LAB 3. BUILDING BASIC WEB APPLICATIONS WITH ASP.NET MVC 5 (2)

1. Introduction

This lab is a continuation of the previous two labs. We will focus on the **View** layer, the final component of the MVC pattern. You will learn how to create views to display data passed from the **Controller** and use **Razor Syntax** and **Tag Helpers** to build dynamic and interactive web pages.

2. Objective

Upon completion of this lab, you will be able to:

- Create **Views** that correspond to your controller's action methods.
- Display data from the **Model** on a web page.
- Use **Razor Syntax** to embed C# code directly into HTML.
- Utilize **Helper Methods** to simplify form creation and link generation.

3. Project setup

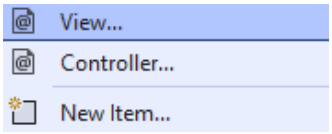
- This lab builds directly on **Lab 2**.
- Please ensure you have completed all the steps in **Lab 2**.
- Your ProductController should have all the necessary action methods for CRUD operations.

4. Complete the following requirements

a. Create the "Index" View

- In Solution Explorer, right-click the **Views** folder.
- Select Add => New Folder. Name the folder **Product**.

- Right-click the newly created Product folder and select Add -> View....



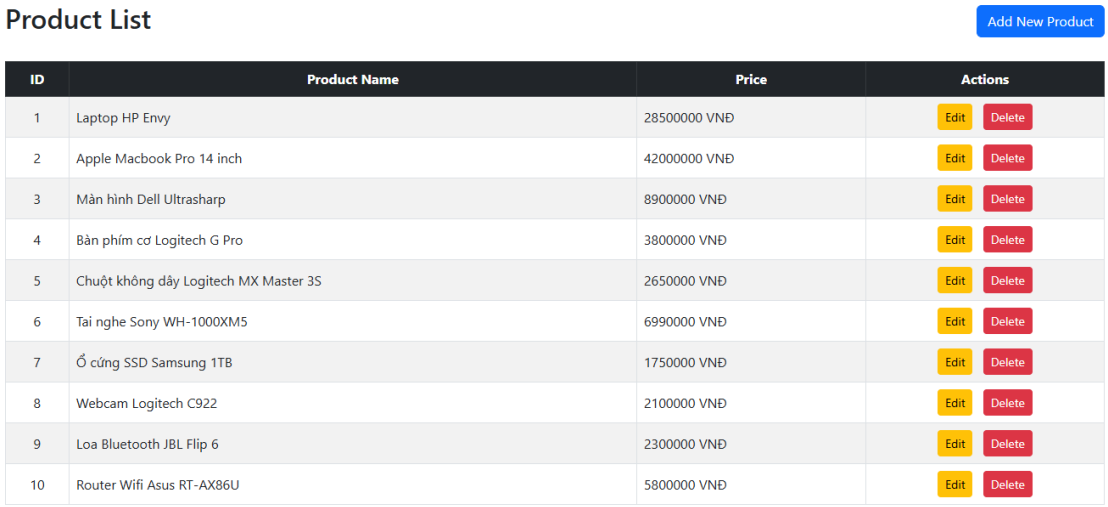
- Name the view **Index.cshtml**.

- Inside **Index.cshtml**, add the following Razor code to display the list of products passed from the Index action method in your **ProductController**:

```
@model List<Lab2.Models.Product>
<div class="container mt-5">
    <div class="d-flex justify-content-between align-items-center mb-4">
        <h2>Product List</h2>
        <a href="@Url.Action("Create","Product")" class="btn btn-primary">Add New Product</a>
    </div>
    @if (Model != null && Model.Any())
    {
        <div class="table-responsive">
            <table class="table table-striped table-hover table-bordered">
                <thead class="table-dark">
                    <tr>
                        <th class="text-center" scope="col">ID</th>
                        <th class="text-center" scope="col">Product Name</th>
                        <th class="text-center" scope="col">Price</th>
                        <th class="text-center" scope="col">Actions</th>
                    </tr>
                </thead>
                <tbody>
                    @{ int count = 1; }
```

```
                @foreach (var item in Model)
                {
                    <tr>
                        <td class="align-middle text-center">@count</td>
                        <td class="align-middle">@item.Name</td>
                        <td class="align-middle">@item.Price VNĐ</td>
                        <td class="align-middle text-center">
                            <a href="@Url.Action("Update","Product", new { id = item.Id })"
class="btn btn-sm btn-warning me-2">Edit</a>
                            <a href="@Url.Action("Delete","Product", new { id = item.Id })"
class="btn btn-sm btn-danger"
                               onclick="return confirm('Are you sure you want to delete this
product?');">Delete</a>
                        </td>
                    </tr>
                    count++;
                }
            </tbody>
        </table>
    </div>
}
else
{
    <span class="text-danger">No data available</span>
}
</div>
```

- **Index view**



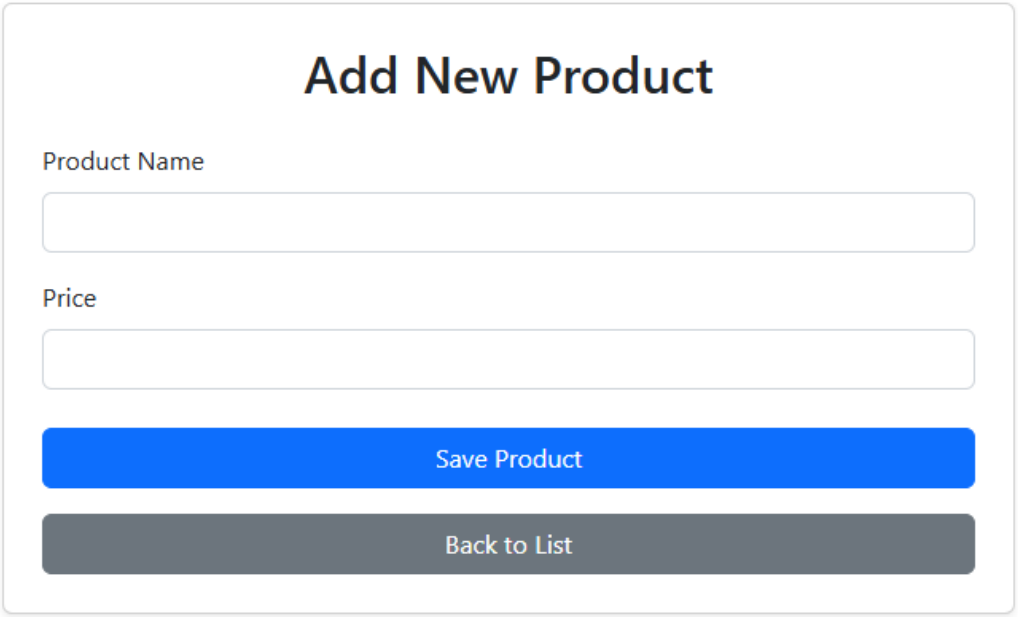
b. Create the "Create" View:

- In Solution Explorer, right-click the Views/Product folder.
- Select Add -> View....
- Name the view **Create.cshtml**.
- Add the following code to the **Create.cshtml** file:

```
<div class="container mt-5">
  <div class="row justify-content-center">
    <div class="col-md-8 col-lg-6">
      <div class="card shadow-sm p-4">
        <h2 class="card-title text-center mb-4">Add New Product</h2>

        <form action="@Url.Action("Create","Product")" method="post">
          <div class="mb-3">
            <label for="productName" class="form-label">Product Name</label>
            <input type="text" class="form-control" id="productName"
name="Name" required>
          </div>
```

```
                <div class="mb-3">
                    <label for="productPrice" class="form-label">Price</label>
                    <input type="number" class="form-control" id="productPrice"
name="Price" required min="0">
                </div>


                @Html.AntiForgeryToken()
                <div class="d-grid gap-2 mt-4">
                    <button type="submit" class="btn btn-primary">Save Product</button>
                    <a href="@Url.Action("Index","Product")" class="btn btn-secondary mt-
2">Back to List</a>
                </div>
            </form>
        </div>
    </div>
  </div>
</div>
```

- Create view



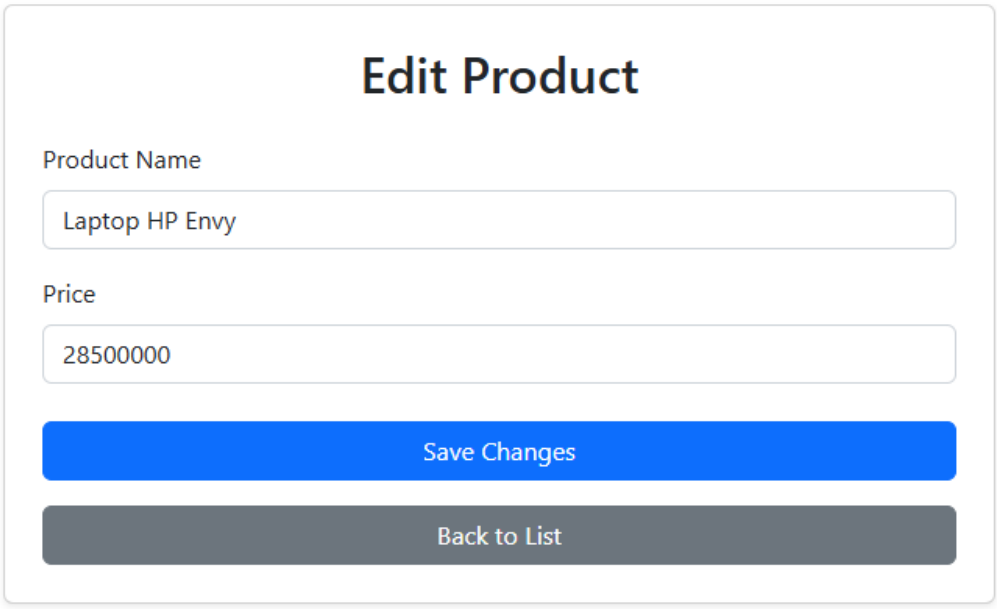
c. Create the "Update" View:

- In Solution Explorer, right-click the Views/Product folder.
- Select Add -> View....
- Name the view **Update.cshtml.**
- Add the following code to the **Update.cshtml** file:

```
@model Lab2.Models.Product
<div class="container mt-5">
   <div class="row justify-content-center">
      <div class="col-md-8 col-lg-6">
         <div class="card shadow-sm p-4">
            <h2 class="card-title text-center mb-4">Edit Product</h2>

            <form action="@Url.Action("Update","Product")" method="post">
               <!-- Hidden field to store the product ID, crucial for updates -->
               <input type="hidden" id="productId" name="Id" value="@Model.Id">

               <div class="mb-3">
                  <label for="productName" class="form-label">Product Name</label>
                  <input type="text" class="form-control" id="productName"
name="Name" value="@Model.Name" required>
               </div>
               <div class="mb-3">
                  <label for="productPrice" class="form-label">Price</label>
                  <input type="number" class="form-control" id="productPrice"
name="Price" value="@Model.Price" required min="0">
               </div>

               @Html.AntiForgeryToken()
               <div class="d-grid gap-2 mt-4">
```

```
                    <button type="submit" class="btn btn-primary">Save Changes</button>
                    <a href="@Url.Action("Index","Product")" class="btn btn-secondary mt-2">Back to List</a>
                </div>
            </form>
        </div>
    </div>
</div>
```

- Update view



d. Understand Razor Syntax & Helper Methods

Razor is a markup syntax for embedding C# code directly into HTML. It uses the @ symbol to transition from HTML to C# code. In the code you provided, you've used several important Razor features and helper methods:

- **Implicit Expressions**: @Url.Action(...) is a powerful example. This helper method generates a URL to a specific controller action, ensuring that your links are always correct, even if your routing rules change.

- **Code Blocks**: @{ int count = 1; } is a C# code block used to declare a variable. The code within these blocks is not rendered as output.
- **Control Structures**: @if(...) and @foreach(...) are used to add conditional logic and loops, allowing you to dynamically generate HTML based on your model's data.
- @Html.AntiForgeryToken(): This helper method generates a hidden form field that is used to protect against Cross-Site Request Forgery (CSRF) attacks.
- **Data Binding**: The Update view demonstrates how to bind model data to form elements using @Model.PropertyName. This populates the form with the current product's information. The hidden input field for Id is crucial, as it ensures the correct product is updated when the form is submitted.

5. Summary

In this lab, you successfully built the View layer of your ASP.NET MVC application. You learned how to display data passed from the controller using Razor Syntax and simplified your HTML with Helper Methods like @Url.Action. You are now equipped with the fundamental knowledge to create dynamic web pages in ASP.NET MVC. The next lab will focus on building the corresponding views for the Update, and Delete functionalities to complete the full CRUD cycle.

6. Exercises

a. Exercise 1: Category CRUD with Razor Views

- Create a Category model with properties: Id, Name, Description.
- Implement CategoryController with CRUD actions (Index, Details, Create, Edit, Delete).
- Create Razor Views:
    - Index.cshtml: Display all categories in an HTML <table>. Each row should have links to Details, Edit, and Delete.
    - Details.cshtml: Show category info with simple HTML.
    - Create.cshtml: Create new category.

- Edit.cshtml: Same as Create, but pre-filled with existing values using Razor (value="@Model.Name").
- Delete.cshtml: Show category info and confirm deletion.

b. Exercise 2: Order CRUD with Razor Views

- Create an Order model: Id, CustomerName, OrderDate, List<int> ProductIds.
- Implement OrderController with CRUD actions (Index, Details, Create, Edit, Delete).
- Create Razor Views:
  - Index.cshtml: Show orders in a <table>, including CustomerName and OrderDate.
  - Details.cshtml: Display order info and use @foreach to list related products.
  - Create.cshtml: A form with plain HTML inputs.
  - Edit.cshtml: Same as Create, but pre-filled with value="@Model.CustomerName" and selected products.
  - Delete.cshtml: Show order info and confirm deletion.