MVC Forms Lab

Lab 1: Add Display Attribute

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

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

Add the appropriate [Display] attributes

```
public partial class Product
{
  public Product()
  {
    SellStartDate = DateTime.Now;
  }

  public int ProductID { get; set; }

  [Display(Name = "Product Name")]
  public string Name { get; set; }

  [Display(Name = "Product Number")]
  public string ProductNumber { get; set; }

  public string Color { get; set; }

  [Display(Name = "Cost")]
  public decimal? StandardCost { get; set; }

  [Display(Name = "Price")]
  public decimal? ListPrice { get; set; }

  public string Size { get; set; }

  public decimal? Weight { get; set; }

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

  [Display(Name = "End Selling Date")]
  public DateTime? SellEndDate { get; set; }

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

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

Try it Out

Run the application, click on the Edit button, and make sure the labels now match the [Display] attributes you added.

Lab 2: Finish the Product Detail Page

Open the **\Views\ProductMaintenance_Detail.cshtml** file

Add the following fields just before the </form> tag

```
<div class="form-group">
  <label asp-for="SelectedProduct.Size"></label>
  <input class="form-control" asp-for="SelectedProduct.Size" />
</div>
<div class="form-group">
  <label asp-for="SelectedProduct.Weight"></label>
  <input class="form-control" asp-for="SelectedProduct.Weight" />
</div>
<div class="form-group">
  <label asp-for="SelectedProduct.SellStartDate"></label>
  <input class="form-control" asp-
for="SelectedProduct.SellStartDate" />
</div>
<div class="form-group">
  <label asp-for="SelectedProduct.SellEndDate"></label>
  <input class="form-control" asp-for="SelectedProduct.SellEndDate"
/>
</div>
<div class="form-group">
  <label asp-for="SelectedProduct.DiscontinuedDate"></label>
  <input class="form-control" asp-
for="SelectedProduct.DiscontinuedDate" />
</div>
```

Add a Save and Cancel button just before the </form> tag

```
<div class="form-group">
  <button class="btn btn-primary">Save</button>
  <a class="btn btn-secondary"
     asp-action="ProductMaintenance">
    Cancel
  </a>
</div>
```

Modify the <form> tag

```
<form method="post" asp-action="ProductMaintenance">
```

Open the **ProductMaintenanceController.cs** file and add a new POST method

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

  // TODO: Add code to save data
  System.Diagnostics.Debugger.Break();

  vm.Search();

  return View(vm);
}
```

Try it Out

Run the application

Click on any product's Edit button

Click the Cancel button and you should be returned to the Product Maintenance page

Click on any product's Edit button

Make some changes to any of the input fields

Click the Save button and in the [HttpPost] method and check the *vm.SelectedProduct* property and view the changes you made

Lab 3: Add Drop-Down of Colors

Open the **ProductViewModel.cs** file and add modify the **Get**() method to add the call to the LoadColors() method.

```
public virtual void Get(int id)
{
  IsDetailVisible = true;

  if (Repository == null) {
    throw new ApplicationException("Must set the Repository
property.");
  }
  else {
    SelectedProduct = Repository.Get(id);
  }

  if (SelectedProduct != null) {
    TotalRows = 1;
  }

  // Load Colors
  LoadColors();
}
```

Open the **_Detail.cshtml** file and replace the color input with a <select> list

```
<div class="form-group">
  <label asp-for="SelectedProduct.Color"></label>
  <select class="form-select"
          asp-for="SelectedProduct.Color"
          asp-items="@(new SelectList(Model.Colors, "ColorName",
"ColorName"))">
  </select>
</div>
```

Modify Controller

Open the **ProductMaintenanceController.cs** and add a new property

```
private readonly IRepository<Color, ColorSearch> _colorRepo;
```

Modify the **constructor**

```
public ProductMaintenanceController(
  ILogger<ProductMaintenanceController> logger,
  IProductRepository repo, IRepository<Color, ColorSearch>
colorRepo)
{
  _logger = logger;
  _repo = repo;
  _colorRepo = colorRepo;
}
```

Modify the **ProductEdit**() method to pass in the ColorRepository object

```
[HttpGet]
public IActionResult ProductEdit(int id)
{
  // Create view model passing in repository
  ProductViewModel vm = new(_repo, _colorRepo);

  // Call method to load a product
  vm.LoadProduct(id);

  return View("ProductMaintenance", vm);
}
```

Try it Out

Run the application

Click on any product and see the color select list

Look at the Color select list

Change the color selection and check the value posted back

Demo 4: Radio Buttons

Open the **Product.cs** file in the **AdvWorks.EntityLayer** project

Add a new property to this class

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

Open the **_Detail.cshtml** file and add a new form-group after the **Product Name** form group.

```
<div class="form-group">
  <div class="row">
    <label asp-for="SelectedProduct.IsActive"></label>
    <div class="col-1">
      <div class="form-check">
        <input asp-for="SelectedProduct.IsActive" value="true"
               type="radio" class="form-check-input" />
        <label asp-for="SelectedProduct.IsActive"
               class="form-check-label">Yes</label>
      </div>
    </div>
    <div class="col-1">
      <div class="form-check">
        <input asp-for="SelectedProduct.IsActive"
               value="false" type="radio" class="form-check-input"
/>
        <label asp-for="SelectedProduct.IsActive"
               class="form-check-label">No</label>
      </div>
    </div>
  </div>
</div>
```

Try it Out

Run the application

View the Radio buttons on the detail page

Change the Radio button to Yes and post back to see the value changes in the IsActive property