.NET App Dev Hands-On Lab

Razor Pages Lab 7 – Custom Validation

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

Part 1: Create the Server-Side validation attributes

- Add the following global using statements to the `GlobalUsings.cs` file in the `AutoLot.Web` 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.Web` project named `Validation`. Add a new class named `MustBeGreaterThanZeroAttribute.cs`. Make the class public, inherit from `ValidationAttribute`, and implement `IClientModelValidator`. Add a primary constructor and a secondary constructor:

```
namespace AutoLot.Web.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 (value is not int intValue)
  {
    return new ValidationResult(FormatErrorMessage(validationContext.DisplayName));
  }
  return intValue <= 0
     ? new ValidationResult(FormatErrorMessage(validationContext.DisplayName))
     : ValidationResult.Success;
}
```

- 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 statements to the `GlobalUsings.cs` file in the `AutoLot.Web` project:

```
global using AutoLot.Web.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, and add a primary constructor and another constructor:

```
namespace AutoLot.Web.Validation;
[AttributeUsage(AttributeTargets.Property, AllowMultiple = true)]
public class MustNotBeGreaterThanAttribute(
    string otherPropertyName, string errorMessage, string prefix)
  : ValidationAttribute(errorMessage), IClientModelValidator
{
  readonly string _otherPropertyName = otherPropertyName;
  string _otherPropertyDisplayName = otherPropertyName;
  readonly string _prefix = prefix;
  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 format the `ErrorMessageString` properly:

```
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 (value is not int intValue)
  {
    return new ValidationResult(FormatErrorMessage(validationContext.DisplayName));
  }
  var otherPropObjectValue = otherPropertyInfo.GetValue(validationContext.ObjectInstance, null);
  if (otherPropObjectValue is not int otherValue)
  {
    return new ValidationResult(FormatErrorMessage(validationContext.DisplayName));
  }
  return intValue > 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. Also add the adapter. The rules property is simply set to true to enable validation, and the message is message from the `AddValidation` method:

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

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

- 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');
        }
    }
});
```

Step 3: Minify the JavaScript Files

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

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

```
builder.Services.AddWebOptimizer(options =>
{
  options.AddJavaScriptBundle("js/validations/validationCode.js", "js/validations/**/*.js");
  //This is another format to bundle and minify the files
  //options.AddJavaScriptBundle("js/validations/validationCode.js",
  //  "js/validations/validators.js", "js/validations/errorFormatting.js");
});
```

Part 3: 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 4: Use the Attribute

Step 1: Create the AddToCartViewModel

- Create a new folder named `Models` in the `AutoLot.Web` project. In this folder, add a new class named `AddToCartViewModel` and update it to the following:

```
namespace AutoLot.Web.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; }
}
```

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

```
global using AutoLot.Web.Models;
```

Step 2: Create the Validation Page

- Create a new Razor Page named Validation in the Pages directory, and update the code behind to the following:

```
namespace AutoLot.Web.Pages;
public class ValidationModel : PageModel
{
  [ViewData]
  public string Title => "Validation Example";
  [BindProperty]
  public AddToCartViewModel Entity { get; set; }
  public void OnGet()
  {
    Entity = new AddToCartViewModel
    {
      Id = 1,
      ItemId = 1,
      StockQuantity = 2,
      Quantity = 0
    };
  }
  public IActionResult OnPost()
  {
    if (!ModelState.IsValid)
    {
      return Page();
    }
    return RedirectToPage("Validation");
  }
}
```

- Next, update the markup to the following:

```
@page
@model AutoLot.Web.Pages.ValidationModel
<h1>Validation</h1>
<h4>Add To Cart</h4><hr />
<div class="row">
  <div class="col-md-4">
    <form asp-page="/Validation">
      <div asp-validation-summary="ModelOnly" class="text-danger"></div>
      <div>
        <label asp-for="Entity.Id" class="col-form-label"></label>
        <input asp-for="Entity.Id" class="form-control" />
        <span asp-validation-for="Entity.Id" class="text-danger"></span>
      </div>
      <div><label asp-for="Entity.StockQuantity" class="col-form-label"></label>
        <input asp-for="Entity.StockQuantity" class="form-control" />
        <span asp-validation-for="Entity.StockQuantity" class="text-danger"></span>
      </div>
      <div><label asp-for="Entity.ItemId" class="col-form-label"></label>
        <input asp-for="Entity.ItemId" class="form-control" />
        <span asp-validation-for="Entity.ItemId" class="text-danger"></span>
      </div>
      <div>
        <label asp-for="Entity.Quantity" class="col-form-label"></label>
        <input asp-for="Entity.Quantity" class="form-control" />
        <span asp-validation-for="Entity.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: 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-page="/Validation" title="Validation
Example">Validation<i class="fas fa-check"></i></a>
</li>
```

Summary

This lab added custom validation attributes and demonstrated how to use them.

Next Steps

The next lab creates an admin area for Make maintenance.