MVC Validation Lab

Lab 1: Add Validation

Open the **Product.cs** file and add more data annotations

```
using System;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;

namespace AdvWorks.EntityLayer
{
  [Table("Product", Schema = "SalesLT")]
  public partial class Product
  {
    public Product()
    {
      SellStartDate = DateTime.Now;
    }

    public int ProductID { get; set; }

    [Display(Name = "Product Name")]
    [Required]
    [StringLength(50, MinimumLength = 4)]
    public string Name { get; set; }

    [Display(Name = "Product Number")]
    [StringLength(25, MinimumLength = 3)]
    public string ProductNumber { get; set; }

    // This field is optional
    public string Color { get; set; }

    [Display(Name = "Cost")]
    [Required]
    [DataType(DataType.Currency)]
    [Range(0.01, 9999)]
    [Column(TypeName = "decimal(18, 2)")]
    public decimal StandardCost { get; set; }

    [Display(Name = "Price")]
    [Required]
    [DataType(DataType.Currency)]
    [Range(0.01, 9999)]
    [Column(TypeName = "decimal(18, 2)")]
    public decimal ListPrice { get; set; }

    [StringLength(5)]
    public string Size { get; set; }

    [Column(TypeName = "decimal(8, 2)")]
    [Range(0.5, 2000)]
    public decimal? Weight { get; set; }

    [Display(Name = "Start Selling Date")]
    [Required]
    [DataType(DataType.Date)]
    public DateTime SellStartDate { get; set; }

    [Display(Name = "End Selling Date")]
    [DataType(DataType.Date)]
```

```
    public DateTime? SellEndDate { get; set; }

    [Display(Name = "Date Discontinued")]
    [DataType(DataType.Date)]
    public DateTime? DiscontinuedDate { get; set; }

    [NotMapped]
    [Display(Name = "Is Active?")]
    public bool IsActive { get; set; }

    public override string ToString()
    {
      return $"{Name} ({ProductID})";
    }
  }
}
```

Open the **_Detail.cshtml** file

Add the validation summary just after the hidden input tag for the ProductID.

```
<div asp-validation-summary="All" class="text-danger">
</div>
```

Add the following tags after each appropriate <input>

```
<span asp-validation-for="SelectedProduct.Name" class="text-danger"
/>

<span asp-validation-for="SelectedProduct.ProductNumber"
class="text-danger" />

<span asp-validation-for="SelectedProduct.StandardCost" class="text-
danger" />

<span asp-validation-for="SelectedProduct.ListPrice" class="text-
danger" />

<span asp-validation-for="SelectedProduct.Size" class="text-danger"
/>

<span asp-validation-for="SelectedProduct.Weight" class="text-
danger" />

<span asp-validation-for="SelectedProduct.SellStartDate"
class="text-danger" />

<span asp-validation-for="SelectedProduct.SellEndDate" class="text-
danger" />

<span asp-validation-for="SelectedProduct.DiscontinuedDate"
class="text-danger" />
```

Open the **ProductMaintenance.cshtml** file and add the following at the bottom of the page to enable client-side validation

```
@section Scripts
{
  <partial name="_ValidationScriptsPartial" />
}
```

Open the **_Search.cshtml** file and modify the Search button as follows

```
<button formnovalidate="formnovalidate" class="btn btn-success">
  Search
</button>
```

Open the **ProductMaintenanceController.cs** file and make the following modifications. **NOTE:** Make sure you are editing the [HttpPost] action method and not the GET method.

```
[HttpPost]
public IActionResult ProductMaintenance(ProductViewModel vm)
{
  vm.Repository = _repo;
  vm.ColorRepository = _colorRepo;

  if (ModelState.IsValid) {
    return RedirectToAction("ProductMaintenance");
  }
  else {
    vm.LoadColors();
    vm.IsDetailVisible = true;
  }

  return View(vm);
}
```

Try it Out

Run the application

Click on the Edit button to edit any product

Delete all the text in the Product Name, Cost and Price fields

Click the **Save** button and you should see the validation errors appear

Lab 2: Try Server-Side Validation

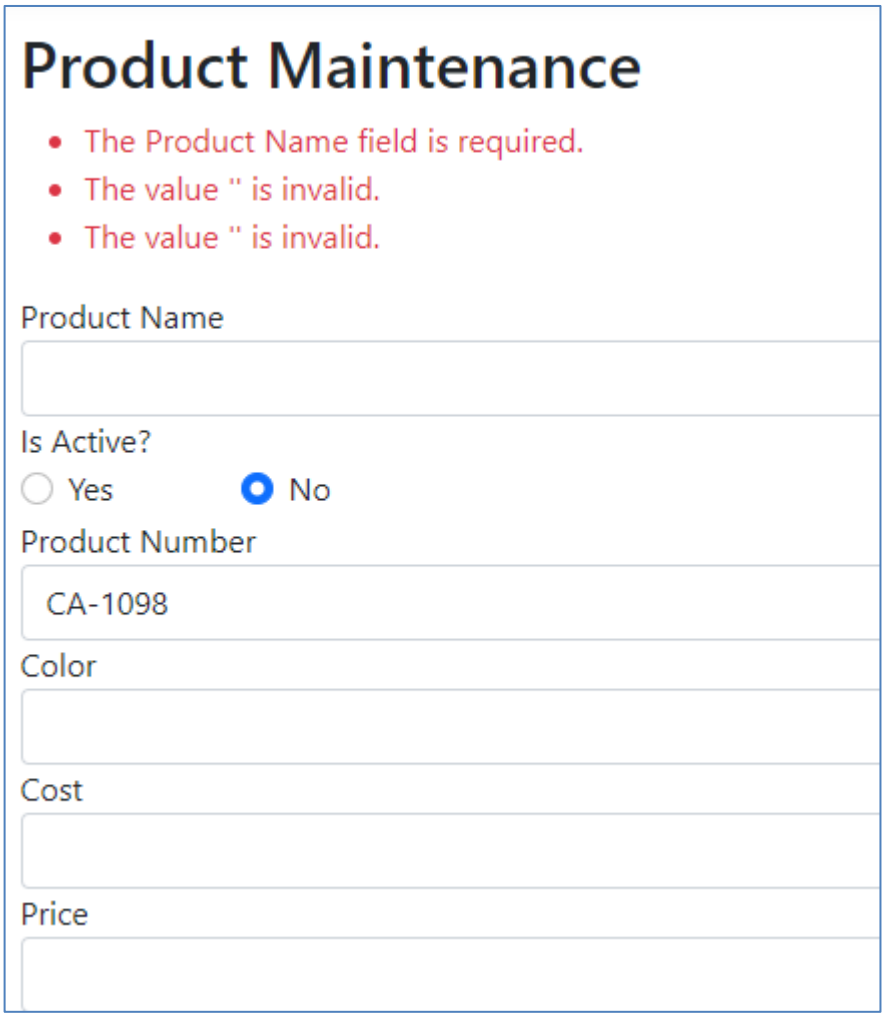
Open the ProductController.cs file and set a breakpoint on the **if (ModelState.IsValid)** line.

While you are still in the detail page and have no data in the Product Name, Cost and Price fields, disable JavaScript in your browser and click on Save to have it post-back

Notice when you hit the breakpoint, the IsValid property is set to false.

Click the Continue button to see the error messages.

After the post-back you should see some weird error messages for Cost and Price.



To fix this, you need to make the Standard Cost and List Price fields nullable decimal? values.

Lab 3: Custom Server-Side Validators

Go to **AdvWorks.Common** project

Right mouse-click and add a new folder named **ValidatorClasses**

Right mouse-click on the **ValidatorClasses** folder and create a new class called **DecimalLessThanAttribute.cs**.

```
using System.ComponentModel.DataAnnotations;
using System.Reflection;

namespace AdvWorks.Common.Validators
{
  public class DecimalLessThanAttribute : ValidationAttribute
  {
    private readonly string _propertyToCompare;

    public DecimalLessThanAttribute(string propToCompare)
    {
      _propertyToCompare = propToCompare;
    }

    protected override ValidationResult IsValid(object value,
ValidationContext validationContext)
    {
      ErrorMessage = ErrorMessageString;
      if (value != null) {
        decimal currentValue = (decimal)value;
        decimal comparisonValue;
        PropertyInfo pi =
validationContext.ObjectType.GetProperty(_propertyToCompare);

        if (pi == null) {
          throw new ArgumentException("Property: " +
_propertyToCompare + " was not found.");
        }
        comparisonValue =
(decimal)pi.GetValue(validationContext.ObjectInstance);

        if (currentValue > comparisonValue) {
          return new ValidationResult(ErrorMessage);
        }

        return ValidationResult.Success;
      }
      else {
        return new ValidationResult("Property: " +
_propertyToCompare + " was null.");
      }
    }
  }
}
```

Add another class called **DateLessThanAttribute.cs** in the **ValidatorClasses** folder.

```
using System.ComponentModel.DataAnnotations;
using System.Reflection;

namespace AdvWorks.Common.Validators
{
  public class DateLessThanAttribute : ValidationAttribute
  {
    private readonly string _propertyToCompare;

    public DateLessThanAttribute(string propToCompare)
    {
      _propertyToCompare = propToCompare;
    }

    protected override ValidationResult IsValid(object value,
ValidationContext validationContext)
    {
      ErrorMessage = ErrorMessageString;
      if (value != null) {
        DateTime currentValue = (DateTime)value;
        DateTime comparisonValue;
        PropertyInfo pi =
validationContext.ObjectType.GetProperty(_propertyToCompare);

        if (pi == null) {
          throw new ArgumentException("Property: " +
_propertyToCompare + " was not found.");
        }
        if (pi.GetValue(validationContext.ObjectInstance) != null) {
          comparisonValue =
(DateTime)pi.GetValue(validationContext.ObjectInstance);

          if (currentValue > comparisonValue) {
            return new ValidationResult(ErrorMessage);
          }
        }

        return ValidationResult.Success;
      }
      else {
        return new ValidationResult("Property: " +
_propertyToCompare + " was null.");
      }
    }
  }
}
```

Decorate Properties

Open the **Product.cs** file and add a using statement.

```
using AdvWorks.Common.Validators;
```

Decorate the ***StandardCost*** property

```
[DecimalLessThan("ListPrice", ErrorMessage = "Cost must be less than
the Price.")]
public decimal StandardCost { get; set; }
```

Decorate the ***SellStartDate*** property

```
[DateLessThan("SellEndDate", ErrorMessage = "Start Selling Date must
be less than the End Selling Date.")]
public DateTime SellStartDate { get; set; }
```

Try it Out

Run the application and click the Edit button

Fill in the following data:

Cost: 20

Price: 10

End Selling Date: A year earlier than the Start Selling Date

Click the Save button and you should see the custom validation appear **after** the post-back.