.NET App Dev Hands-On Lab

MVC Lab 7 – Custom Validation

This lab walks you through creating custom validation attributes and the related client-side scripts. Prior to starting this lab, you must have completed MVC Lab 6.

Part 1: Create the Server-Side validation attributes

- Add the following global using statements to the `GlobalUsings.cs` file in the `AutoLot.Mvc` project:

```
global using System.ComponentModel.DataAnnotations;
global using System.Reflection;
global using System.ComponentModel;
global using Microsoft.AspNetCore.Mvc.ModelBinding.Validation;
```

Step 1: Create the MustBeGreaterThanZeroAttribute attribute

- Create a new folder in the `AutoLot.Mvc` project named `Validation`. Add a new class named `MustBeGreaterThanZeroAttribute.cs`. Make the class public, inherit from `ValidationAttribute`, and implement `IClientModelValidator`. Add a primary constructor that takes in an error message and a default constructor that sets a default error message:

```
namespace AutoLot.Mvc.Validation;
public class MustBeGreaterThanZeroAttribute(string errorMessage)
    : ValidationAttribute(errorMessage), IClientModelValidator
{
  public MustBeGreaterThanZeroAttribute() : this("{0} must be greater than 0") { }
  public void AddValidation(ClientModelValidationContext context)
  {
  }
}
```

- Override the `FormatErrorMessage` method to properly format the `ErrorMessageString` (which is a property on the base `ValidationAttribute` class)

```
public override string FormatErrorMessage(string name) => string.Format(ErrorMessageString, name);
```

- Override the `IsValid` method to test if the value is greater than zero. This is used for server-side processing:

```
protected override ValidationResult IsValid(object value, ValidationContext validationContext)
{
  if (!int.TryParse(value.ToString(), out int result))
  {
    return new ValidationResult(FormatErrorMessage(validationContext.DisplayName));
  }
  return result > 0
    ? ValidationResult.Success
    : new ValidationResult(FormatErrorMessage(validationContext.DisplayName));
}
```

- Implement the `AddValidation` method. This method is used when generating the client-side implementation of the property.

```
public void AddValidation(ClientModelValidationContext context)
{
  string propertyDisplayName =
    context.ModelMetadata.DisplayName ?? context.ModelMetadata.PropertyName;
  string errorMessage = FormatErrorMessage(propertyDisplayName);
  context.Attributes.Add("data-val-greaterthanzero", errorMessage);
}
```

- Add the following global using statement to the `GlobalUsings.cs` file in the `AutoLot.Mvc` project:

```
global using AutoLot.Mvc.Validation;
```

Step 2: Create the MustNotBeGreaterThanAttribute attribute

- Add a new class named `MustNotBeGreaterThanAttribute.cs`.
- Make the class public, inherit from `ValidationAttribute`, and implement `IClientModelValidator`. Add the `AttributeUsage` attribute to the class so it targets properties and can be used more than once in a class. Create a primary constructor that takes in the other property name, the error message, and a prefix and another constructor that defaults the error message:

```
namespace AutoLot.Mvc.Validation;
[AttributeUsage(AttributeTargets.Property, AllowMultiple = true)]
public class MustNotBeGreaterThanAttribute(
  string otherPropertyName, string errorMessage, string prefix)
    : ValidationAttribute(errorMessage), IClientModelValidator
{
  private string _otherPropertyDisplayName;
  public MustNotBeGreaterThanAttribute(string otherPropertyName, string prefix = "")
    : this(otherPropertyName, "{0} must not be greater than {1}", prefix) { }
  public void AddValidation(ClientModelValidationContext context)
  {
  }
}
```

- Override the `FormatErrorMessage` method to properly format the `ErrorMessageString`:

```
public override string FormatErrorMessage(string name)
  => string.Format(ErrorMessageString, name, _otherPropertyDisplayName);
```

- Add the `SetOtherPropertyName` method, which will get the `Display` property of the other property.

```
internal void SetOtherPropertyName(PropertyInfo otherPropertyInfo)
{
  _otherPropertyDisplayName =
    otherPropertyInfo.GetCustomAttributes<DisplayAttribute>().FirstOrDefault()?.Name
    ?? otherPropertyInfo.GetCustomAttributes<DisplayNameAttribute>().FirstOrDefault()?.DisplayName
    ?? otherPropertyName;
}
```

- Override the `IsValid` method to test if the value is less than or equal to the other property. Once again, this is used for server-side processing:

```
protected override ValidationResult IsValid(object value, ValidationContext validationContext)
{
  var otherPropertyInfo = validationContext.ObjectType.GetProperty(otherPropertyName);
  if (otherPropertyInfo == null)
  {
    return new ValidationResult("Unable to validate property. Please contact support");
  }
  SetOtherPropertyName(otherPropertyInfo);
  if (!int.TryParse(value.ToString(), out int toValidate))
  {
    return new ValidationResult($"{validationContext.DisplayName} must be numeric.");
  }
  var otherPropObjectValue = otherPropertyInfo.GetValue(validationContext.ObjectInstance, null);
  if (otherPropObjectValue == null || !int.TryParse(otherPropObjectValue.ToString(),
    out var otherValue))
  {
    return new ValidationResult(FormatErrorMessage(validationContext.DisplayName));
  }
  return toValidate > otherValue
    ? new ValidationResult(FormatErrorMessage(validationContext.DisplayName))
    : ValidationResult.Success;
}
```

- Implement the `AddValidation` method. This method uses a helper method to get the `Display` attribute (if it exists) or the straight property name of the other property. This method is used when generating the client-side implementation of the property.

```
public void AddValidation(ClientModelValidationContext context)
{
  string propertyDisplayName = context.ModelMetadata.GetDisplayName();
  var propertyInfo = context.ModelMetadata.ContainerType.GetProperty(otherPropertyName);
  SetOtherPropertyName(propertyInfo);
  string errorMessage = FormatErrorMessage(propertyDisplayName);
  context.Attributes.Add("data-val-notgreaterthan", errorMessage);
  context.Attributes.Add("data-val-notgreaterthan-otherpropertyname", otherPropertyName);
  context.Attributes.Add("data-val-notgreaterthan-prefix", prefix);
}
```

Part 2: Create the Client-Side validation scripts

Step 1: Create the Validators

- Add a new folder named validations under the `wwwroot/js` folder. Add a new JavaScript file named `validators.js` in the new folder. Add the validator method for the `GreaterThanZero` validation. This name must match the name from the `AddValidation` method in the C# class:

```
$.validator.addMethod("greaterthanzero", function (value, element, params) {
    return value > 0;
});
```

- Add the unobtrusive adapter for the `GreaterThanZero` validation next. The rules property is simply set to true to enable validation, and the message is message from the `AddValidation` method:

```
$.validator.unobtrusive.adapters.add("greaterthanzero", function (options) {
    options.rules["greaterthanzero"] = true;
    options.messages["greaterthanzero"] = options.message;
});
```

- Add the validator method for the `NotGreaterThan` validation. As with the previous example, the name must match the name from the `AddValidation` method:

```
$.validator.addMethod("notgreaterthan", function (value, element, params) {
    return +value <= +$(params).val();
});
```

- Add the adapter for the `NotGreaterThan` validation:

```
$.validator.unobtrusive.adapters.add("notgreaterthan", ["otherpropertyname","prefix"], function
(options) {
    options.rules["notgreaterthan"] = "#" + options.params.prefix +
options.params.otherpropertyname;
    options.messages["notgreaterthan"] = options.message;
});
```

Step 2: Create the Error Formatter code

This code prettifies errors in the UI.

- Create a new JavaScript file named `errorFormatting.js` in the validations folder. Update the code to match the following:

```
$.validator.setDefaults({
  highlight: function (element, errorClass, validClass) {
    if (element.type === "radio") {
      this.findByName(element.name).addClass(errorClass).removeClass(validClass);
    } else {
      $(element).addClass(errorClass).removeClass(validClass);
      $(element).closest('.form-group').addClass('has-error');
    }
  },
  unhighlight: function (element, errorClass, validClass) {
    if (element.type === "radio") {
      this.findByName(element.name).removeClass(errorClass).addClass(validClass);
    } else {
      $(element).removeClass(errorClass).addClass(validClass);
      $(element).closest('.form-group').removeClass('has-error');
    }
  }
});
```

Part 3: Minify the JavaScript Files

To minimize specific files or to create bundles, add configuration options to the `AddWebOptimizer` method in the `Program.cs` file.

- Use `AddJavaScriptBundle` to bundle files. The first argument is the bundle name, then the files to be bundled:

```
builder.Services.AddWebOptimizer(options =>
{
...
  options.AddJavaScriptBundle("js/validations/validationCode.js", "js/validations/**/*.js");
  //options.AddJavaScriptBundle("js/validations/validationCode.js",
  //  "js/validations/validators.js", "js/validations/errorFormatting.js");
});
```

Part 4: Update the _ValidationScriptsPartial.cshtml

- Open `Views\Shared\_ValidationScriptsPartial.cshtml`. Add the following at the end of the development block:

```
<script src="~/js/validations/validators.js" asp-append-version="true"></script>
<script src="~/js/validations/errorFormatting.js" asp-append-version="true"></script>
```

- Open `Views\Shared\_ValidationScriptsPartial.cshtml`. Add the following at the end of the non-development block:

```
<script src="~/js/validations/validationCode.js"></script>
```

Part 5: Use the attribute

Step 1: Create the AddToCartViewModel

- Create a new class named `AddToCartViewModel` in the Models directory of the `AutoLot.Mvc` project, and update it to the following:

```
namespace AutoLot.Mvc.Models;
public class AddToCartViewModel
{
  public int Id { get; set; }
  [Display(Name="Stock Quantity")] public int StockQuantity { get; set; }
  public int ItemId { get; set; }
  [Required, MustBeGreaterThanZero, MustNotBeGreaterThan(nameof(StockQuantity))]
  public int Quantity { get; set; }
}
```

Step 2: Add the View

- Create a new view named `Validation.cshtml` in the Views\Home folder and update it to match the following:

```
@model AutoLot.Mvc.Models.AddToCartViewModel
@{ ViewData["Title"] = "Validation"; }
<h1>Validation</h1>
<div class="row">
  <div class="col-md-4">
    <form asp-action="Validation">
      <div asp-validation-summary="ModelOnly" class="text-danger"></div>
      <div><label asp-for="Id" class="col-form-label"></label>
        <input asp-for="Id" class="form-control" />
        <span asp-validation-for="Id" class="text-danger"></span>
      </div>
      <div><label asp-for="StockQuantity" class="col-form-label"></label>
        <input asp-for="StockQuantity" class="form-control" />
        <span asp-validation-for="StockQuantity" class="text-danger"></span>
      </div>
      <div><label asp-for="ItemId" class="col-form-label"></label>
        <input asp-for="ItemId" class="form-control" />
        <span asp-validation-for="ItemId" class="text-danger"></span>
      </div>
      <div><label asp-for="Quantity" class="col-form-label"></label>
        <input asp-for="Quantity" class="form-control" />
        <span asp-validation-for="Quantity" class="text-danger"></span>
      </div>
      <div style="margin-top:5px">
        <input type="submit" value="Save" class="btn btn-primary" />
      </div>
    </form>
  </div>
</div>
@section Scripts {
  @{await Html.RenderPartialAsync("_ValidationScriptsPartial");}
}
```

Step 3: Add the Controller Method

- Add the method to the `HomeController`:

```
[HttpGet]
public IActionResult Validation()
{
  var vm = new AddToCartViewModel
  {
    Id = 1,
    ItemId = 1,
    StockQuantity = 2,
    Quantity = 0
  };
  return View(vm);
}
```

Step 4: Update the _Menu Partial

- Add the following menu item to the `_Menu.cshtml` partial:

```
<li class="nav-item">
  <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Validation"
title="Validation Example">Validation<i class="fas fa-check"></i></a>
</li>
```

Summary

The lab created the custom validation attributes and their client-side validation scripts, minified the scripts, and updated the validation partial view. It then used the attributes on a view model. The next lab will build the `Admin` area for the application.