Build an EF and ASP.NET Core 3.0 App HOL

Lab 11

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 Lab 10.

Part 1: Create the Server-Side validation attributes

Step 1: Create the MustBeGreaterThanZeroAttribute attribute

1) Create a new folder in the `SpyStore.Hol.Mvc` project named `Validation`. Add a new class named `MustBeGreaterThanZeroAttribute.cs`.

2) Add the following using statements:

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

3) Make the class public, inherit from `ValidationAttribute`, and implement `IClientModelValidator`:

```
public class MustBeGreaterThanZeroAttribute : ValidationAttribute, IClientModelValidator
{
  public void AddValidation(ClientModelValidationContext context)
  {
  }
}
```

4) Add two constructors. One that takes a custom error message and another that uses a default error message:

```
public MustBeGreaterThanZeroAttribute() : this("{0} must be greater than 0") { }
public MustBeGreaterThanZeroAttribute(string errorMessage) : base(errorMessage) { }
```

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

```
public override string FormatErrorMessage(string name)
{
  return string.Format(ErrorMessageString, name);
}
```

6) 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));
  }
  if (result > 0)
  {
    return ValidationResult.Success;
```

All files copyright Phil Japikse (http://www.skimedic.com/blog)

```
  };
  return new ValidationResult(FormatErrorMessage(validationContext.DisplayName));
}
```

7) 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);
}
```

Step 2: Create the MustNotBeGreaterThanAttribute attribute

1) Add a new class named `MustNotBeGreaterThanAttribute.cs`.

2) Add the following using statements to the top of the file:

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

3) Make the class public, inherit from `ValidationAttribute`, and implement `IClientModelValidator`. Also, add the `AttributeUsage` attribute to the class so it targets properties and can be used more than once in a class:

```
[AttributeUsage(AttributeTargets.Property, AllowMultiple = true)]
public class MustNotBeGreaterThanAttribute : ValidationAttribute, IClientModelValidator
{
  public void AddValidation(ClientModelValidationContext context)
  {
  }
}
```

4) Add two constructors. Since this attribute compares the value of this property to another property of the class instance, the constructors need to take in the other property name and an optional prefix. Just like the previous example, one takes a custom error message and the other uses a default error message:

```
readonly string _otherPropertyName;
string _otherPropertyDisplayName;
readonly string _prefix;
public MustNotBeGreaterThanAttribute(string otherPropertyName, string prefix = "")
  : this(otherPropertyName, "{0} must not be greater than {1}", prefix) { }
public MustNotBeGreaterThanAttribute(string otherPropertyName, string errorMessage, string prefix)
  : base(errorMessage)
{
  _otherPropertyName = otherPropertyName;
  _otherPropertyDisplayName = otherPropertyName;
  _prefix = prefix;
}
```

5) Override the `FormatErrorMessage` method to properly format the `ErrorMessageString`:

```csharp
public override string FormatErrorMessage(string name)
{
  return string.Format(ErrorMessageString, name, _otherPropertyDisplayName);
}
```

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

```csharp
internal void SetOtherPropertyName(PropertyInfo otherPropertyInfo)
{
  var displayAttribute =
    otherPropertyInfo.GetCustomAttributes<DisplayAttribute>().FirstOrDefault();
  _otherPropertyDisplayName = displayAttribute?.Name ?? _otherPropertyName;
}
```

7) 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:

```csharp
protected override ValidationResult IsValid(object value, ValidationContext validationContext)
{
  var otherPropertyInfo = validationContext.ObjectType.GetProperty(_otherPropertyName);
  SetOtherPropertyName(otherPropertyInfo);
  if (!int.TryParse(value.ToString(), out int toValidate))
  {
    return new ValidationResult($"{validationContext.DisplayName} must be numeric.");
  }
  var otherValue = (int)otherPropertyInfo.GetValue(validationContext.ObjectInstance, null);
  return toValidate > otherValue
    ? new ValidationResult(FormatErrorMessage(validationContext.DisplayName))
    : ValidationResult.Success;
}
```

8) 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.

```csharp
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: Update the View Models in the MVC Project

Step 1: Update the AddToCartViewModel

1) Add the following using statement:

```csharp
using SpyStore.Hol.Mvc.Validation;
```

2) Update the quantity property to the following:

```
public class AddToCartViewModel : CartRecordWithProductInfo
{
  [Required, MustNotBeGreaterThan(nameof(UnitsInStock)), MustBeGreaterThanZero]
  public new int Quantity { get; set; }
}
```

Step 2: Update the CartRecordViewModel

1) Add the following using statements:

```
using SpyStore.Hol.Mvc.Validation;
```

2) Update the quantity property to the following:

```
public class CartRecordViewModel : CartRecordWithProductInfo
{
  [Required, MustNotBeGreaterThan(nameof(UnitsInStock))]
  public new int Quantity { get; set; }
}
```

Part 3: Create the Client-Side validation scripts

Step 1: Create the Validators

1) Add a new folder named validations under the wwwroot/js folder. Add a new JavaScript file named validators.js in the new folder.

2) 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;
});
```

3) 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;
});
```

4) 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();
});
```

5) 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;
});
```

All files copyright Phil Japikse (http://www.skimedic.com/blog)

Step 2: Create the formatter code

This code prettifies errors in the UI.

1) Create a new JavaScript file named `errorFormatting.js` in the validations folder.

2) Update the code to match the following:

```javascript
$.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'); //.removeClass('has-
success');
        }
    },
    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');  //.addClass('has-
success');
        }
    }
});
```

Part 4: Bundle and Minify the JavaScript Files

Step 1: Bundle the JavaScript validation files

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

1) Use `AddJavaScriptBundle` to bundle files. First argument is the bundle name, next are the files to be bundled:

```csharp
services.AddWebOptimizer(options =>
{
  options.MinifyCssFiles(); //Minifies all CSS files
  //options.MinifyJsFiles(); //Minifies all JS files
  options.MinifyJsFiles("js/site.js");
  options.AddJavaScriptBundle("js/validations/validationCode.js", "js/validations/**/*.js");
});
```

Part 5: Update the _ValidationScriptsPartial.cshtml

1) Open `Views\Shared\_ValidationScriptsPartial.cshtml`.

2) Add the following to the block defined as the `include="Development"` environment:

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

3) Add the following to the block defined as the `exclude="Development"` environment:

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

Summary

The lab created the custom validation attribute, client-side validation scripts and formatting, bundled and minified the scripts, and updated the validation partial view.