

.NET 9 App Dev Hands-On Lab

MVC Lab 7b – Custom Validation

This lab walks you through creating custom validation attributes and the related client-side scripts. Before starting this lab, you must have completed Razor Pages\MVC Lab 7a.

Part 1: Create the Client-Side Validation Scripts

Step 1: Create the Validators

This code attaches the validation attributes to the jQuery validation system.

- Add a new folder named `validations` under the `wwwroot/js` folder in the `AutoLot.Mvc` project. Add a new JavaScript file named `validators.js` to the new folder. Add the validators and adapters for the two validation attributes:

```
$.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;
});

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

$.validator.unobtrusive.adapters.add("notgreaterthan", ["otherpropertyname", "prefix"], function (options) {
    var rule = "#" + options.params.otherpropertyname;
    if (options.params.prefix !== undefined && options.params.prefix !== null && options.params.prefix !== '') {
        rule = "#" + options.params.prefix + "_" + options.params.otherpropertyname;
    }
    options.rules["notgreaterthan"] = rule;
    options.messages["notgreaterthan"] = options.message;
});
```

- Add a new function that will force revalidation of the entire form:

```
function reValidate(ctl) {
    var form = ctl.form;
    $(form).valid();
}
```

Step 2: Create the Error Formatter

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');
    }
  }
});
```

Step 3: Minify and Bundle the JavaScript Files

Add configuration options to the `AddWebOptimizer` method in the `Program.cs` file to minimize (in dev) and minimize and bundle (in prod) the new JS files.

- Use `AddJavaScriptBundle` to bundle the files. The first argument is the bundle name, and the second argument(s) is(are) the files to be bundled:

```
builder.Services.AddWebOptimizer(options =>
{
  //omitted for brevity
  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");
});
```

Step 4: Update the _ValidationScriptsPartial.cshtml

- Open Views\Shared_ValidationScriptsPartial.cshtml. Add the following at the end of the include="Development,Local" block:

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

- Add the following at the end of the non-development block:

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

Part 2: Use the Validation Attributes

Step 1: Add the MVC-specific View Model

- Add a new class named AddToCartViewModelMvc in the ViewModels folder of the AutoLot.Services project and update it to the following:

```
namespace AutoLot.Services.Models;
public class AddToCartViewModelMvc : AddToCartViewModelBase
{
    [Required]
    [MustBeGreaterThanZero]
    [MustNotBeGreaterThan(nameof(StockQuantity))]
    public int Quantity { get; set; }
}
```

Step 2: Add the Controller Methods

- Add the HTTPGet and HTTPPost methods to the HomeController:

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

[HttpPost]
public async Task<IActionResult> Validation(AddToCartViewModelMvc viewModel)
{
    if (!ModelState.IsValid)
    {
        return View(viewModel);
    }
    await Task.Delay(5000);
    return RedirectToAction(nameof(Validation), nameof(HomeController).RemoveController());
}
```

Step 3: Add the View

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

```
@model AddToCartViewModelMvc
@{ ViewData["Title"] = "Validation"; }
<h1>Validation</h1>
<div class="row">
    <div class="col-md-4">
        <form id="validationForm" 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" oninput="reValidate(this)"
                    class="form-control" onchange="reValidate(this)" />
                <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" oninput="reValidate(this)" class="form-control" />
                <span asp-validation-for="Quantity" class="text-danger"></span>
            </div>
            <div style="margin-top: 5px">
                <button id="submitEditForm" type="submit" class="btn btn-primary">
                    Save <i class="fas fa-save"></i>
                </button>
            </div>
        </form>
    </div>
</div>
```

```

@section Scripts {
    <partial name="_ValidationScriptsPartial"/>
    <script type="text/javascript" language="javascript">
        function reValidate(input) {
            const form = $("#validationForm");
            form.validate().element(input);
        }

        $(document).ready(function() {
            $("#validationForm").on("submit",
                function(e) {
                    e.preventDefault();
                    const form = $(this);
                    const validator = form.validate();
                    if (form.valid()) {
                        const submitButton = document.getElementById("submitEditForm");
                        submitButton.disabled = true;
                        submitButton.innerHTML = 'Submitting...';
                        this.submit();
                    } else {
                        validator.resetForm();
                    }
                }
            );
        });
    </script>
}

```

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 added validation scripts and demonstrated how to use the validation attributes.

Next Steps

The next lab will add an admin area for Make maintenance.