumeration.

DataType Annotation

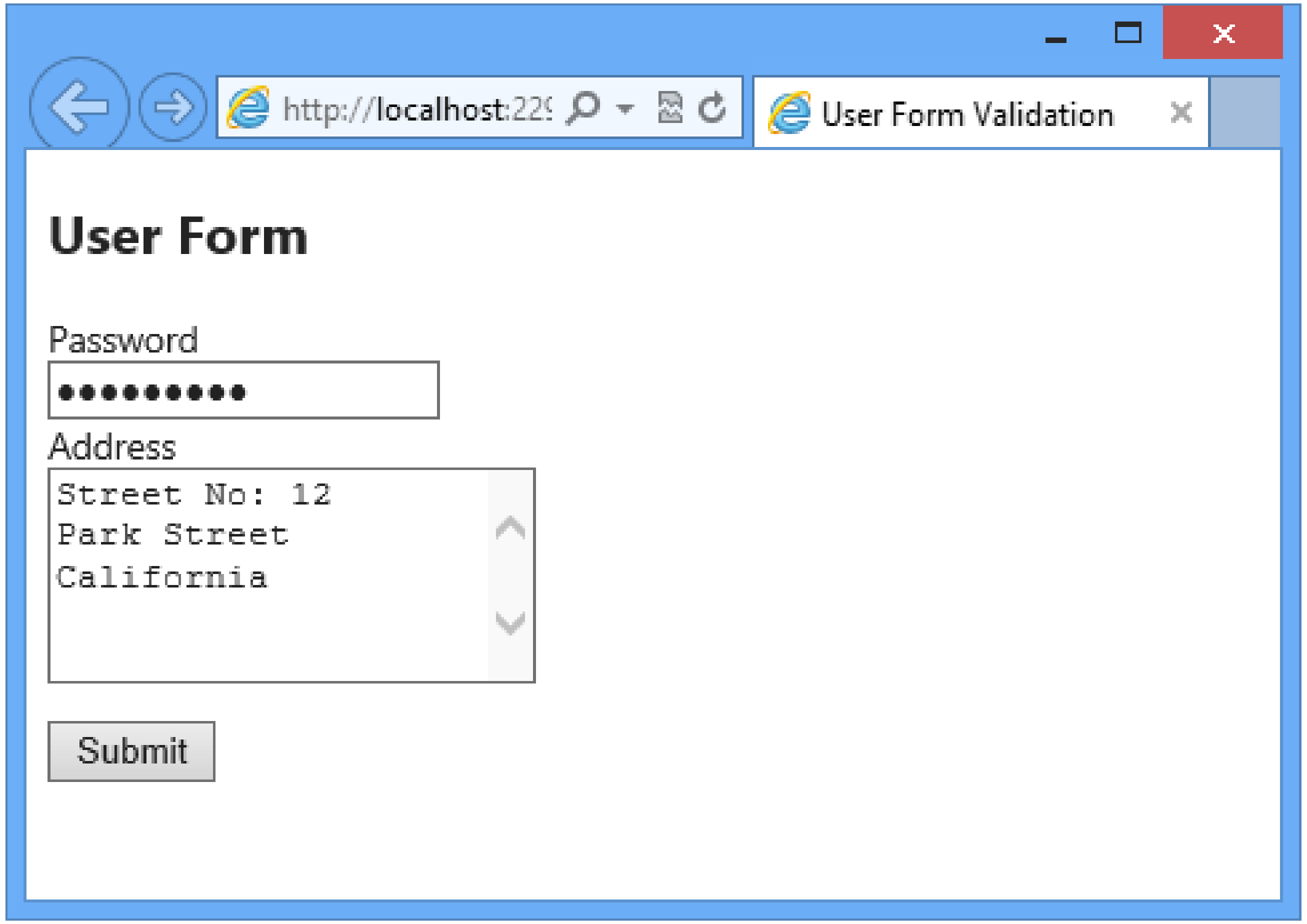
- Following code shows applying the DataType annotation to the Password and Address properties of a model

```
[DataType(DataType.Password)]
public string Password { get; set; }
[DataType(DataType.MultilineText)]
public string Address { get; set; }
```

- In this code:
  - The DataType annotation is first applied to the Password property. This instructs the view to display password field masks the user entered password.
  - Next, the DataType annotation is applied to the Address property. This instructs the view to display a multi-line text area for the Address property.

DataType Annotation

- Following figure shows the view that displays the fields for the Password and Address properties:



User Form

Password

●●●●●●●●●●

Address

```
Street No: 12
Park Street
California
```

ScaffoldColumn Annotation

- When you use scaffolding using the Create template, the view by default will create UI fields for all the properties of the model.
- However, you might need to ensure that the template does not create UI fields for certain properties.
- In such scenarios, you can use the ScaffoldColumnannotation passing a false value.
- Following code shows using the ScaffoldColumn annotation:

```
[ScaffoldColumn (false)]
public long Id { get; set; }
```

- In this code, the ScaffoldColumn annotation with a false value instructs the Create scaffolding template not to create a UI field in the view for the Id property.

ModelState Validation

- ModelState:
  - Is a class in the System.Web.Mvc namespace that encapsulate the state of model binding in an application.
  - Object stores the values that a user submits to update a model and any validation errors.
- You can check whether the state of a model is valid by using the ModelState.IsValid property.
- If the model binding succeeds, the ModelState.IsValid property returns true and you can accordingly perform the required function with the model data.

ModelState Validation

- Following code shows how to use the ModelState.IsValid property:

```
publicActionResult Index(User model)
{
        if (ModelState.IsValid)
        {
                /*Perform required function with the model data*/
                return Content("You have successfully registered");
        }
        else
                return View();
}
```

- In this code, the ModelState.IsValid is used to check the state of the User model.
  - If the property returns true, a success message is displayed.
  - Else the view is returned so that the user can enter valid values.

Summary

- The MVC Framework implements a validation workflow to validate user data.
- In an ASP.NET MVC application, you can manually validate data in the controller action.
- The MVC Framework provides several data annotations that you can apply as attributes to the properties of a model.
- The Required annotation when applied to a property validates that a value is specified for that property.
- The RegularExpression annotation when applied to a property validates that a value specified for that property matches the specified regular expression.
- The @Html.LabelFor() helper method when used in a strongly typed view displays a label with a text whose value is the corresponding property name.
- ModelState is a class in the System.Web.Mvc namespace that encapsulate the state of model binding in an application.