**BỘ GIÁO DỤC VÀ ĐÀO TẠO**

**ĐẠI HỌC CÔNG NGHỆ TP.HCM**



**Thực hành Lập trình Web**

|  |  |
| --- | --- |
| **Biên soạn:** | Nguyễn Đình Ánh  Nguyễn Huy cường  Trần Đăng Khoa  Phạm Hữu Kỳ |

**Thực hành Lập trình Web**

Ấn bản 2023

MỤC LỤC

[MỤC LỤC I](#_Toc154494611)

[HƯỚNG DẪN III](#_Toc154494612)

[BÀI 1: XÂY DỰNG LAYOUT CHO TRANG WEB BẰNG HTML & CSS 1](#_Toc154494613)

[1.1 Yêu cầu chung của bài thực hành 1](#_Toc154494614)

[1.1.1 Bố cục trang web 1](#_Toc154494615)

[1.1.2 Menu điều hướng 1](#_Toc154494616)

[1.1.3 Danh sách sản phẩm 1](#_Toc154494617)

[1.1.4 Thanh bên (Sidebar) 2](#_Toc154494618)

[1.1.5 Chân trang (Footer) 2](#_Toc154494619)

[1.1.6 CSS Grid 2](#_Toc154494620)

[1.1.7 Độ phản hồi 2](#_Toc154494621)

[1.2 Nội dung bổ sung 2](#_Toc154494622)

[1.3 Chú ý 2](#_Toc154494623)

[1.4 Hướng dẫn 4](#_Toc154494624)

[1.4.1 Tập tin Index.html 4](#_Toc154494625)

[1.4.2 Tập tin Style.css 7](#_Toc154494626)

[1.5 Yêu cầu bổ sung 11](#_Toc154494627)

[1.5.1 Xem chi tiết sản phẩm 11](#_Toc154494628)

[1.5.2 Sử dụng Bootstrap 11](#_Toc154494629)

[1.5.3 Triển khai yêu cầu bổ sung 12](#_Toc154494630)

[BÀI 2: XÂY DỰNG ỨNG DỤNG WEBSITE CƠ BẢN VỚI ASP.NET CORE MVC 17](#_Toc154494631)

[2.1 Khởi tạo ứng dụng ASP.NET Core 18](#_Toc154494632)

[2.2 Cấu trúc dự án ASP.NET Core MVC 20](#_Toc154494633)

[2.2.1 \_\_Layout 21](#_Toc154494634)

[2.2.2 Razor View Engine 22](#_Toc154494635)

[2.2.3 RenderBody 22](#_Toc154494636)

[2.2.4 RenderSectionAsync 23](#_Toc154494637)

[2.3 Bài tập 24](#_Toc154494638)

[2.3.1 Yêu Cầu Bài Thực Hành: Ứng Dụng Thêm/Đọc/Xóa/Sửa trên ASP.NET Core MVC 24](#_Toc154494639)

[2.3.2 Code Mẫu Chi Tiết 25](#_Toc154494640)

[2.3.3 Kết quả 35](#_Toc154494641)

[2.3.5 Bổ sung tính năng upload file cho ứng dụng 38](#_Toc154494642)

[2.3.6 Yêu cầu bổ sung 43](#_Toc154494643)

[BÀI 3: XÂY DỰNG ỨNG DỤNG WEBSITE BÁN HÀNG VỚI ASP.NET CORE MVC (PHẦN 1) 44](#_Toc154494644)

[3.1 Bài tập 44](#_Toc154494645)

[3.1.1 Yêu cầu 44](#_Toc154494646)

[3.1.2 Hướng dẫn thực hiện 44](#_Toc154494647)

[3.1.3 Kết quả 56](#_Toc154494648)

[3.1.4 Yêu cầu bổ sung 58](#_Toc154494649)

[BÀI 4: XÂY DỰNG ỨNG DỤNG WEB BÁN HÀNG VỚI ASP.NET CORE MVC (PHẦN 2) 59](#_Toc154494650)

[4.1 Bài thực hành 59](#_Toc154494651)

[4.1.1 Yêu cầu 59](#_Toc154494652)

[4.2 Hướng dẫn thực hiện 59](#_Toc154494653)

[4.2.1 Cấu Hình ASP.NET Core Identity 59](#_Toc154494654)

[4.2.2 Scaffolding ASP.NET Core Identity 61](#_Toc154494655)

[4.2.3 Thêm Liên Kết Đăng Ký và Đăng Nhập 62](#_Toc154494656)

[4.2.4 Phần vùng Area 64](#_Toc154494657)

[4.3 Yêu cầu bổ sung 65](#_Toc154494658)

[BÀI 5: XÂY DỰNG ỨNG DỤNG WEBSITE BÁN HÀNG VỚI ASP.NET CORE MVC (PHẦN 3) 66](#_Toc154494659)

[5.1 Mục tiêu của bài thực hành 66](#_Toc154494660)

[5.1.1 Yêu cầu 66](#_Toc154494661)

[5.2 Hướng dẫn thực hiện 66](#_Toc154494662)

[5.2.1 Code mẫu thực hiện chức năng giỏ hàng 66](#_Toc154494663)

[5.2.2 Hướng dẫn thực hiện chức năng đặt hàng 70](#_Toc154494664)

[5.2.3 Yêu cầu bổ sung 73](#_Toc154494665)

[BÀI 6: RESTful API 74](#_Toc154494666)

[6.1 Mục tiêu của bài thực hành 74](#_Toc154494667)

[6.1.1 Giới thiệu 74](#_Toc154494668)

[6.1.2 Yêu cầu 74](#_Toc154494669)

[6.2 Hướng dẫn thực hiện 75](#_Toc154494670)

[6.3 Yêu cầu bổ sung 79](#_Toc154494671)

[TÀI LIỆU THAM KHẢO 80](#_Toc154494672)

HƯỚNG DẪN

MÔ TẢ MÔN HỌC

Học phần trang bị cho sinh viên những kiến thức và kỹ năng cơ bản về ASP.NET Core MVC để có thể xây dựng các ứng dụng web từ đơn giản đến nâng cao. Sinh viên sẽ nắm được các khái niệm cơ bản về ASP.NET Core MVC như: cấu trúc chung của một ứng dụng MVC, vai trò và mối quan hệ của các thành phần chính như Controller, Action, View, Model. Sinh viên sẽ hiểu được sự khác biệt giữa ASP.NET Core MVC với các framework khác như ASP.NET Web Form, ASP.NET MVC cũ. Điều này giúp sinh viên nắm được lợi thế của ASP.NET Core MVC để có thể lựa chọn và áp dụng phù hợp. Cụ thể, Sinh viên sẽ biết cách tạo project ASP.NET Core MVC, tạo controller và action, tạo view và layout, tạo model và sử dụng Entity Framework Core, xử lý form và validate dữ liệu, cấu hình routing và xây dựng hệ thống authentication. Những kiến thức và kỹ năng này sẽ giúp sinh viên có thể tự tin xây dựng các ứng dụng web bằng ASP.NET Core MVC.

NỘI DUNG MÔN HỌC

* Bài 1. Xây dựng layout cho trang web bán hàng bằng HTML & CSS
* Bài 2. Xây dựng ứng dụng cơ bản với ASP.NET Core MVC
* Bài 3: Xây dựng ứng dụng website bán hàng với ASP.NET Core MVC (Phần 1)
* Bài 4: Xây dựng ứng dụng website bán hàng với APS.NET Core MVC (Phần 2)
* Bài 5: Xây dựng ứng dụng website bán hàng với APS.NET Core MVC (Phần 3)
* Bài 6: Xây dựng ứng dụng website bán hàng với APS.NET Core MVC (Phần 4)

KIẾN THỨC TIỀN ĐỀ

Kiến thức cơ bản về lập trình hướng đối tượng bằng C# hoặc VB.NET. Điều này bao gồm các khái niệm như lớp, đối tượng, kế thừa, đa hình,...

Kiến thức cơ bản về cơ sở dữ liệu, đặc biệt là cách thao tác với cơ sở dữ liệu SQL Server.

YÊU CẦU MÔN HỌC

Người học phải dự học đầy đủ các buổi lên lớp và làm bài tập đầy đủ ở nhà.

CÁCH TIẾP NHẬN NỘI DUNG MÔN HỌC

Để học tốt môn này, người học cần ôn tập các bài đã học, trả lời các câu hỏi và làm đầy đủ bài tập; đọc trước bài mới và tìm thêm các thông tin liên quan đến bài học.

Đối với mỗi bài học, người học đọc trước mục tiêu và tóm tắt bài học, sau đó đọc nội dung bài học. Kết thúc mỗi ý của bài học, người học cần thực hành đầy đủ các hướng dẫn trong phần bài tập.

PHƯƠNG PHÁP ĐÁNH GIÁ MÔN HỌC

Môn học được đánh giá gồm:

* Điểm chuyên cần (30%): Hình thức và cách đánh giá do giảng viên dạy thực hành quyết định được phê duyệt của bộ môn.
* Điểm bài tập (70%): Hình thức làm bài tập trong các buổi học thực hành và giảng viên đánh giá chấm điểm. Danh sách bài tập thực hành được bộ môn kiểm duyệt và cung cấp vào đầu khóa học.

# XÂY DỰNG LAYOUT CHO TRANG WEB BẰNG HTML & CSS

Sau khi học xong bài này, sinh viên có thể:

* Hiểu và biết cách sử dụng HTML & CSS trong việc thiết kế các trang web tĩnh.
* Có khả năng thiết kế được các trang web với tính linh hoạt cao và đẹp mắt.
* Biết cách thiết kế trang web đảm bảo được tính nhất quán và trải nghiệm người dùng tốt trên cả máy tính và thiết bị di động.

## Yêu cầu chung của bài thực hành

### Bố cục trang web

* Chia trang thành hai cột trên máy tính. Cột trái sẽ hiển thị sản phẩm và cột phải sẽ hiển thị thông tin mạng xã hội.
* Trên thiết bị di động, hiển thị các nội dung thành 1 cột.

### Menu điều hướng

* Thêm một menu điều hướng ở phía trên trang web với ít nhất ba liên kết: Trang chủ, Sản phẩm và Liên hệ.

### Danh sách sản phẩm

* Tạo một danh sách sản phẩm trong cột trái.
* Hiển thị mỗi sản phẩm trong một thẻ sản phẩm với hình ảnh, tên sản phẩm, mô tả và giá.

### Thanh bên (Sidebar)

* Tạo một phần sidebar trong cột phải
* Thêm ít nhất bốn liên kết mạng xã hội vào sidebar (ví dụ: Facebook, Twitter, Instagram, LinkedIn).
* Sử dụng font-awesome để hiển thị các icon cho mạng xã hội tương ứng.

### Chân trang (Footer)

* Tạo một phần footer ở dưới cùng của trang web.
* Trên máy tính, hiển thị ba cột trong footer. Trên thiết bị di động, hiển thị thành 1 cột, các cột xếp chồng lên nhau.

### CSS Grid

* Sử dụng CSS Grid để xây dựng bố cục của trang web.

### Độ phản hồi

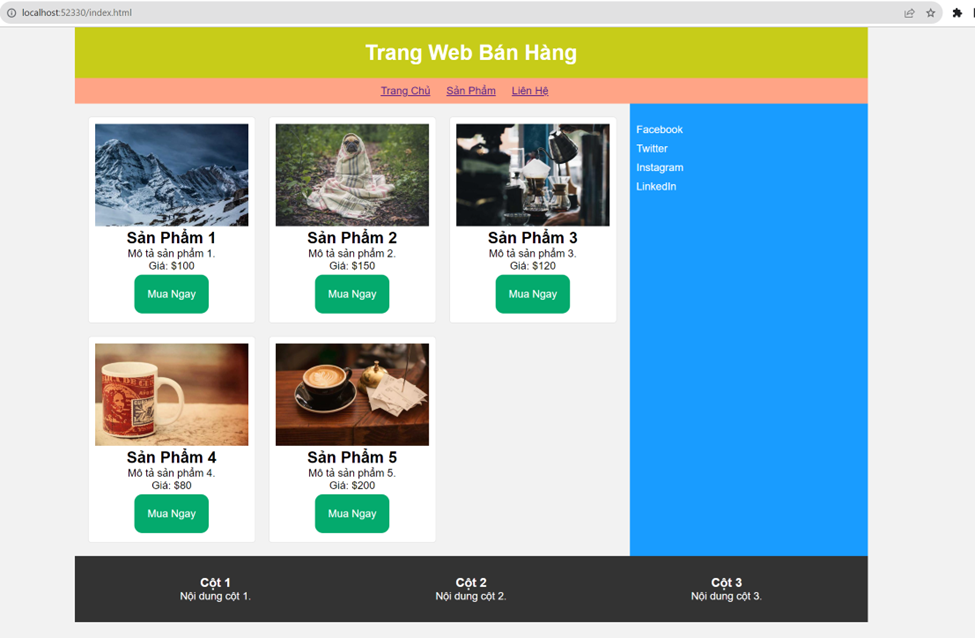
* Đảm bảo trang web đáp ứng tốt trên các kích thước màn hình khác nhau.

## Nội dung bổ sung

* Tùy chỉnh màu sắc, kiểu chữ và biểu đồ màu sắc để làm cho trang web thẩm mỹ hơn.
* Thêm hiệu ứng hover cho sản phẩm và các liên kết.
* Sử dụng hình ảnh thay thế cho các sản phẩm.

## Chú ý

* Sử dụng HTML và CSS để thực hiện bài tập này.
* Tuân theo các nguyên tắc CSS tốt như sử dụng các **class** và **id** có ý nghĩa.



Hình . Thực hành HTML và CSS

## Hướng dẫn

### Tập tin Index.html

A screenshot of a computer code

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

### Tập tin Style.css

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer code

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer code

Description automatically generated

## Yêu cầu bổ sung

### Xem chi tiết sản phẩm

- Khi người dùng nhấp vào một sản phẩm trong danh sách sản phẩm, họ sẽ được chuyển hướng đến một trang xem chi tiết sản phẩm.

- Trang xem sản phẩm cần hiển thị thông tin chi tiết về sản phẩm, bao gồm hình ảnh lớn, tên sản phẩm, mô tả chi tiết và giá cả.

- Thêm một nút "Quay lại" hoặc "Tiếp tục mua sắm" để người dùng có thể trở lại danh sách sản phẩm.

### Sử dụng Bootstrap

- Thay vì sử dụng CSS tùy chỉnh, bạn có thể sử dụng Bootstrap để xây dựng giao diện tương tự.

- Thêm tập tin của Bootstrap CSS và JavaScript trong mã HTML (có thể sử dụng các phiên bản Bootstrap từ trang chính thức của Bootstrap).

- Sử dụng các lớp và thành phần của Bootstrap để thiết kế trang web

### Triển khai yêu cầu bổ sung

#### Xem chi tiết sản phẩm

- Tạo một trang mới trong dự án của bạn để hiển thị thông tin chi tiết về sản phẩm.

- Trong danh sách sản phẩm, sử dụng một thẻ `<a>` để bao quanh hình ảnh và tên sản phẩm. Đặt href của thẻ `<a>` thành URL của trang xem chi tiết sản phẩm.

- Sử dụng một nút hoặc liên kết để cho phép người dùng quay lại trang danh sách sản phẩm.

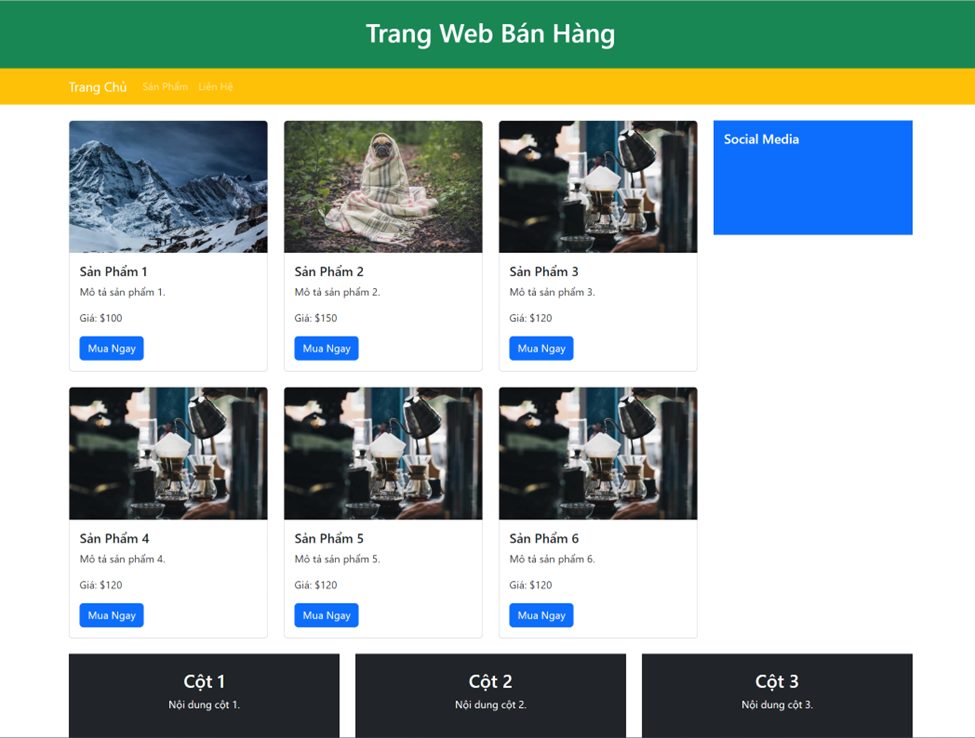
#### Sử dụng Bootstrap

- Tải Bootstrap CSS và JavaScript từ trang chính thức của Bootstrap (<https://getbootstrap.com/>).

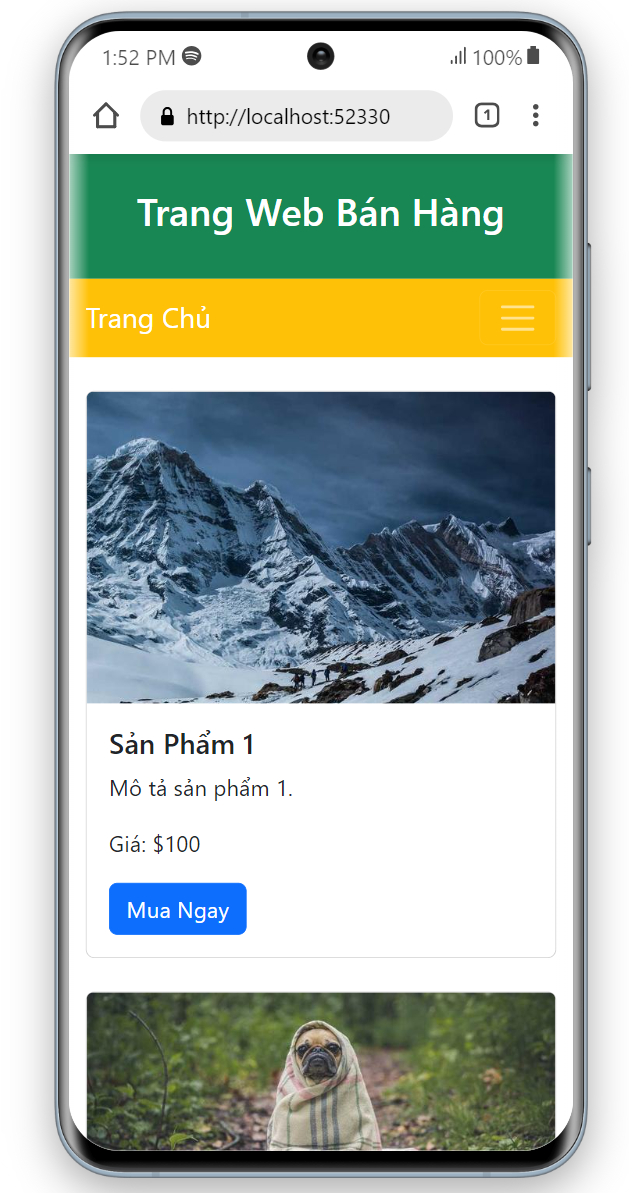
- Bao gồm tệp Bootstrap CSS và JavaScript vào mã HTML của bạn, thường được đặt trong phần `<head>` của tệp HTML.

- Sử dụng lớp và các thành phần Bootstrap để thiết kế giao diện của trang web.

#### Kết quả thực hiện bằng Bootstrap



Hình .. Kết quả thực hiện bằng Bootstrap



Hình . Kết quả hiểu thị trên thiết bị di động

A screenshot of a computer code

Description automatically generated

A screenshot of a computer program

Description automatically generatedA screenshot of a computer program

Description automatically generated

A screenshot of a computer code

Description automatically generated

*---Hết Lab TH 01---*

# XÂY DỰNG ỨNG DỤNG WEBSITE CƠ BẢN VỚI ASP.NET CORE MVC

Sau khi học xong bài này, sinh viên có thể:

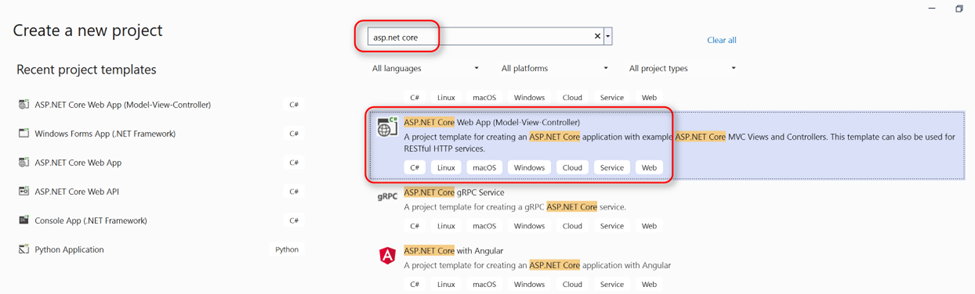
* Hiểu rõ cấu trúc nội bộ của ứng dụng ASP.NET Core MVC, bao gồm sự phân chia và tương tác giữa các lớp và modules.
* Hiểu cơ chế truyền tải và nhận dữ liệu trong mô hình MVC, bao gồm cách thức hoạt động của binding và routing
* Nắm vững khái niệm và quản lý của 'view', cũng như sử dụng Razor syntax
* Hiểu sâu về cấu trúc và chức năng của 'model' trong MVC, cũng như cách thức tương tác và xử lý dữ liệu.
* Hiểu rõ vai trò của 'controller' trong MVC, bao gồm cách quản lý luồng dữ liệu, xử lý yêu cầu và phản hồi.
* Hiểu rõ cách truyền 'model' giữa các thành phần trong MVC.
* Thực hành kỹ thuật xử lý và lưu trữ hình ảnh.

## Khởi tạo ứng dụng ASP.NET Core

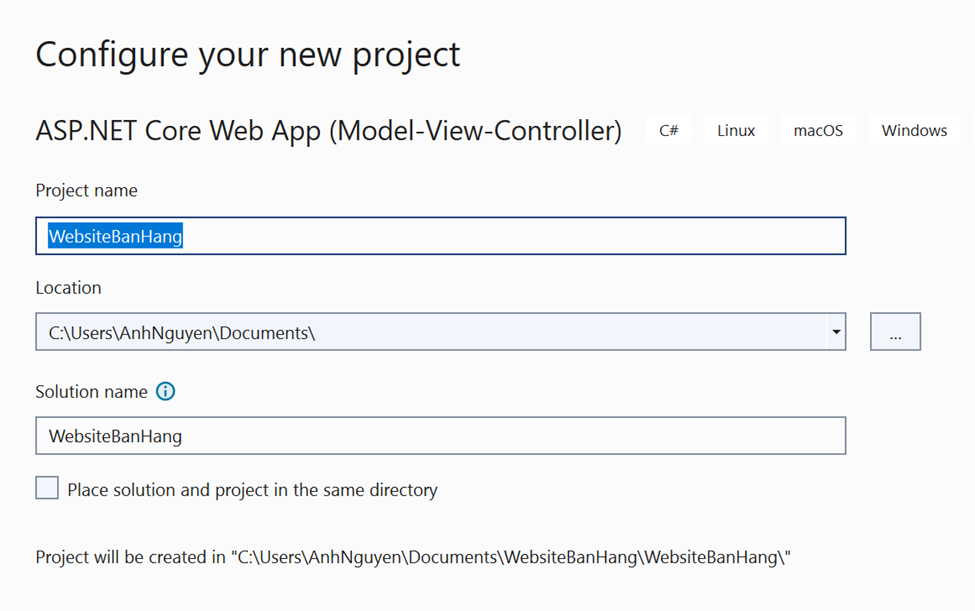
Khởi động phần mềm **Visual Studio 2022**

A purple sign with white text

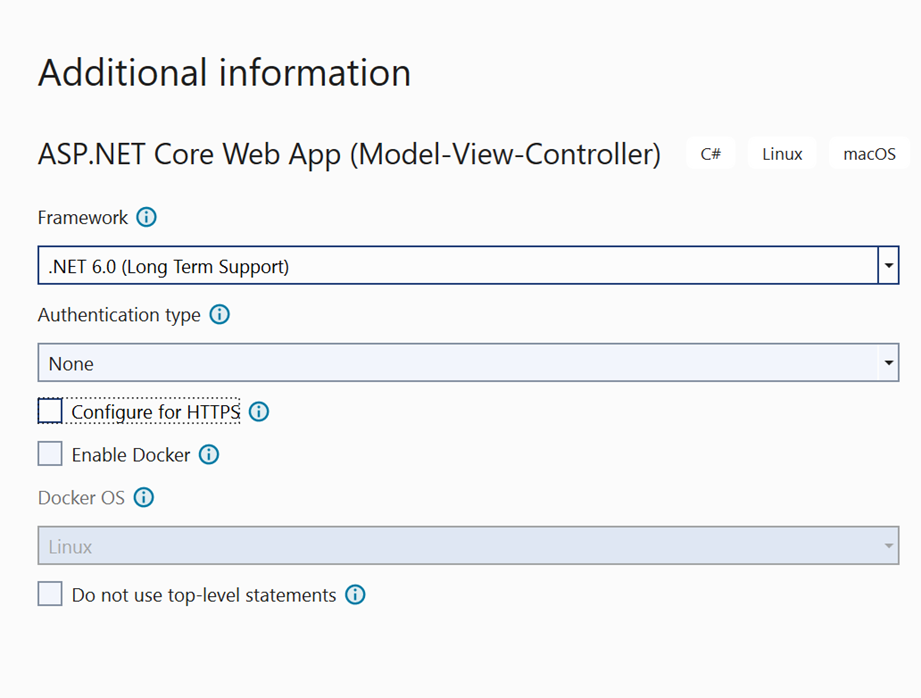
Description automatically generated



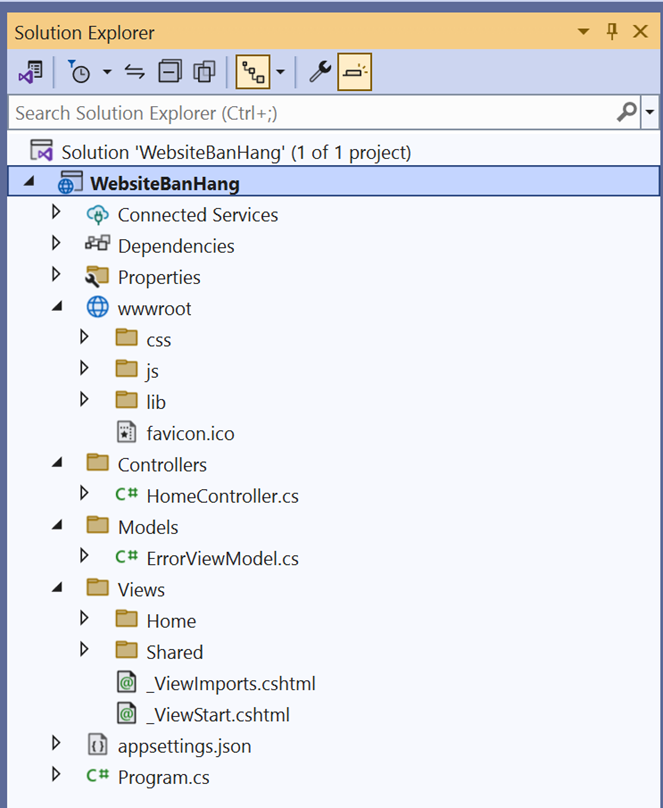
Hình . Tạo project ASP.NET CORE mới



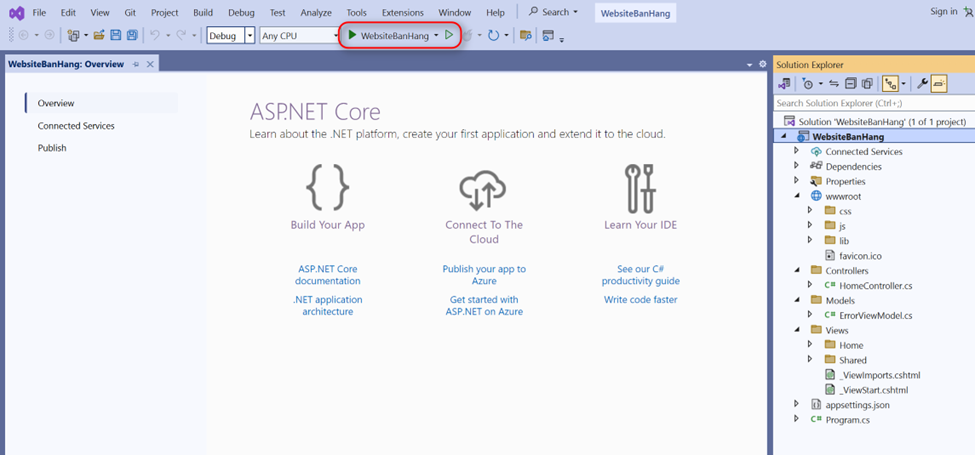
Hình . Đặt tên cho Project



Hình . Thiết lập Framework cho project



Hình . Cấu trúc thư mục cho project



Hình . Tiến hành Run project

## Cấu trúc dự án ASP.NET Core MVC

1. **Dependencies:**

Thư mục này bao gồm các phụ thuộc cần thiết, tức là các gói cần thiết để chạy ứng dụng ASP.NET Core.

1. **Properties:**

Thư mục này chứa tệp launchsettings.json và chỉ được sử dụng trong môi trường phát triển.

1. **wwwroot:**

Đây là thư mục gốc web của dự án. Thư mục wwwroot sẽ chứa tất cả các tệp tĩnh như .css, .js, và các tệp bootstrap, v.v.

1. **Controllers:**

Chứa các lớp điều khiển để xử lý nghiệp vụ trong ứng dụng.

1. **Models:**

Chứa các lớp Model để thao tác với dữ liệu hoặc trong ứng dụng web.

1. **Views:**

Chứa các tệp giao diện Razor (.cshtml) để hiển thị nội dung HTML. Razor là một công cụ giao diện được sử dụng trong ASP.NET để tạo HTML động

1. **Shared:**

Thư mục chứa \_Layout.cshtml. Đó là bố cục mặc định cho ứng dụng ASP.NET Core bao gồm các thành phần dùng chung của tất cả các trang con.

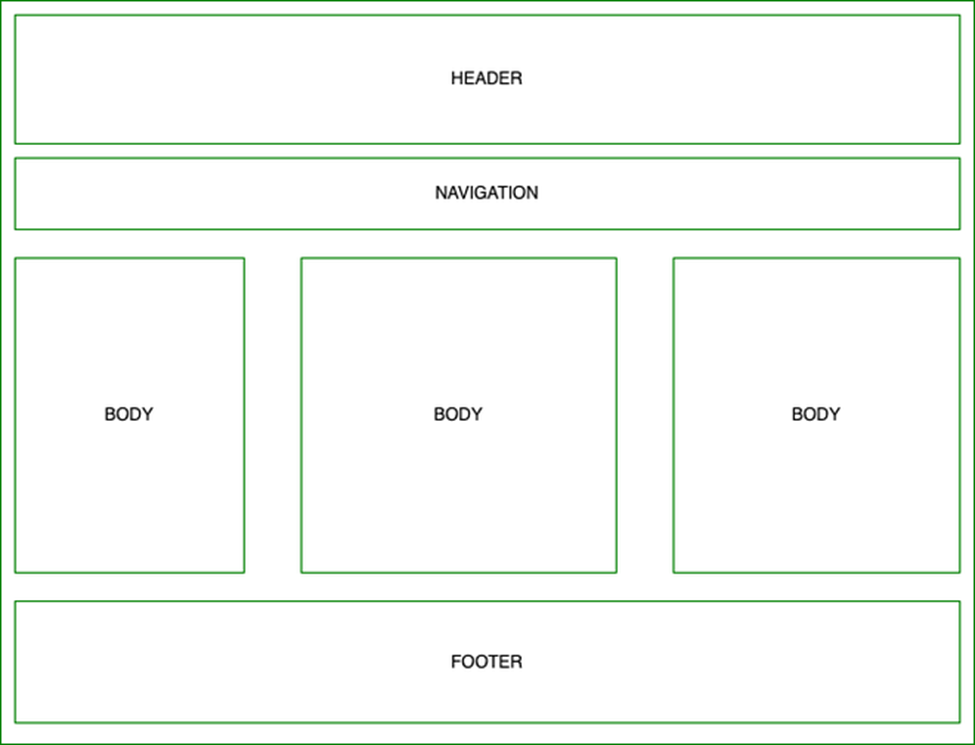
1. **appsettings.json:**

Tệp này sẽ chứa các cài đặt chung trên toàn bộ ứng dụng, như chuỗi kết nối, biến toàn cục phạm vi ứng dụng, v.v..

1. **Program.cs:**

Lớp Program chứa phương thức Main. Phương thức Main chịu trách nhiệm thiết lập máy chủ web, cấu hình các dịch vụ, cấu hình các Thành phần Middleware và khởi động ứng dụng để ứng dụng có thể lắng nghe các yêu cầu từ client.

### \_\_Layout



Hình . Cấu trúc của \_\_Layout

### Razor View Engine



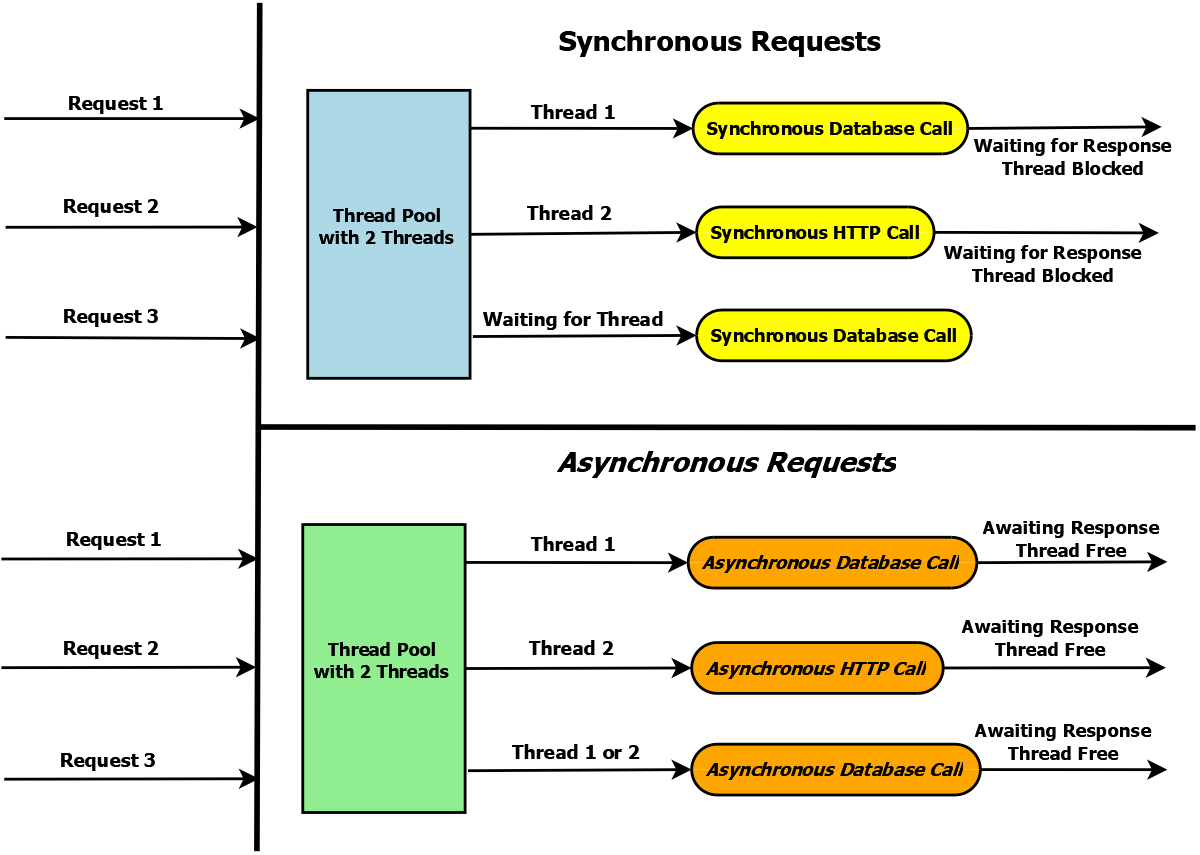
Hình . Mô hình code Razor View Engine

### RenderBody



Hình . Mô hình RenderBody

### RenderSectionAsync



**Hình 2.9 Mô tả RenderSectionAsync**

**Khai báo tại trang \_\_layout**

@await RenderSectionAsync("Scripts", required: true)

**Khai báo tại trang View**

@section Scripts{

}

## Bài tập

### Yêu Cầu Bài Thực Hành: Ứng Dụng Thêm/Đọc/Xóa/Sửa trên ASP.NET Core MVC

#### Mục Đích:

Xây dựng ứng dụng Web với các thao tác CRUD sử dụng ASP.NET Core MVC, áp dụng mô hình **MVC, Model Binding, Data Annotations và sử dụng Repository Pattern với Mock Data.**

#### Yêu Cầu Cụ Thể:

1. Cấu Hình Dự Án và Môi Trường

- Sử dụng ASP.NET Core MVC.

- Khởi tạo dự án mới với ASP.NET Core Web App (**MVC, .NET 6**).

2. Tạo Models

- Tạo `**Product**` và `**Category**` models với các thuộc tính thích hợp.

- Sử dụng Data Annotations cho validation.

3. Repository Pattern

- Tạo `**IProductRepository**` và `**ICategoryRepository**` interfaces.

- Implement các interfaces này với Mock Data.

4. Controllers

- Tạo `**ProductController**` và `CategoryController`.

- Implement các actions: `Add`, `Display`, `Delete`, `Update` cho sản phẩm.

5. Views

- Tạo các views tương ứng cho các chức năng trên.

6. Routing và Navigation

- Cấu hình routing và tạo menu điều hướng.

### Code Mẫu Chi Tiết

* Models

// Product.cs

public class Product

{

public int Id { get; set; }

[Required, StringLength(100)]

public string Name { get; set; }

[Range(0.01, 10000.00)]

public decimal Price { get; set; }

public string Description { get; set; }

public int CategoryId { get; set; }

}

// Category.cs

public class Category

{

public int Id { get; set; }

[Required, StringLength(50)]

public string Name { get; set; }

}

* IProductRepository.cs

using System.Collections.Generic;

using YourNamespace.Models; // Thay thế bằng namespace thực tế của bạn

public interface IProductRepository

{

IEnumerable<Product> GetAll();

Product GetById(int id);

void Add(Product product);

void Update(Product product);

void Delete(int id);

}

* MockProductRepository.cs

using System.Collections.Generic;

using System.Linq;

using YourNamespace.Models; // Thay thế bằng namespace thực tế của bạn

public class MockProductRepository : IProductRepository

{

private readonly List<Product> \_products;

public MockProductRepository()

{

// Tạo một số dữ liệu mẫu

\_products = new List<Product>

{

new Product { Id = 1, Name = "Laptop", Price = 1000, Description = "A high-end laptop"},

// Thêm các sản phẩm khác

};

}

public IEnumerable<Product> GetAll()

{

return \_products;

}

public Product GetById(int id)

{

return \_products.FirstOrDefault(p => p.Id == id);

}

public void Add(Product product)

{

product.Id = \_products.Max(p => p.Id) + 1;

\_products.Add(product);

}

public void Update(Product product)

{

var index = \_products.FindIndex(p => p.Id == product.Id);

if (index != -1)

{

\_products[index] = product;

}

}

public void Delete(int id)

{

var product = \_products.FirstOrDefault(p => p.Id == id);

if (product != null)

{

\_products.Remove(product);

}

}

}

* ICategoryRepository

public interface ICategoryRepository

{

IEnumerable<Category> GetAllCategories();

}

* MockCategoryRepository.cs

public class MockCategoryRepository : ICategoryRepository

{

private List<Category> \_categoryList;

public MockCategoryRepository()

{

\_categoryList = new List<Category>

{

new Category { Id = 1, Name = "Laptop" },

new Category { Id = 2, Name = "Desktop" },

// Thêm các category khác

};

}

public IEnumerable<Category> GetAllCategories()

{

return \_categoryList;

}

}

* ProductController.cs

using Microsoft.AspNetCore.Mvc;

using YourNamespace.Models; // Thay thế bằng namespace thực tế của bạn

using YourNamespace.Repositories; // Thay thế bằng namespace thực tế của bạn

public class ProductController : Controller

{

private readonly IProductRepository \_productRepository;

private readonly ICategoryRepository \_categoryRepository;

public ProductController(IProductRepository productRepository, ICategoryRepository categoryRepository)

{

\_productRepository = productRepository;

\_categoryRepository = categoryRepository;

}

public IActionResult Add()

{

var categories = \_categoryRepository.GetAllCategories();

ViewBag.Categories = new SelectList(categories, "Id", "Name");

return View();

}

[HttpPost]

public IActionResult Add(Product product)

{

if (ModelState.IsValid)

{

\_productRepository.Add(product);

return RedirectToAction("Index"); // Chuyển hướng tới trang danh sách sản phẩm

}

return View(product);

}

// Các actions khác như Display, Update, Delete

// Display a list of products

public IActionResult Index()

{

var products = \_productRepository.GetAll();

return View(products);

}

// Display a single product

public IActionResult Display(int id)

{

var product = \_productRepository.GetById(id);

if (product == null)

{

return NotFound();

}

return View(product);

}

// Show the product update form

public IActionResult Update(int id)

{

var product = \_productRepository.GetById(id);

if (product == null)

{

return NotFound();

}

return View(product);

}

// Process the product update

[HttpPost]

public IActionResult Update(Product product)

{

if (ModelState.IsValid)

{

\_productRepository.Update(product);

return RedirectToAction("Index");

}

return View(product);

}

// Show the product delete confirmation

public IActionResult Delete(int id)

{

var product = \_productRepository.GetById(id);

if (product == null)

{

return NotFound();

}

return View(product);

}

// Process the product deletion

[HttpPost, ActionName("Delete")]

public IActionResult DeleteConfirmed(int id)

{

\_productRepository.Delete(id);

return RedirectToAction("Index");

}

}

* Program.cs

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllersWithViews();

builder.Services.AddSingleton<IProductRepository, MockProductRepository>();

builder.Services.AddScoped<ICategoryRepository, MockCategoryRepository>();

var app = builder.Build();

// Configure the HTTP request pipeline.

if (!app.Environment.IsDevelopment())

{

app.UseExceptionHandler("/Home/Error");

app.UseHsts();

}

app.UseHttpsRedirection();

app.UseStaticFiles();

app.UseRouting();

app.UseAuthorization();

app.MapControllerRoute(

name: "default",

pattern: "{controller=Home}/{action=Index}/{id?}");

app.Run();

* AddProduct.cshtml

@model YourNamespace.Models.Product

@using Microsoft.AspNetCore.Mvc.Rendering

@{

ViewData["Title"] = "Add Product";

}

<h1>Add Product</h1>

<form asp-action="Add">

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

<div class="form-group">

<label asp-for="Name"></label>

<input asp-for="Name" class="form-control" />

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

</div>

<div class="form-group">

<label asp-for="Price"></label>

<input asp-for="Price" class="form-control" />

<span asp-validation-for="Price" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="Description"></label>

<textarea asp-for="Description" class="form-control"></textarea>

<span asp-validation-for="Description" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="CategoryId">Category</label>

<select asp-for="CategoryId" asp-items="ViewBag.Categories" class="form-control"></select>

</div>

<button type="submit" class="btn btn-primary">Add</button>

</form>

***Giải Thích***

* **Index**: Hiển thị danh sách tất cả sản phẩm.
* **Display**: Hiển thị thông tin chi tiết của một sản phẩm cụ thể.
* **Update** (GET): Hiển thị form để cập nhật thông tin sản phẩm.
* **Update** (POST): Xử lý việc cập nhật sản phẩm.
* **Delete** (GET): Hiển thị xác nhận xóa sản phẩm.
* **DeleteConfirmed**: Xử lý việc xóa sản phẩm từ database.
* Đối với mỗi action, bạn sẽ cần tạo các views tương ứng trong thư mục **Views**/**Product**, ví dụ **Index**.cshtml, **Display**.cshtml, **Update**.cshtml, và **Delete**.cshtml.
* Đảm bảo rằng bạn đã đăng ký **IProductRepository** và **implementation** của nó (ví dụ: **MockProductRepository**) trong **Program**.cs của ứng dụng để **Dependency** **Injection** hoạt động đúng cách.
* Index.cshtml (Hiển Thị Danh Sách Sản Phẩm)

@model IEnumerable<YourNamespace.Models.Product>

<h2>Products</h2>

<table class="table">

<thead>

<tr>

<th>Name</th>

<th>Price</th>

<th>Description</th>

<th>Actions</th>

</tr>

</thead>

<tbody>

@foreach (var product in Model)

{

<tr>

<td>@product.Name</td>

<td>@product.Price</td>

<td>@product.Description</td>

<td>

<a asp-action="Display" asp-route-id="@product.Id">View</a> |

<a asp-action="Update" asp-route-id="@product.Id">Edit</a> |

<a asp-action="Delete" asp-route-id="@product.Id">Delete</a>

</td>

</tr>

}

</tbody>

</table>

* Display.cshtml (Hiển Thị Thông Tin Chi Tiết Sản Phẩm)

@model YourNamespace.Models.Product

<h2>Product Details</h2>

<div>

<h4>Name: @Model.Name</h4>

<h4>Price: @Model.Price</h4>

<h4>Description: @Model.Description</h4>

</div>

<a asp-action="Index">Back to List</a>

* Delete.cshtml (Xác Nhận Xóa Sản Phẩm)

@model YourNamespace.Models.Product

<h2>Are you sure you want to delete this?</h2>

<div>

<h4>Name: @Model.Name</h4>

<h4>Price: @Model.Price</h4>

<h4>Description: @Model.Description</h4>

</div>

<form asp-action="DeleteConfirmed" method="post">

<input type="hidden" asp-for="Id" />

<input type="submit" value="Delete" class="btn btn-danger" /> |

<a asp-action="Index">Cancel</a>

</form>

* Update.cshtml (Cập Nhật Sản Phẩm)

@model YourNamespace.Models.Product

<h2>Edit Product</h2>

<form asp-action="Update">

<input type="hidden" asp-for="Id" />

<div class="form-group">

<label asp-for="Name"></label>

<input asp-for="Name" class="form-control" />

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

</div>

<div class="form-group">

<label asp-for="Price"></label>

<input asp-for="Price" class="form-control" />

<span asp-validation-for="Price" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="Description"></label>

<textarea asp-for="Description" class="form-control"></textarea>

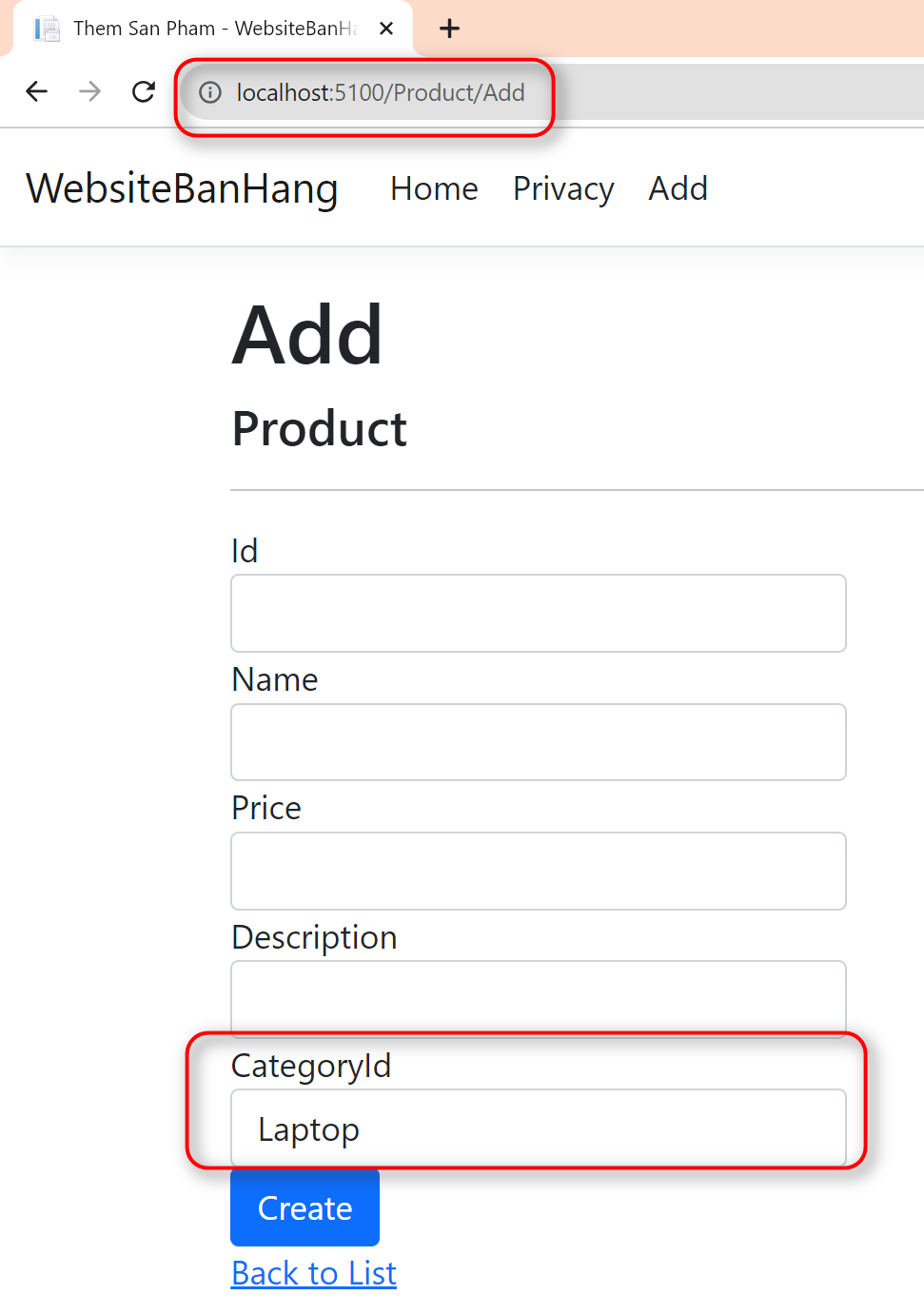
<span asp-validation-for="Description" class="text-danger"></span>

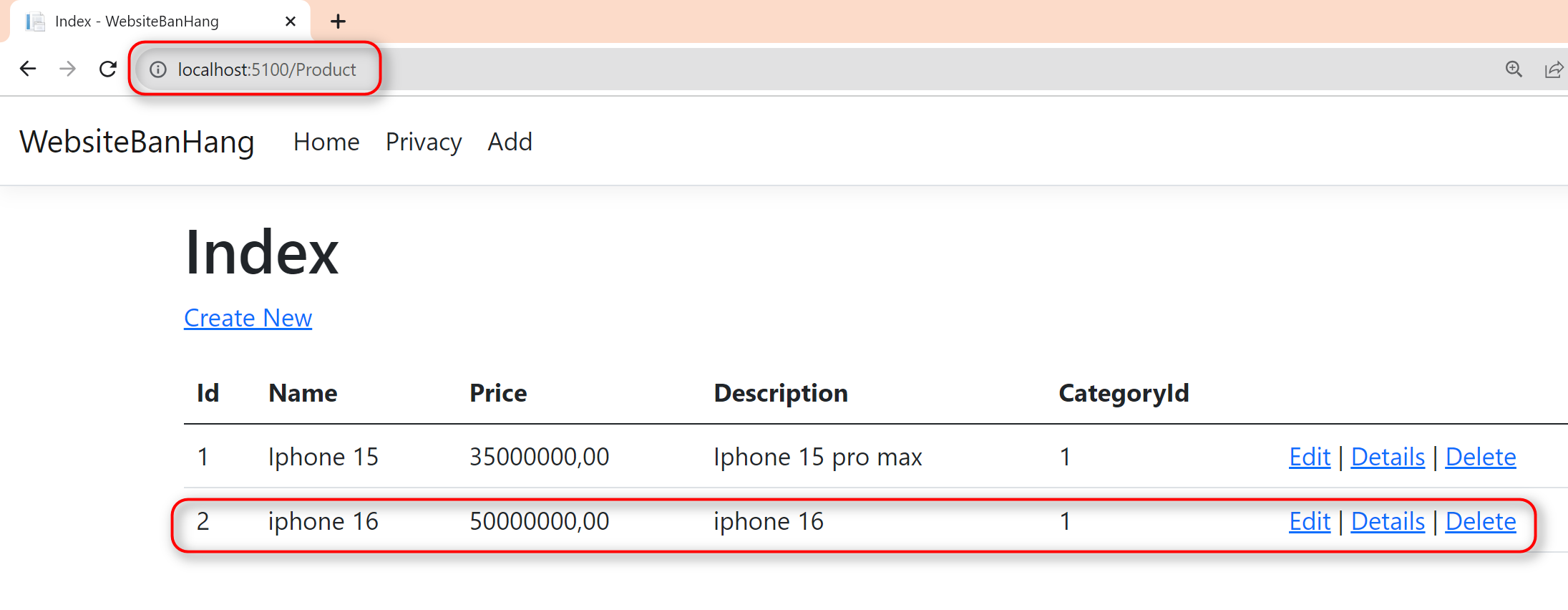
</div>

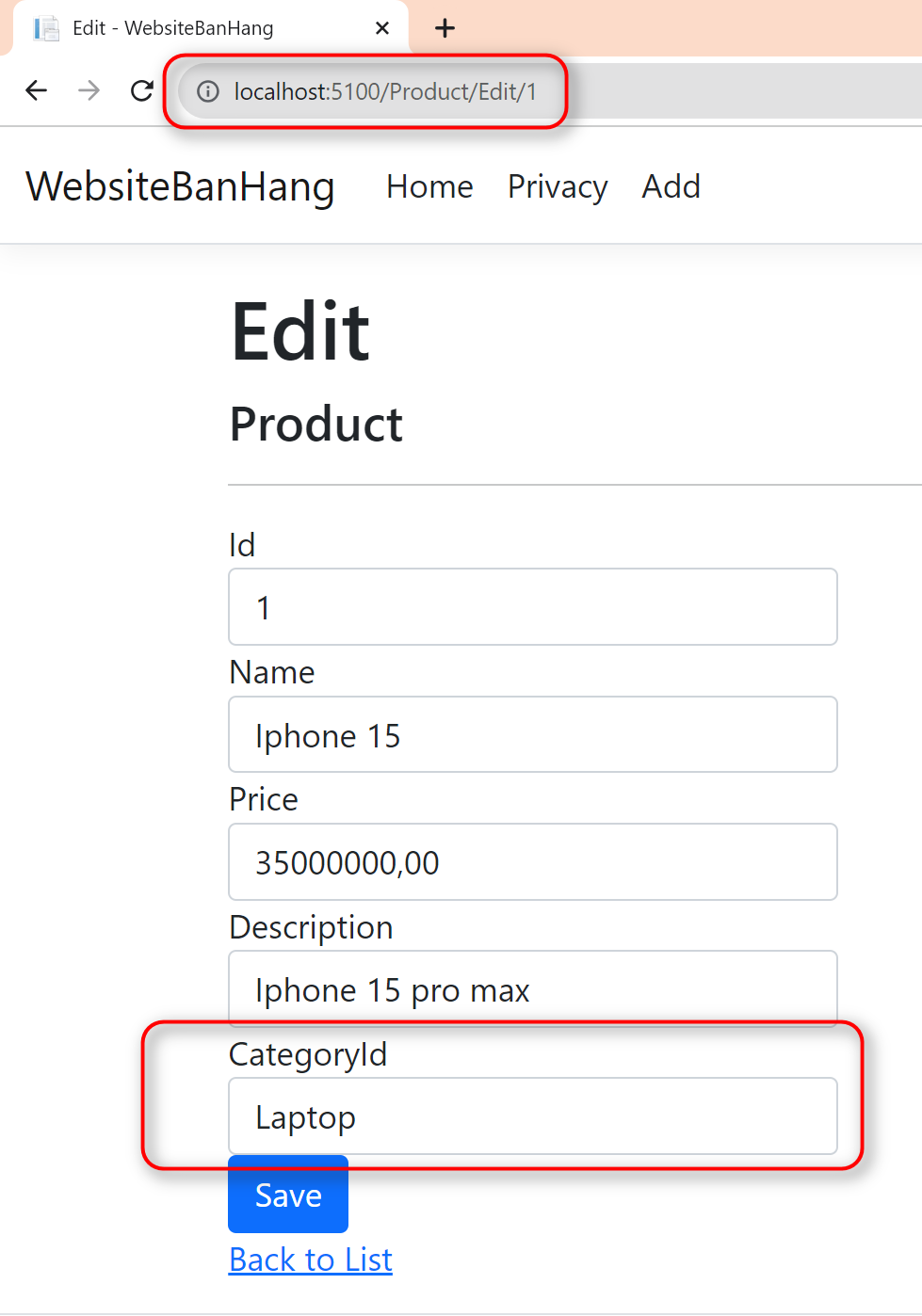
<button type="submit" class="btn btn-primary">Update</button>

</form>

### Kết quả







### Bổ sung tính năng upload file cho ứng dụng

Để bổ sung tính năng thêm hình ảnh đại diện và nhiều hình ảnh cho sản phẩm trong ứng dụng trên, bạn cần mở rộng model `Product`, cập nhật views và controllers để xử lý việc upload hình ảnh. Dưới đây là các bước cơ bản để thực hiện điều này:

* Mở Rộng Model `Product`

Mở rộng model `Product` để bao gồm các thuộc tính cho hình ảnh đại diện và danh sách các hình ảnh:

public class Product

{

// Các thuộc tính hiện có

public string? ImageUrl { get; set; } // Đường dẫn đến hình ảnh đại diện

public List<string>? ImageUrls { get; set; } // Danh sách các hình ảnh khác

}

* Cập Nhật View `AddProduct.cshtml`

Cập nhật view `AddProduct.cshtml` để thêm fields cho việc upload hình ảnh:

@model YourNamespace.Models.Product

<!-- Phần còn lại của form -->

<div class="form-group">

<label asp-for="ImageUrl">Image</label>

<input type="file" asp-for="ImageUrl" class="form-control" />

</div>

<div class="form-group">

<label>Additional Images</label>

<input type="file" **asp-for**="ImageUrls" multiple class="form-control" />

</div>

<button type="submit" class="btn btn-primary">Add</button>

* Cập nhập thêm vào form Add Product

<form asp-action="Add" enctype="multipart/form-data"> ...... <form>

* Xử Lý Upload Trong Controller

Cập nhật `ProductController` để xử lý việc upload hình ảnh:

[HttpPost]

public async Task<IActionResult> Add(Product product, IFormFile imageUrl, List<IFormFile> imageUrls)

{

if (ModelState.IsValid)

{

if (imageUrl!= null)

{

// Lưu hình ảnh đại diện

product.ImageUrl = await SaveImage(imageUrl);

}

if (imageUrls!= null)

{

product.ImageUrls = new List<string>();

foreach (var file in imageUrl)

{

// Lưu các hình ảnh khác

product.ImageUrls.Add(await SaveImage(file));

}

}

\_productRepository.Add(product);

return RedirectToAction("Index");

}

return View(product);

}

private async Task<string> SaveImage(IFormFile image)

{

var savePath = Path.Combine("wwwroot/images", image.FileName); // Thay đổi đường dẫn theo cấu hình của bạn

using (var fileStream = new FileStream(savePath, FileMode.Create))

{

await image.CopyToAsync(fileStream);

}

return "/images/" + image.FileName; // Trả về đường dẫn tương đối

}

* Cấu Hình `Program.cs`

Đảm bảo rằng ứng dụng của bạn cấu hình đúng cách để phục vụ các tệp tĩnh:

app.UseStaticFiles(); // Cho phép ứng dụng phục vụ các tệp tĩnh từ thư mục wwwroot

* Hiển Thị Hình Ảnh

Cập nhật các views cần thiết để hiển thị hình ảnh của sản phẩm. Ví dụ, trong `Display.cshtml`:

@model YourNamespace.Models.Product

<h2>@Model.Name</h2>

<img src="@Model.ImageUrl" alt="Product Image" />

@foreach (var imageUrl in Model.ImageUrls)

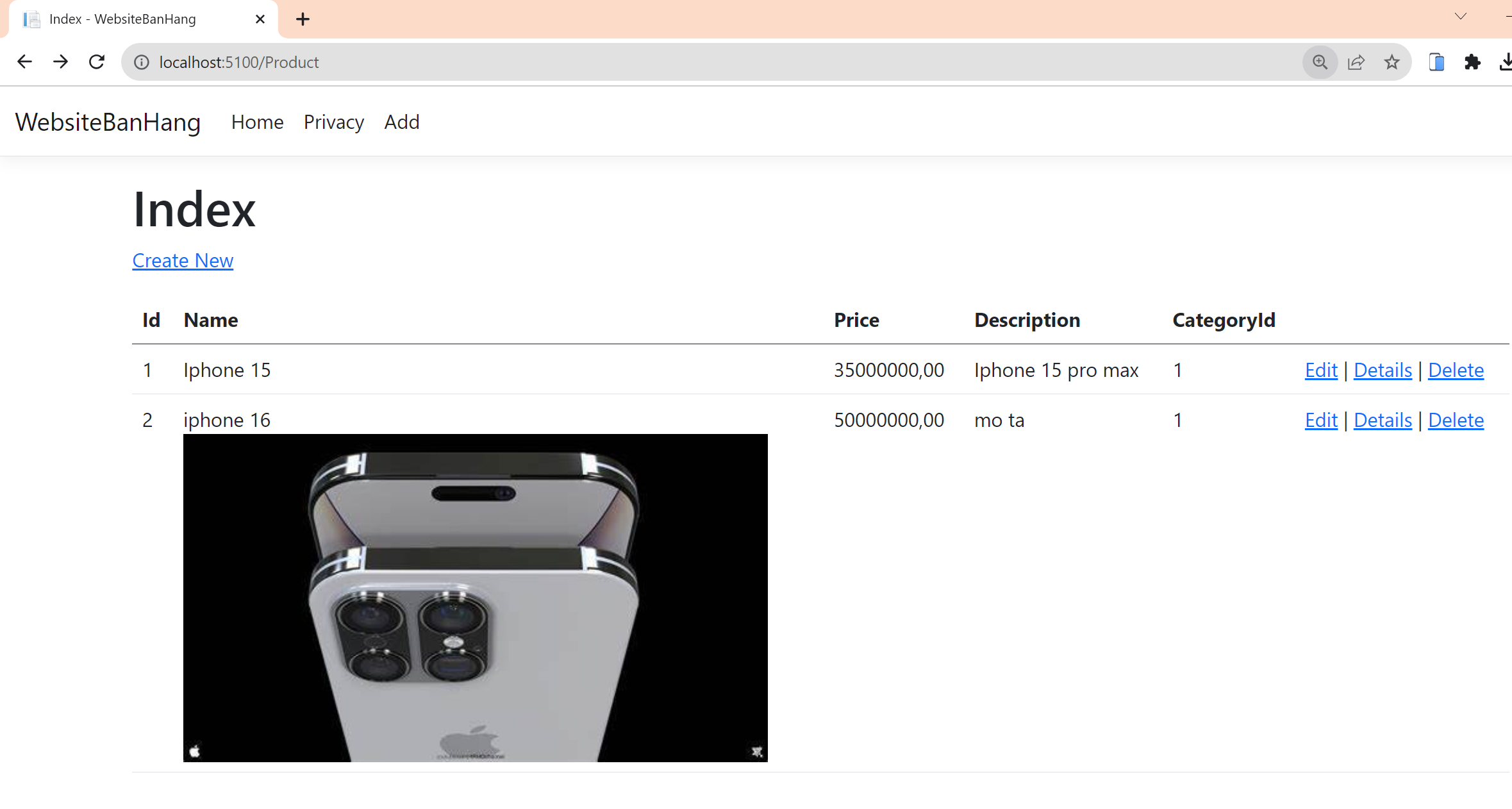
{

<img src="@imageUrl" alt="Product Image" />

}

**Kết quả**





**Lưu ý:** Thêm kiểm tra và xử lý lỗi cho các tình huống như tệp không phải hình ảnh được tải lên hoặc kích thước tệp quá lớn.

### Yêu cầu bổ sung

Áp dụng giao diện đã thiết kế ở LAB 1 cho ứng dụng website bán hàng

# XÂY DỰNG ỨNG DỤNG WEBSITE BÁN HÀNG VỚI ASP.NET CORE MVC (PHẦN 1)

Sau khi học xong bài này, sinh viên có thể nắm được:

* Học cách sử dụng Entity Framework Core để tương tác với CSDL MS SQL Server.
* Xây dựng ứng dụng web bán hàng dựa trên ASP.NET Core MVC, kết hợp với cơ sở dữ liệu và Entity Framework Core.

## Bài tập

### Yêu cầu

Xây dựng web bán hàng có các chức năng Hiển thị/ Thêm/ Xóa/ Sửa với Entity Framework Core và Cơ sở dữ liệu MS SQL Server

### Hướng dẫn thực hiện

* Cài Đặt Gói NuGet Cần Thiết

Cài đặt các gói sau bằng NuGet:

* ‘ Microsoft.EntityFrameworkCore`
* `Microsoft.EntityFrameworkCore.SqlServer` (hoặc provider cơ sở dữ liệu khác)
* `Microsoft.EntityFrameworkCore.Tools`
* Thiết Kế Models

Tạo các models `Product` và `Category`:

public class Product

{

public int Id { get; set; }

[Required, StringLength(100)]

public string Name { get; set; }

[Range(0.01, 10000.00)]

public decimal Price { get; set; }

public string Description { get; set; }

public string? ImageUrl { get; set; }

public List<ProductImage>? Images { get; set; }

public int CategoryId { get; set; }

public Category? Category { get; set; }

}

public class Category

{

public int Id { get; set; }

[Required, StringLength(50)]

public string Name { get; set; }

public List<Product>? Products { get; set; }

}

public class ProductImage

{

public int Id { get; set; }

public string Url { get; set; }

public int ProductId { get; set; }

public Product? Product { get; set; }

}

* Cấu Hình Entity Framework Core

Tạo một lớp `ApplicationDbContext` và cấu hình:

using Microsoft.EntityFrameworkCore;

public class ApplicationDbContext : DbContext

{

Public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options) : base(options)

{

}

public DbSet<Product> Products { get; set; }

public DbSet<Category> Categories { get; set; }

public DbSet<ProductImage> ProductImages { get; set; }

}

* Cấu hình EF Core trong `Program.cs`:

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddDbContext<ApplicationDbContext>(options =>

options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection")));

// Các cấu hình khác

```

* Cấu hình connection string trong `appsettings.json`.

{

"ConnectionStrings": {

"DefaultConnection": "Server=(localdb)\\mssqllocaldb;Database=YourDbName;Trusted\_Connection=True;MultipleActiveResultSets=true"

},

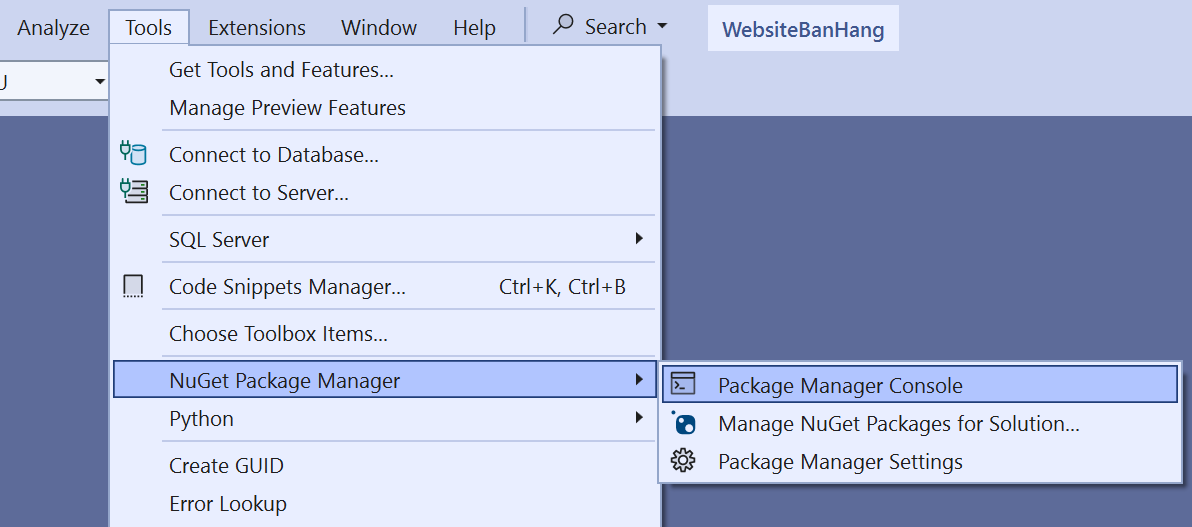
// Các cấu hình khác

}

* Tạo Migrations

Cài đặt dotnet-ef tool bằng cách mở Command Line

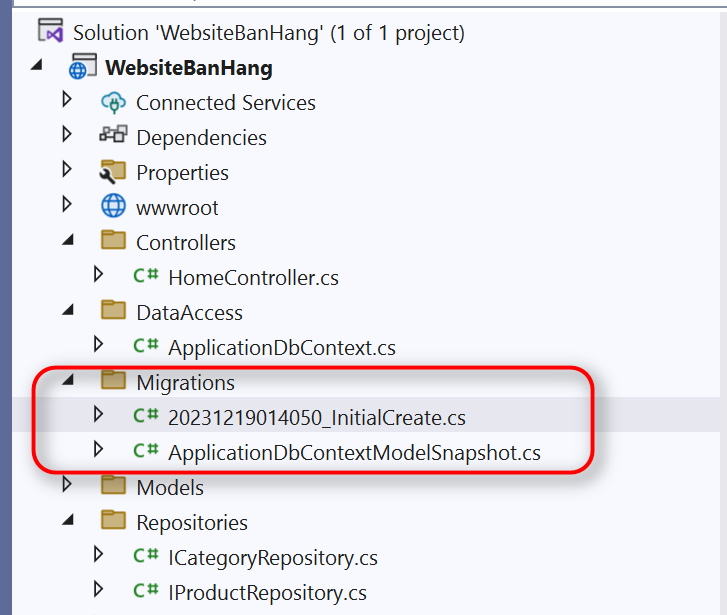


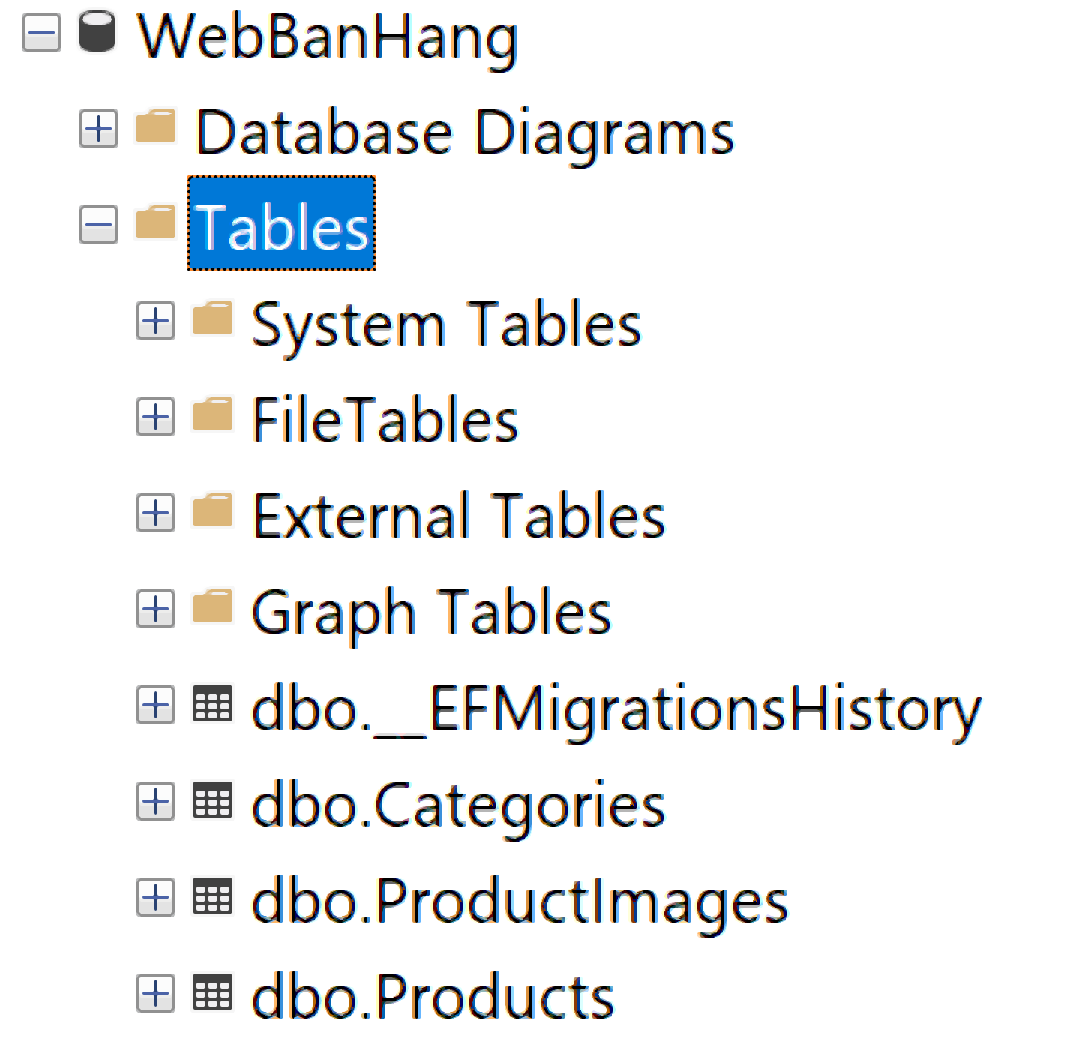


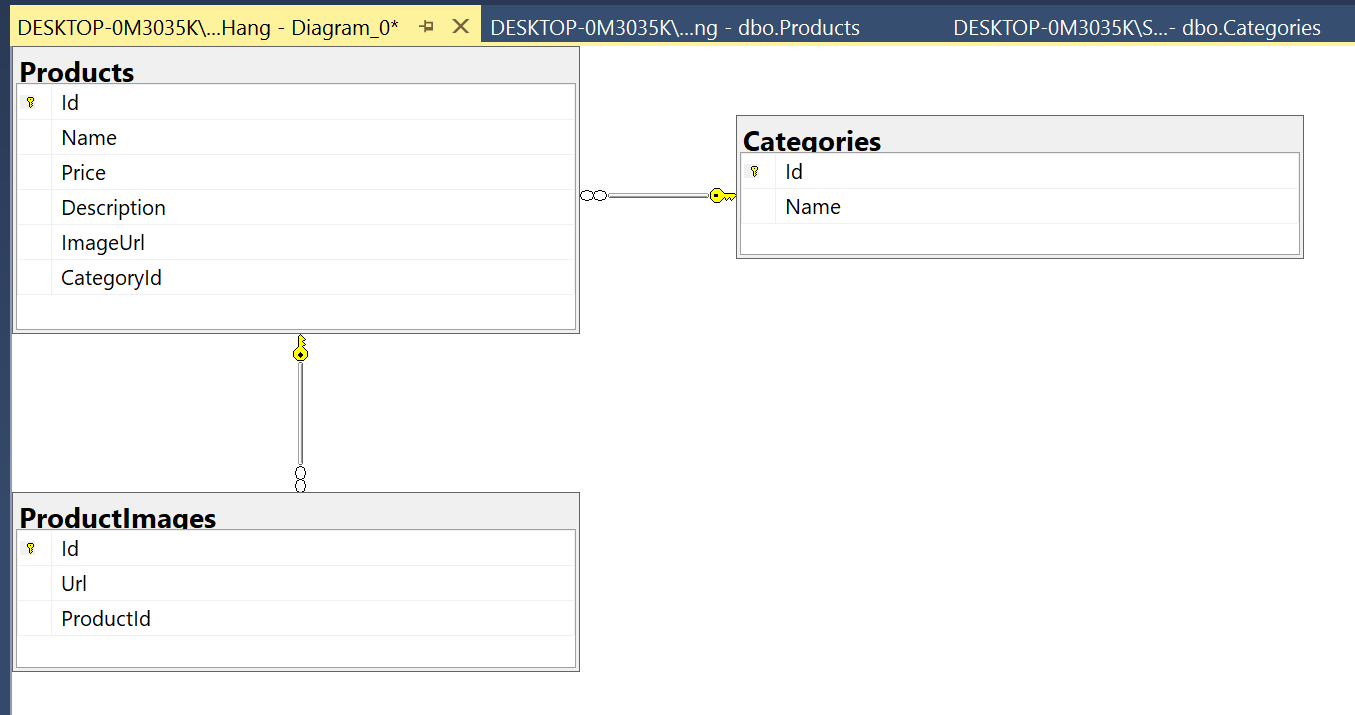
Tạo migrations để tạo cơ sở dữ liệu:

dotnet ef migrations add InitialCreate --project WebsiteBanHang

dotnet ef database update --project WebsiteBanHang







* IProductRepository và ICategoryRepository

Đây là các interface định nghĩa các phương thức cần thiết để tương tác với cơ sở dữ liệu cho Product và Category.

public interface IProductRepository

{

Task<IEnumerable<Product>> GetAllAsync();

Task<Product> GetByIdAsync(int id);

Task AddAsync(Product product);

Task UpdateAsync(Product product);

Task DeleteAsync(int id);

}

public interface ICategoryRepository

{

Task<IEnumerable<Category>> GetAllAsync();

Task<Category> GetByIdAsync(int id);

Task AddAsync(Category category);

Task UpdateAsync(Category category);

Task DeleteAsync(int id);

}

* EFProductRepository và EFCategoryRepository

public class EFProductRepository : IProductRepository

{

private readonly ApplicationDbContext \_context;

public EFProductRepository(ApplicationDbContext context)

{

\_context = context;

}

public async Task<IEnumerable<Product>> GetAllAsync()

{

return await \_context.Products.ToListAsync();

}

public async Task<Product> GetByIdAsync(int id)

{

return await \_context.Products.FindAsync(id);

}

public async Task AddAsync(Product product)

{

\_context.Products.Add(product);

await \_context.SaveChangesAsync();

}

public async Task UpdateAsync(Product product)

{

\_context.Products.Update(product);

await \_context.SaveChangesAsync();

}

public async Task DeleteAsync(int id)

{

var product = await \_context.Products.FindAsync(id);

\_context.Products.Remove(product);

await \_context.SaveChangesAsync();

}

}

public class EFCategoryRepository : ICategoryRepository

{

private readonly ApplicationDbContext \_context;

public EFCategoryRepository(ApplicationDbContext context)

{

\_context = context;

}

// Tương tự như EFProductRepository, nhưng cho Category

}

* Đăng Ký trong Program.cs

builder.Services.AddScoped<IProductRepository, EFProductRepository>();

builder.Services.AddScoped<ICategoryRepository, EFCategoryRepository>();

* Tạo và Cập Nhật Controllers

Sử Dụng Repository trong Controllers

using Microsoft.AspNetCore.Mvc;

using YourNamespace.Models; // Thay thế bằng namespace thực tế của bạn

using YourNamespace.Repositories; // Thay thế bằng namespace thực tế của bạn

public class ProductController : Controller

{

private readonly IProductRepository \_productRepository;

private readonly ICategoryRepository \_categoryRepository;

public ProductController(IProductRepository productRepository, ICategoryRepository categoryRepository)

{

\_productRepository = productRepository;

\_categoryRepository = categoryRepository;

}

// Hiển thị danh sách sản phẩm

public async Task<IActionResult> Index()

{

var products = await \_productRepository.GetAllAsync();

return View(products);

}

// Hiển thị form thêm sản phẩm mới

public IActionResult Add()

{

var categories = await \_categoryRepository.GetAllAsync();

ViewBag.Categories = new SelectList(categories, "Id", "Name");

return View();

}

// Xử lý thêm sản phẩm mới

[HttpPost]

public async Task<IActionResult> Add(Product product)

{

if (ModelState.IsValid)

{

await \_productRepository.AddAsync(product);

return RedirectToAction(nameof(Index));

}

// Nếu ModelState không hợp lệ, hiển thị form với dữ liệu đã nhập

var categories = await \_categoryRepository.GetAllAsync();

ViewBag.Categories = new SelectList(categories, "Id", "Name");

return View(product);

}

// Hiển thị thông tin chi tiết sản phẩm

public async Task<IActionResult> Display(int id)

{

var product = await \_productRepository.GetByIdAsync(id);

if (product == null)

{

return NotFound();

}

return View(product);

}

// Hiển thị form cập nhật sản phẩm

public async Task<IActionResult> Update(int id)

{

var product = await \_productRepository.GetByIdAsync(id);

if (product == null)

{

return NotFound();

}

var categories = await \_categoryRepository.GetAllAsync();

ViewBag.Categories = new SelectList(categories, "Id", "Name", product.CategoryId);

return View(product);

}

// Xử lý cập nhật sản phẩm

[HttpPost]

public async Task<IActionResult> Update(int id, Product product)

{

if (id != product.Id)

{

return NotFound();

}

if (ModelState.IsValid)

{

await \_productRepository.UpdateAsync(product);

return RedirectToAction(nameof(Index));

}

return View(product);

}

// Hiển thị form xác nhận xóa sản phẩm

public async Task<IActionResult> Delete(int id)

{

var product = await \_productRepository.GetByIdAsync(id);

if (product == null)

{

return NotFound();

}

return View(product);

}

// Xử lý xóa sản phẩm

[HttpPost, ActionName("Delete")]

public async Task<IActionResult> DeleteConfirmed(int id)

{

await \_productRepository.DeleteAsync(id);

return RedirectToAction(nameof(Index));

}

}

* Tạo và Cập nhật Views

Xây dựng các view của `ProductController` trong ASP.NET Core MVC, bao gồm các trang hiển thị danh sách sản phẩm, thêm sản phẩm mới, cập nhật thông tin sản phẩm, và xóa sản phẩm.

* Index.cshtml (Danh sách sản phẩm)

@model IEnumerable<YourNamespace.Models.Product>

<h2>Products</h2>

<table class="table">

<thead>

<tr>

<th>Name</th>

<th>Price</th>

<th>Description</th>

<th>Category</th>

<th>Actions</th>

</tr>

</thead>

<tbody>

@foreach (var product in Model)

{

<tr>

<td>@product.Name</td>

<td>@product.Price</td>

<td>@product.Description</td>

<td>@product.Category?.Name ?? "No Category"</td>

<td>

<a asp-action="Display" asp-route-id="@product.Id">View</a> |

<a asp-action="Update" asp-route-id="@product.Id">Edit</a> |

<a asp-action="Delete" asp-route-id="@product.Id">Delete</a>

</td>

</tr>

}

</tbody>

</table>

* Add.cshtml (Thêm sản phẩm mới)

@model YourNamespace.Models.Product

@using Microsoft.AspNetCore.Mvc.Rendering

@{

ViewData["Title"] = "Add Product";

}

<h2>Add Product</h2>

<form asp-action="Add" method="post" enctype="multipart/form-data">

<div class="form-group">

<label asp-for="Name"></label>

<input asp-for="Name" class="form-control" />

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

</div>

<div class="form-group">

<label asp-for="Price"></label>

<input asp-for="Price" class="form-control" />

<span asp-validation-for="Price" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="Description"></label>

<textarea asp-for="Description" class="form-control"></textarea>

<span asp-validation-for="Description" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="CategoryId">Category</label>

<select asp-for="CategoryId" asp-items="ViewBag.Categories" class="form-control"></select>

</div>

<div class="form-group">

<label asp-for="ImageUrl">Product Image</label>

<input type="file" asp-for="ImageUrl" class="form-control" />

</div>

<button type="submit" class="btn btn-primary">Add</button>

</form>

* Display.cshtml (Hiển thị thông tin chi tiết sản phẩm)

@model YourNamespace.Models.Product

<h2>@Model.Name</h2>

<div>

<h3>Price: @Model.Price</h3>

<p>@Model.Description</p>

@if (!string.IsNullOrEmpty(Model.ImageUrl))

{

<img src="@Model.ImageUrl" alt="Product Image" style="max-width: 200px;"/>

}

</div>

<a asp-action="Index">Back to List</a>

* Update.cshtml (Cập nhật thông tin sản phẩm)

@model YourNamespace.Models.Product

@using Microsoft.AspNetCore.Mvc.Rendering

@{

ViewData["Title"] = "Edit Product";

}

<h2>Edit Product</h2>

<form asp-action="Update" method="post" enctype="multipart/form-data">

<input type="hidden" asp-for="Id" />

<div class="form-group">

<label asp-for="Name"></label>

<input asp-for="Name" class="form-control" />

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

</div>

<div class="form-group">

<label asp-for="Price"></label>

<input asp-for="Price" class="form-control" />

<span asp-validation-for="Price" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="Description"></label>

<textarea asp-for="Description" class="form-control"></textarea>

<span asp-validation-for="Description" class="text-danger"></span>

</div>

<div class="form-group">

<label asp-for="CategoryId">Category</label>

<select asp-for="CategoryId" asp-items="@ViewBag.Categories" class="form-control"></select>

</div>

<div class="form-group">

<label asp-for="ImageUrl">Product Image</label>

<input type="file" asp-for="ImageUrl" class="form-control" />

<img src="@Model.ImageUrl" alt="Product Image" style="max-width: 200px;"/>

</div>

<button type="submit" class="btn btn-primary">Update</button>

</form>

* Delete.cshtml (Xác nhận xóa sản phẩm)

@model YourNamespace.Models.Product

<h2>Delete Product</h2>

<form asp-action="DeleteConfirmed" method="post">

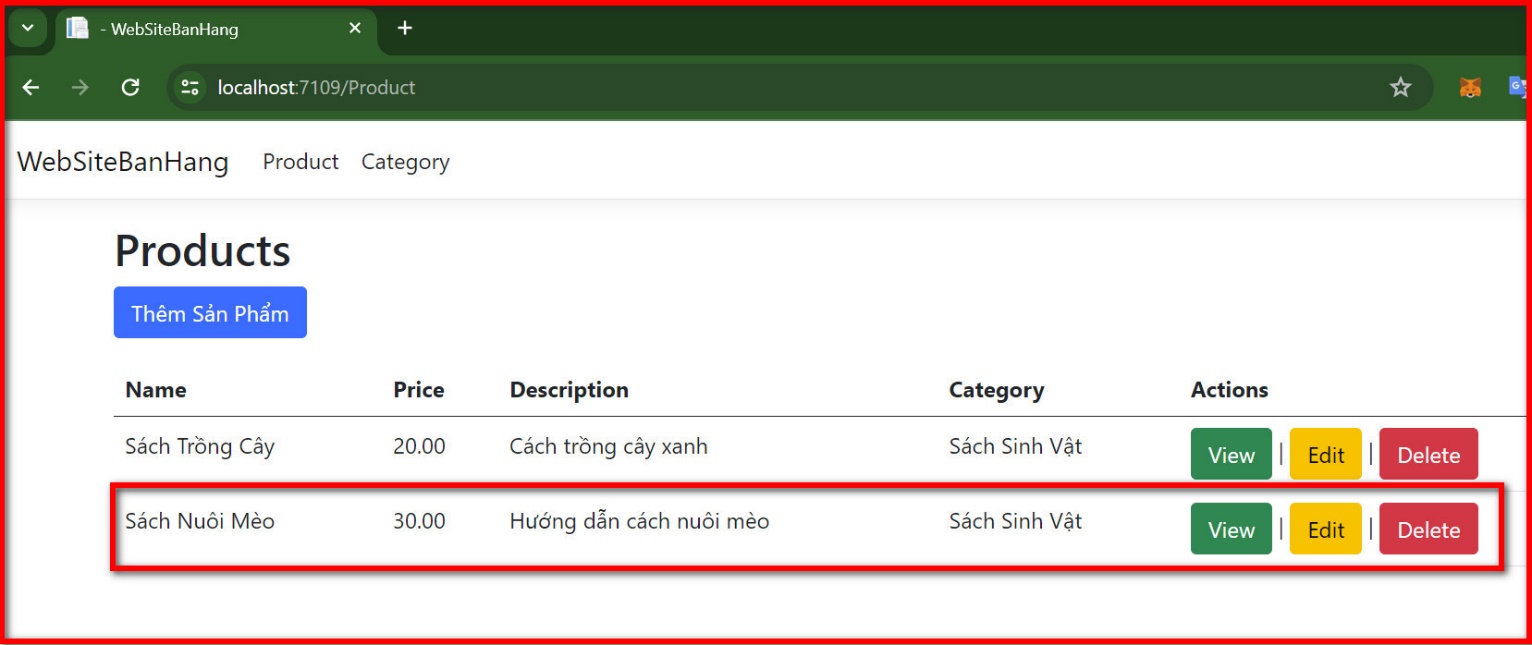
<input type="hidden" asp-for="Id" />

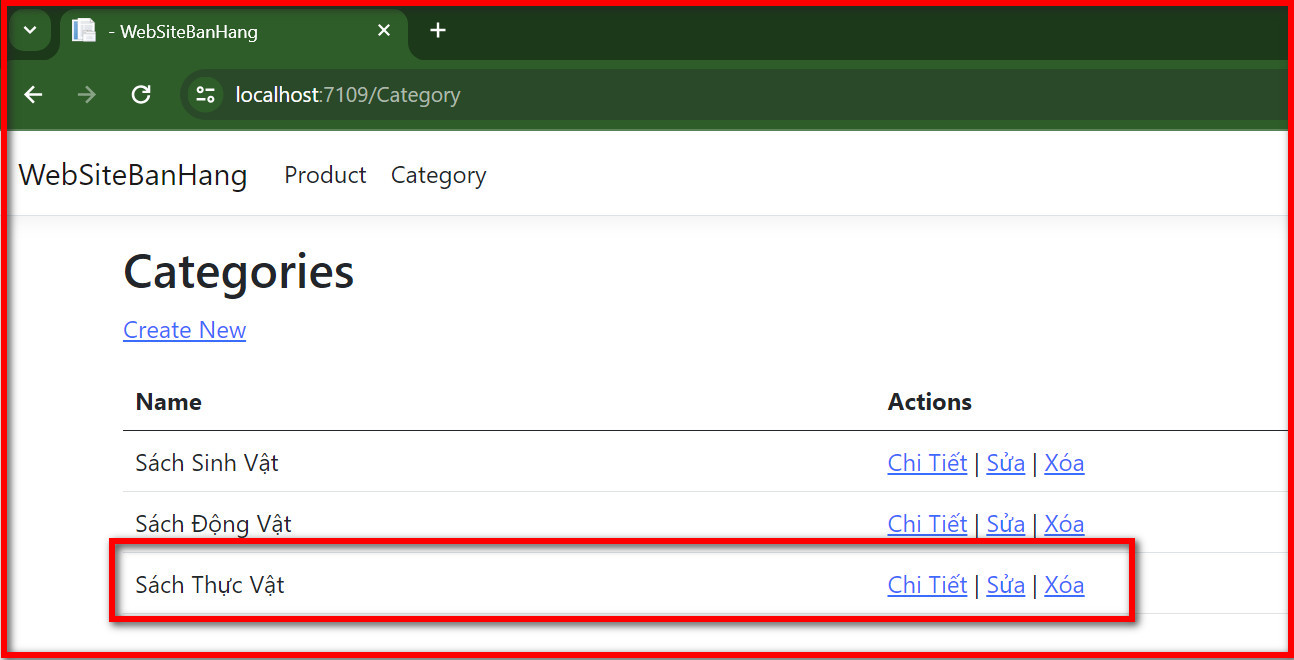
<p>Are you sure you want to delete @Model.Name?</p>

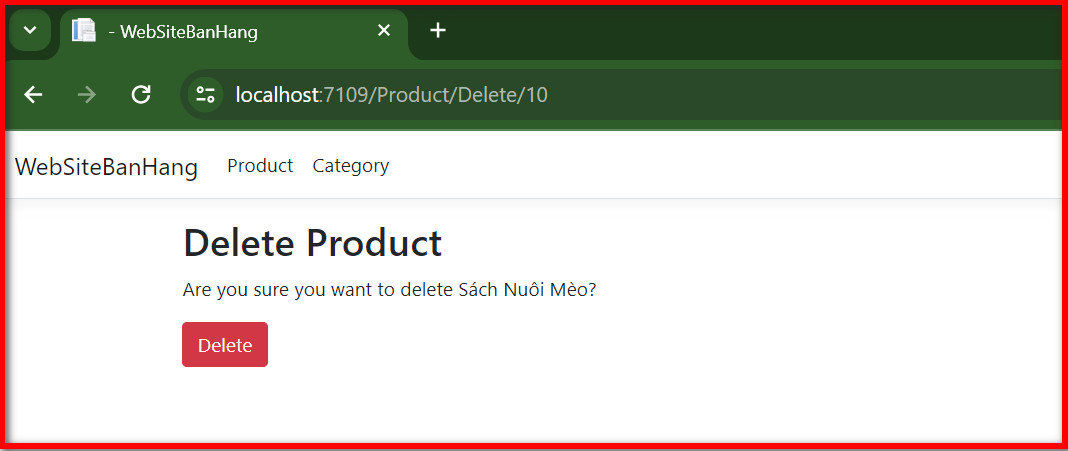
<button type="submit" class="btn btn-danger">Delete</button>

</form>

### Kết quả







### Yêu cầu bổ sung

Thực hiện cho các chức năng **CRUD** tương tự cho **Category**

***Lưu ý:***

* Thay thế ‘YourNamespace.Models.Product` với namespace thực tế của model `Product` trong dự án của bạn.
* Đảm bảo rằng `ViewBag.Categories` được đúng cách gán giá trị trong controller trước khi gửi đến `Add.cshtml` và `Update.cshtml`.
* Thêm logic để xử lý đường dẫn hình ảnh trong `Display.cshtml` và `Update.cshtml` tùy theo cách bạn lưu trữ và truy xuất hình ảnh.

# XÂY DỰNG ỨNG DỤNG WEB BÁN HÀNG VỚI ASP.NET CORE MVC (PHẦN 2)

Sau khi học xong bài này, sinh viên có thể nắm được:

* Sử dụng Identity Core để thực hiện các chức năng quản lý người dùng trong ứng dụng
* Kỹ năng sử dụng Area trong ASP.NET Core MVC để phân tách và tổ chức các chức năng quản trị ra khỏi phần còn lại của ứng dụng web. Điều này giúp tăng cường bảo mật và rõ ràng trong cấu trúc ứng dụng.
* Tạo khu vực Admin để chứa các chức năng như quản lý người dùng, cài đặt hệ thống, báo cáo, và các công cụ quản trị khác.

## Bài thực hành

### Yêu cầu

Tích hợp Identity Core vào ứng dụng web bán hàng ở bài thực hành LAB 2 để quản lý người dùng.

Thiết lập một Area Admin trong ứng dụng web bán hàng để tách biệt chức năng dành cho người quản trị.

## Hướng dẫn thực hiện

### Cấu Hình ASP.NET Core Identity

Cấu hình ASP.NET Core Identity trong `ApplicationDbContext` và `Program.cs`:

* ApplicationDbContext.cs

using Microsoft.AspNetCore.Identity.EntityFrameworkCore;

using Microsoft.EntityFrameworkCore;

using YourNamespace.Models; // Thay thế bằng namespace thực tế của bạn

public class ApplicationDbContext : IdentityDbContext

{

public ApplicationDbContext(DbContextOptions<ApplicationDbContext> options)

: base(options)

{

}

public DbSet<Product> Products { get; set; }

public DbSet<Category> Categories { get; set; }

// Các DbSet khác nếu cần

}

* Program.cs

using Microsoft.AspNetCore.Identity;

using Microsoft.EntityFrameworkCore;

using WebsiteBanHang.DataAccess;

using WebsiteBanHang.Repositories;

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllersWithViews();

builder.Services.AddDbContext<ApplicationDbContext>(options =>

options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection")));

builder.Services.AddIdentity<IdentityUser, IdentityRole>()

.AddDefaultTokenProviders()

.AddDefaultUI()

.AddEntityFrameworkStores<ApplicationDbContext>();

builder.Services.AddRazorPages();

builder.Services.AddScoped<IProductRepository, EFProductRepository>();

builder.Services.AddScoped<ICategoryRepository, EFCategoryRepository>();

var app = builder.Build();

// Configure the HTTP request pipeline.

if (!app.Environment.IsDevelopment())

{

app.UseExceptionHandler("/Home/Error");

}

app.UseStaticFiles();

app.UseRouting();

app.UseAuthentication();;

app.UseAuthorization();

app.MapRazorPages();

app.MapControllerRoute(

name: "default",

pattern: "{controller=Home}/{action=Index}/{id?}");

app.Run();

* Cập nhật Database

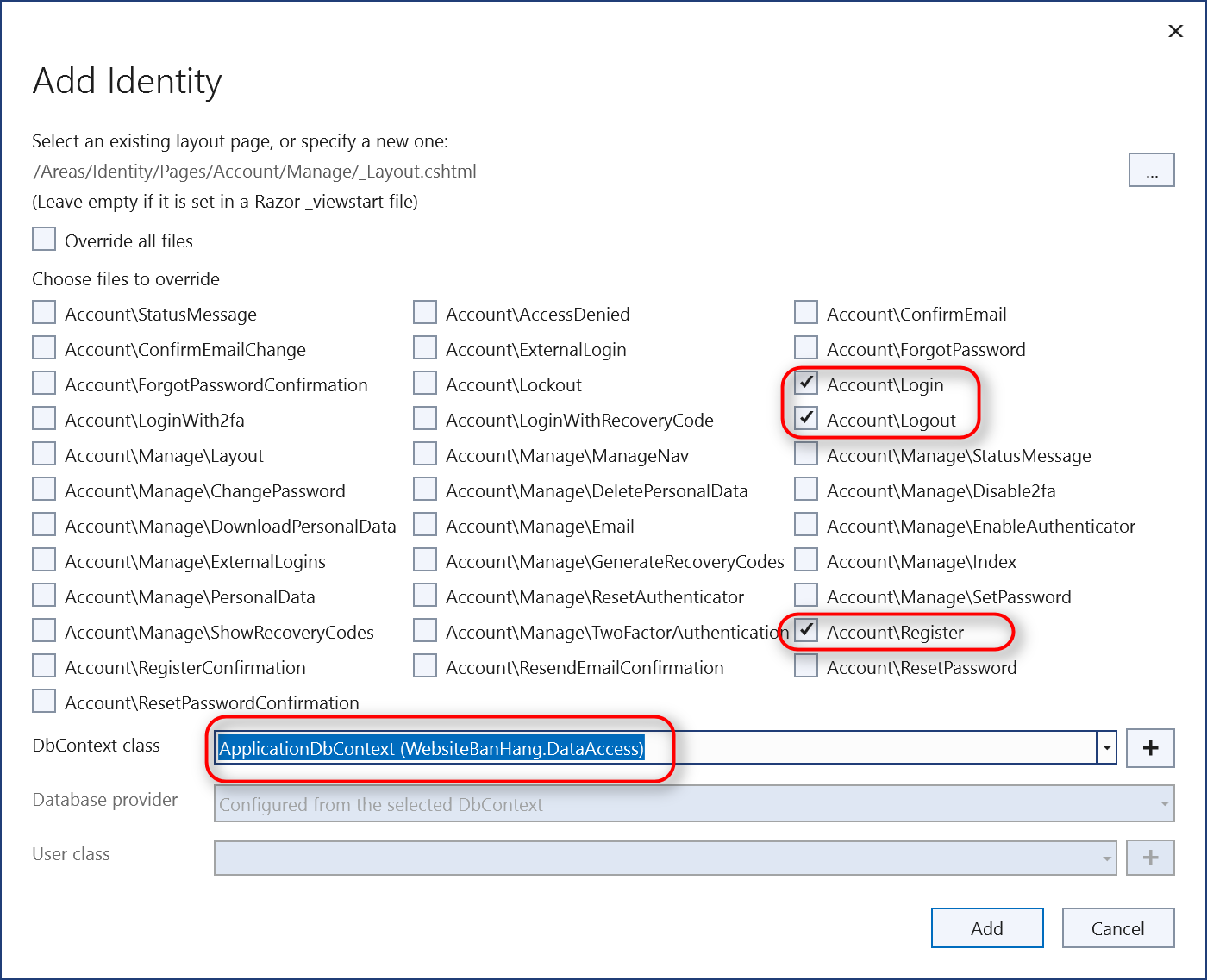
dotnet ef migrations add InitialCreateIdentity --project WebsiteBanHang

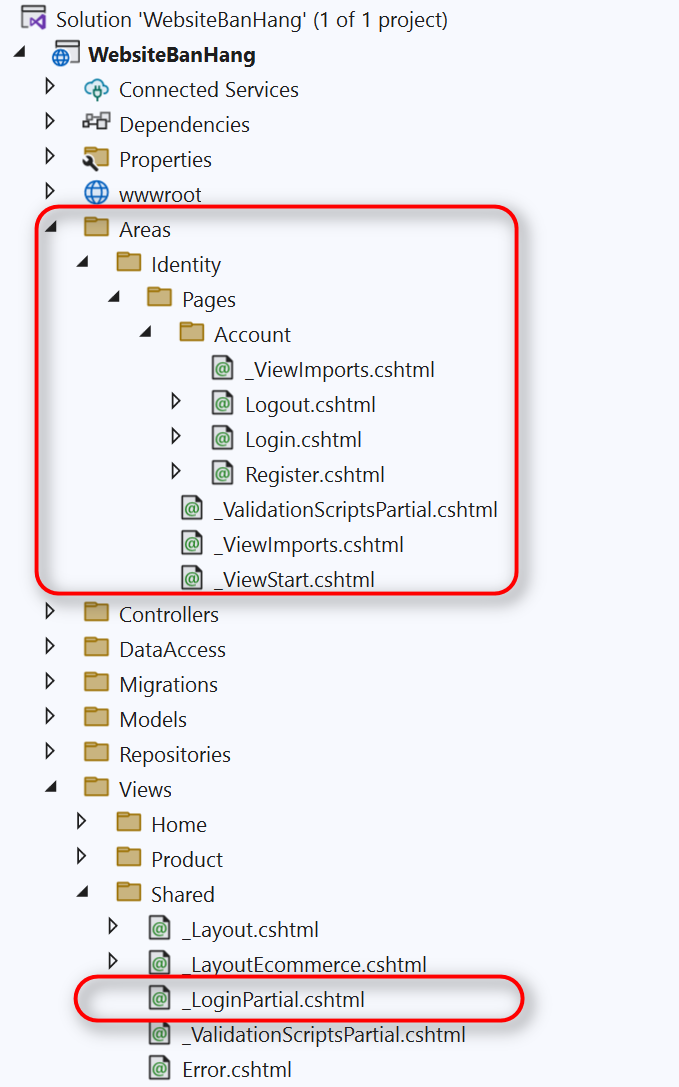
dotnet ef database update --project WebsiteBanHang

### Scaffolding ASP.NET Core Identity

Sử dụng ASP.NET Core scaffolder để thêm các view đăng nhập và đăng ký:

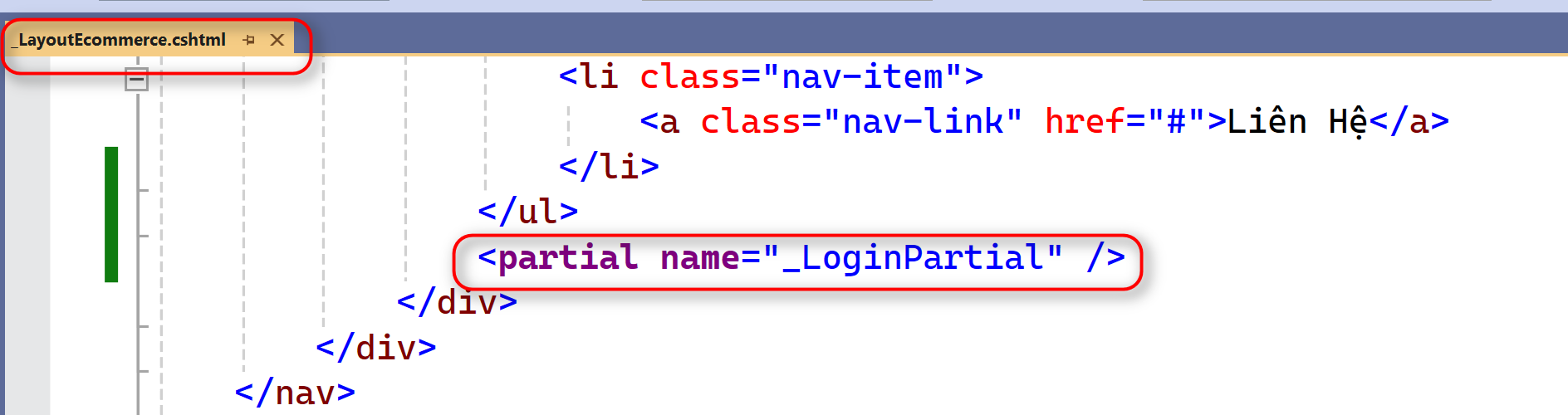
1. Trong Visual Studio, click chuột phải vào project, chọn **Add -> New Scaffolded Item.**
2. Chọn **Identity** và click **Add**.
3. Chọn các **pages** bạn muốn thêm, ví dụ: **Login** và **Register**.
4. Chọn **Add**.





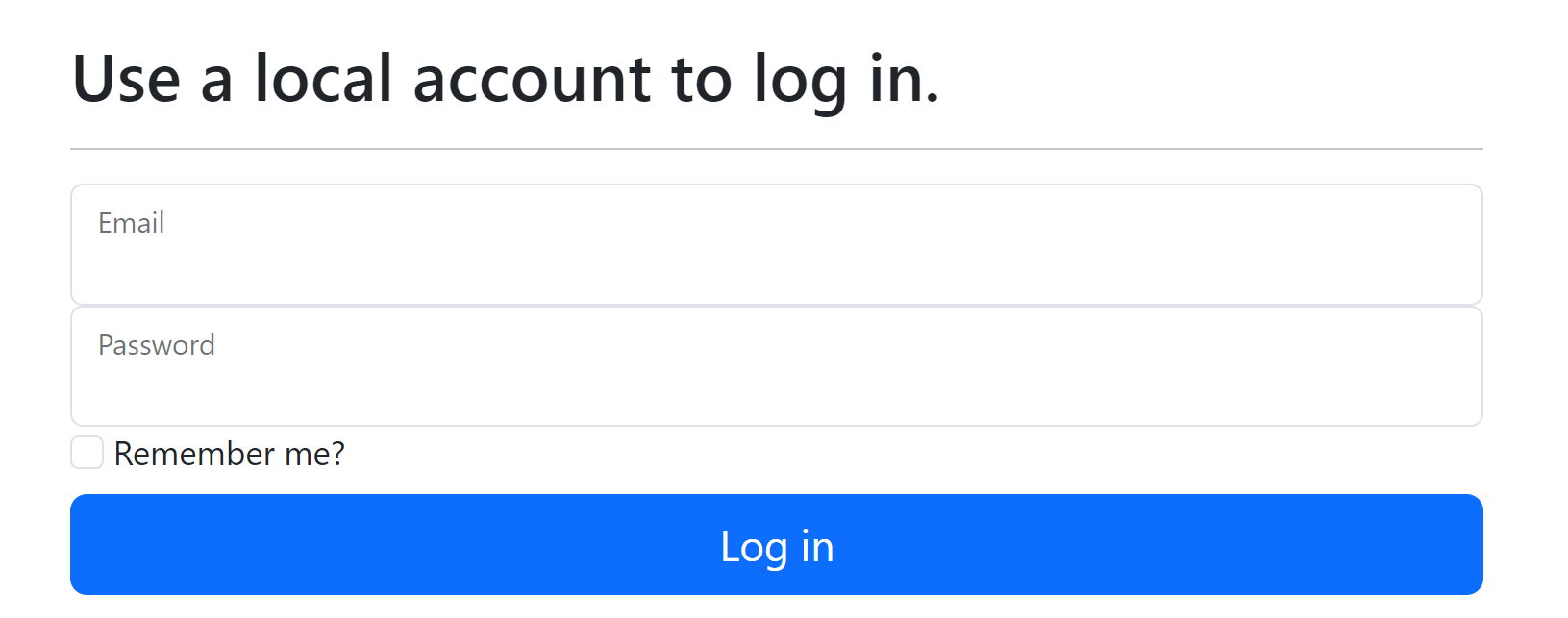
### Thêm Liên Kết Đăng Ký và Đăng Nhập

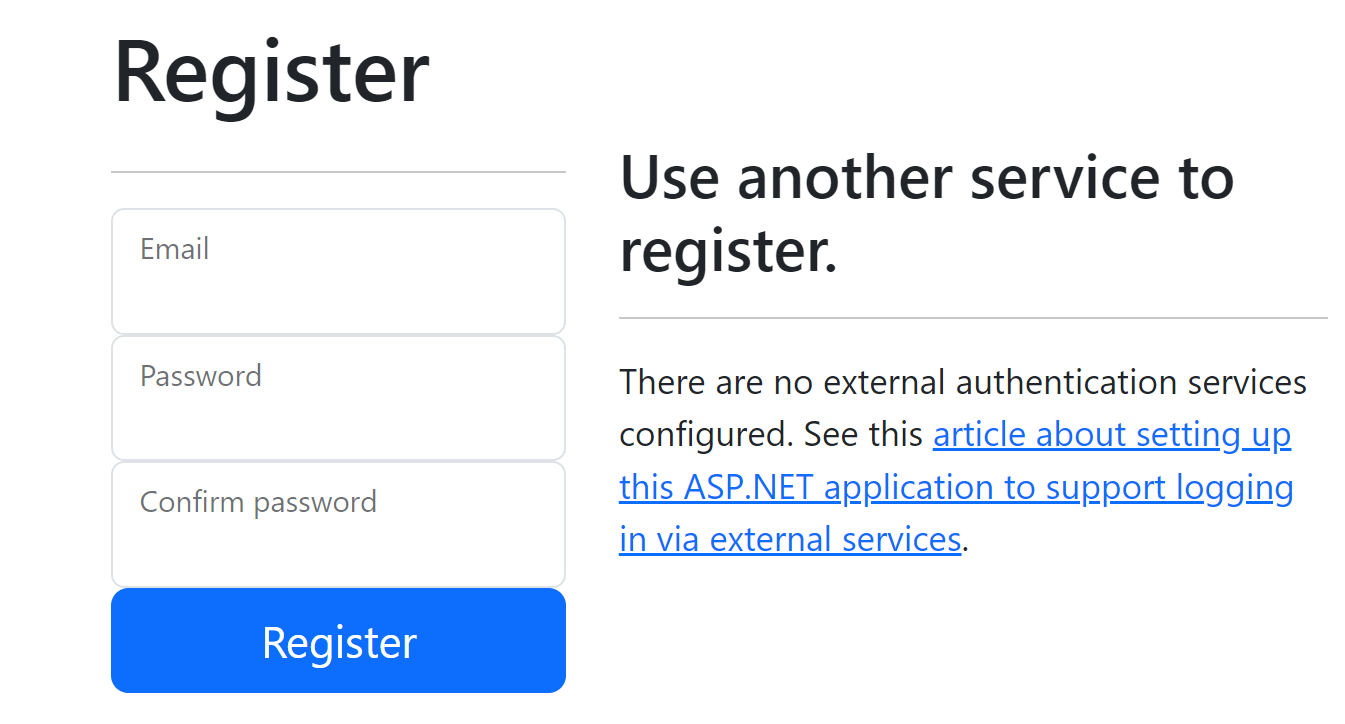
* Cập nhật `\_Layout.cshtml` để thêm các liên kết đăng ký và đăng nhập:



Kết quả







Thay đổi giao diện trang đăng ký và đăng nhập cho phù hợp với trang \_Layout mặc định.

### Phần vùng Area

Để tạo và thiết lập một Area Admin trong ứng dụng web bán hàng với ASP.NET Core MVC, bạn có thể làm theo các bước chi tiết sau:

1. Tạo Area Admin

- Trong Visual Studio, chuột phải vào project, chọn Add -> New Scaffolded Item

- Chọn Area, đặt tên là `Admin`, và tạo area.

2. Tạo Controllers và Views trong Admin Area:

- Trong `Areas/Admin`, tạo các thư mục `Controllers` và `Views`.

- Tạo `AdminController` trong `Areas/Admin/Controllers` với attribute `[Area("Admin")]`.

- Tạo các view tương ứng trong `Areas/Admin/Views/Admin`.

3. Cấu Hình Routing cho Admin Area:

- Trong `Program.cs`, thêm cấu hình routing cho admin area:

endpoints.MapControllerRoute(

name: "admin",

pattern: "{area:exists}/{controller=Home}/{action=Index}/{id?}"

);

4. Thêm Chức Năng Quản Lý Sản Phẩm:

- Tạo `ProductController` trong `Areas/Admin/Controllers`.

- Thêm các action `Index`, `Add`, `Edit`, `Delete` cho việc quản lý sản phẩm.

- Tạo các view tương ứng trong `Areas/Admin/Views/Product`.

5. Quản Lý Đơn Hàng:

- Tương tự, tạo `OrderController` trong `Areas/Admin` và các view tương ứng.

6. Phân Quyền Truy Cập:

- Sử dụng `[Authorize(Roles = "Admin")]` để hạn chế quyền truy cập chỉ cho admin.

7. Tạo Điều Hướng trong Admin Area:

- Trong `\_Layout.cshtml` của Admin Area, thêm liên kết tới các chức năng như quản lý sản phẩm, đơn hàng.

## Yêu cầu bổ sung

* Thêm các thông tin bổ sung cho trang đăng ký như: **Address, Age, FirstName, LastName,...vvv**
* Xây dựng lại giao diện cho các trang đăng nhập, đăng ký
* Xây dựng một trang layout mới với mẫu tùy chỉnh phù hợp với một trang web bán hàng, áp dụng trang layout mới này cho tất cả các trang con.
* Triển khai hoàn chỉnh các chức năng cho người quản trị như Thêm/Xóa/Sửa Sản phẩm, danh mục, đơn hàng, ...vvv

# XÂY DỰNG ỨNG DỤNG WEBSITE BÁN HÀNG VỚI ASP.NET CORE MVC (PHẦN 3)

Sau khi học xong bài này, sinh viên có thể nắm được:

* Xây dựng chức năng giỏ hàng và đặt hàng trong ứng dụng ASP.NET Core MVC

## Mục tiêu của bài thực hành

### Yêu cầu

Xây dựng chức năng giỏ hàng và đặt hàng trong ứng dụng ASP.NET Core MVC

## Hướng dẫn thực hiện

### Code mẫu thực hiện chức năng giỏ hàng

Để xây dựng một chức năng giỏ hàng sử dụng session trong ASP.NET Core, bạn cần thực hiện các bước sau:

* Cấu Hình Session trong `Program.cs`

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddDistributedMemoryCache();

builder.Services.AddSession(options =>

{

options.IdleTimeout = TimeSpan.FromMinutes(30);

options.Cookie.HttpOnly = true;

options.Cookie.IsEssential = true;

});

builder.Services.AddControllersWithViews();

var app = builder.Build();

// Đặt trước UseRouting

app.UseSession();

// Các middleware khác...

app.UseRouting();

app.UseEndpoints(endpoints =>

{

endpoints.MapControllerRoute(

name: "default",

pattern: "{controller=Home}/{action=Index}/{id?}");

});

app.Run();

* Tạo Helper Class để Làm Việc với Session

public static class SessionExtensions

{

public static void SetObjectAsJson(this ISession session, string key, object value)

{

session.SetString(key, JsonSerializer.Serialize(value));

}

public static T GetObjectFromJson<T>(this ISession session, string key)

{

var value = session.GetString(key);

return value == null ? default : JsonSerializer.Deserialize<T>(value);

}

}

* CartItem.cs

public class CartItem

{

public int ProductId { get; set; }

public string Name { get; set; }

public decimal Price { get; set; }

public int Quantity { get; set; }

}

* ShoppingCart.cs

public class ShoppingCart

{

public List<CartItem> Items { get; set; } = new List<CartItem>();

public void AddItem(CartItem item)

{

var existingItem = Items.FirstOrDefault(i => i.ProductId == item.ProductId);

if (existingItem != null)

{

existingItem.Quantity += item.Quantity;

}

else

{

Items.Add(item);

}

}

public void RemoveItem(int productId)

{

Items.RemoveAll(i => i.ProductId == productId);

}

// Các phương thức khác...

}

* Tạo `ShoppingCartController`

public class ShoppingCartController : Controller

{

public IActionResult AddToCart(int productId, int quantity)

{

// Giả sử bạn có phương thức lấy thông tin sản phẩm từ productId

var product = GetProductFromDatabase(productId);

var cartItem = new CartItem

{

ProductId = productId,

Name = product.Name,

Price = product.Price,

Quantity = quantity

};

var cart = HttpContext.Session.GetObjectFromJson<ShoppingCart>("Cart") ?? new ShoppingCart();

cart.AddItem(cartItem);

HttpContext.Session.SetObjectAsJson("Cart", cart);

return RedirectToAction("Index");

}

public IActionResult Index()

{

var cart = HttpContext.Session.GetObjectFromJson<ShoppingCart>("Cart") ?? new ShoppingCart();

return View(cart);

}

// Các actions khác...

private Product GetProductFromDatabase(int productId)

{

// Truy vấn cơ sở dữ liệu để lấy thông tin sản phẩm

}

}

* Index.cshtml (Trong Views/ShoppingCart)

@model ShoppingCart

<h2>Your Cart</h2>

<table>

<tr>

<th>Product</th>

<th>Quantity</th>

<th>Price</th>

<th>Total</th>

</tr>

@foreach (var item in Model.Items)

{

<tr>

<td>@item.Name</td>

<td>@item.Quantity</td>

<td>@item.Price</td>

<td>@(item.Price \* item.Quantity)</td>

<td>

<a asp-action="RemoveFromCart" asp-route-productId="@item.ProductId">Remove</a>

</td>

</tr>

}

</table>

***Lưu ý:***

* Đảm bảo rằng bạn đã xử lý việc lấy thông tin sản phẩm từ cơ sở dữ liệu trong phương thức `GetProductFromDatabase`.

### Hướng dẫn thực hiện chức năng đặt hàng

* Model `Order`

public class Order

{

public int Id { get; set; }

public string UserId { get; set; }

public DateTime OrderDate { get; set; }

public decimal TotalPrice { get; set; }

public string ShippingAddress { get; set; }

public string Notes { get; set; }

public IdentityUser User { get; set; }

public List<OrderDetail> OrderDetails { get; set; }

}

* OrderDetail (Chi Tiết Đơn Hàng): Lưu thông tin chi tiết cho mỗi mặt hàng trong đơn.

public class OrderDetail

{

public int Id { get; set; }

public int OrderId { get; set; }

public int ProductId { get; set; }

public int Quantity { get; set; }

public decimal Price { get; set; }

public Order Order { get; set; }

public Product Product { get; set; }

}

* Cập Nhật ApplicationDbContext

Thêm DbSet cho các lớp mới vào ApplicationDbContext.

public class ApplicationDbContext : IdentityDbContext<IdentityUser>

{

// ... Các DbSet hiện có ...

public DbSet<Order> Orders { get; set; }

public DbSet<OrderDetail> OrderDetails { get; set; }

}

* `ShoppingCartController` để Xử Lý Đặt Hàng

Trong `ShoppingCartController`, cập nhật action `Checkout` để xử lý thông tin địa chỉ giao hàng và ghi chú.

[Authorize]

public class ShoppingCartController : Controller

{

private readonly ApplicationDbContext \_context;

private readonly UserManager<IdentityUser> \_userManager;

public ShoppingCartController(ApplicationDbContext context, UserManager<IdentityUser> userManager)

{

\_context = context;

\_userManager = userManager;

}

public IActionResult Checkout()

{

return View(new Order());

}

[HttpPost]

public async Task<IActionResult> Checkout(Order order)

{

var cart = HttpContext.Session.GetObjectFromJson<ShoppingCart>("Cart");

if (cart == null || !cart.Items.Any())

{

// Xử lý giỏ hàng trống...

return RedirectToAction("Index");

}

var user = await \_userManager.GetUserAsync(User);

order.UserId = user.Id;

order.OrderDate = DateTime.UtcNow;

order.TotalPrice = cart.Items.Sum(i => i.Price \* i.Quantity);

order.OrderDetails = cart.Items.Select(i => new OrderDetail

{

ProductId = i.ProductId,

Quantity = i.Quantity,

Price = i.Price

}).ToList();

\_context.Orders.Add(order);

await \_context.SaveChangesAsync();

HttpContext.Session.Remove("Cart");

return View("OrderCompleted", order.Id); // Trang xác nhận hoàn thành đơn hàng

}

}

* Tạo View `Checkout` Checkout.cshtml

Tạo một view mới để nhập thông tin đặt hàng, bao gồm địa chỉ giao hàng và ghi chú.

@model Order

<h2>Checkout</h2>

<form asp-action="Checkout" method="post">

<div class="form-group">

<label asp-for="ShippingAddress">Shipping Address</label>

<input asp-for="ShippingAddress" class="form-control" />

</div>

<div class="form-group">

<label asp-for="Notes">Notes</label>

<textarea asp-for="Notes" class="form-control"></textarea>

</div>

<button type="submit" class="btn btn-primary">Place Order</button>

</form>

* Cập Nhật View `OrderCompleted` OrderCompleted.cshtml

Cập nhật view `OrderCompleted` để hiển thị thông tin xác nhận đơn hàng.

@model int

<h2>Order Completed</h2>

<p>Your order with ID @Model has been placed successfully.</p>

### Yêu cầu bổ sung

* Xây dựng giao diện thân thiện cho giỏ hàng.

# RESTful API

Sau khi học xong bài này, sinh viên có thể nắm được:

* Hiểu Về RESTful API: Nắm vững các khái niệm cơ bản của RESTful API, bao gồm các HTTP methods (GET, POST, PUT, DELETE) và cách tương tác với tài nguyên.
* Thiết Kế API Chuẩn RESTful: Biết cách thiết kế API theo chuẩn RESTful, bao gồm cách đặt endpoint, sử dụng HTTP methods, và quản lý tài nguyên.
* Sử Dụng ControllerBase: Hiểu cách sử dụng ControllerBase để xây dựng API Controllers và ánh xạ chúng với các endpoint.

## Mục tiêu của bài thực hành

### Giới thiệu

RESTful API (Representational State Transfer) là một kiến trúc thiết kế cho các dịch vụ web, dựa trên các nguyên tắc cơ bản như sự độc lập giữa client và server, tương tác thể hiện qua trạng thái biểu diễn, và sử dụng các phương thức HTTP để thực hiện các thao tác. Đây là một số nguyên tắc quan trọng của RESTful API:

* Stateless (Không Lưu Trạng Thái): Mỗi request từ client đều chứa đủ thông tin để server hiểu và xử lý. Server không lưu giữ trạng thái của client giữa các requests.
* Resource-Based (Dựa Trên Tài Nguyên): Mọi thứ trong hệ thống được xem như một tài nguyên (resource) và được xác định bởi URI (Uniform Resource Identifier).
* Representation (Biểu Diễn): Dữ liệu của tài nguyên được truyền tải giữa client và server dưới dạng biểu diễn, thường là JSON hoặc XML.
* CRUD Operations (Create, Read, Update, Delete): Sử dụng các phương thức HTTP tương ứng để thực hiện các thao tác CRUD trên tài nguyên.

### Yêu cầu

Thiết Kế RESTful API cho CRUD Operations trên Product

* GET /api/products: Lấy danh sách tất cả sản phẩm.
* GET /api/products/{id}: Lấy thông tin chi tiết của một sản phẩm theo ID.
* POST /api/products: Tạo một sản phẩm mới.
* PUT /api/products/{id}: Cập nhật thông tin của sản phẩm theo ID.
* DELETE /api/products/{id}: Xóa một sản phẩm theo ID.

## Hướng dẫn thực hiện

Dưới đây là một ví dụ đơn giản về cách xây dựng một **RESTful API** cho **Product** trong **ASP.NET Core**, sử dụng **Controller và Repository Pattern.**

* Product.cs (Model)

public class Product

{

public int Id { get; set; }

public string Name { get; set; }

public decimal Price { get; set; }

// Other properties

}

* IProductRepository.cs

public interface IProductRepository

{

Task<IEnumerable<Product>> GetProductsAsync();

Task<Product> GetProductByIdAsync(int id);

Task AddProductAsync(Product product);

Task UpdateProductAsync(Product product);

Task DeleteProductAsync(int id);

}

* ProductRepository.cs

public class ProductRepository : IProductRepository

{

private readonly YourDbContext \_context;

public ProductRepository(YourDbContext context)

{

\_context = context;

}

public async Task<IEnumerable<Product>> GetProductsAsync()

{

return await \_context.Products.ToListAsync();

}

public async Task<Product> GetProductByIdAsync(int id)

{

return await \_context.Products.FindAsync(id);

}

public async Task AddProductAsync(Product product)

{

\_context.Products.Add(product);

await \_context.SaveChangesAsync();

}

public async Task UpdateProductAsync(Product product)

{

\_context.Entry(product).State = EntityState.Modified;

await \_context.SaveChangesAsync();

}

public async Task DeleteProductAsync(int id)

{

var product = await \_context.Products.FindAsync(id);

if (product != null)

{

\_context.Products.Remove(product);

await \_context.SaveChangesAsync();

}

}

}

* ProductApiController.cs

[ApiController]

[Route("api/products")]

public class ProductApiController : ControllerBase

{

private readonly IProductRepository \_productRepository;

public ProductApiController(IProductRepository productRepository)

{

\_productRepository = productRepository;

}

[HttpGet]

public async Task<IActionResult> GetProducts()

{

try

{

var products = await \_productRepository.GetProductsAsync();

return Ok(products);

}

catch (Exception ex)

{

// Handle exception

return StatusCode(500, "Internal server error");

}

}

[HttpGet("{id}")]

public async Task<IActionResult> GetProductById(int id)

{

try

{

var product = await \_productRepository.GetProductByIdAsync(id);

if (product == null)

return NotFound();

return Ok(product);

}

catch (Exception ex)

{

// Handle exception

return StatusCode(500, "Internal server error");

}

}

[HttpPost]

public async Task<IActionResult> AddProduct([FromBody] Product product)

{

try

{

await \_productRepository.AddProductAsync(product);

return CreatedAtAction(nameof(GetProductById), new { id = product.Id }, product);

}

catch (Exception ex)

{

// Handle exception

return StatusCode(500, "Internal server error");

}

}

[HttpPut("{id}")]

public async Task<IActionResult> UpdateProduct(int id, [FromBody] Product product)

{

try

{

if (id != product.Id)

return BadRequest();

await \_productRepository.UpdateProductAsync(product);

return NoContent();

}

catch (Exception ex)

{

// Handle exception

return StatusCode(500, "Internal server error");

}

}

[HttpDelete("{id}")]

public async Task<IActionResult> DeleteProduct(int id)

{

try

{

await \_productRepository.DeleteProductAsync(id);

return NoContent();

}

catch (Exception ex)

{

// Handle exception

return StatusCode(500, "Internal server error");

}

}

}

Để xây dựng front end để gọi các API của `ProductApiController`, bạn có thể sử dụng JavaScript hoặc một framework JavaScript như Angular, React, hoặc Vue.js. Dưới đây là một ví dụ sử dụng JavaScript để thực hiện các yêu cầu API:

* Lấy Danh Sách Sản Phẩm (GET All Products):

// Sử dụng Fetch API hoặc Axios

fetch('https://your-api-url/api/products')

.then(response => response.json())

.then(products => {

// Xử lý danh sách sản phẩm

console.log(products);

})

.catch(error => console.error('Error:', error));

* Lấy Thông Tin Chi Tiết Sản Phẩm (GET Product by ID):

// Thay {id} bằng ID cụ thể của sản phẩm

const productId = 1;

fetch(`https://your-api-url/api/products/${productId}`)

.then(response => response.json())

.then(product => {

// Xử lý thông tin chi tiết sản phẩm

console.log(product);

})

.catch(error => console.error('Error:', error));

* Tạo Mới Sản Phẩm (POST Create Product):

// Thông tin sản phẩm mới cần tạo

const newProduct = {

name: 'New Product',

price: 100,

description: 'A new product',

// Thêm các thông tin khác

};

fetch('https://your-api-url/api/products', {

method: 'POST',

headers: {

'Content-Type': 'application/json',

},

body: JSON.stringify(newProduct),

})

.then(response => response.json())

.then(createdProduct => {

// Xử lý thông tin sản phẩm đã tạo

console.log(createdProduct);

})

.catch(error => console.error('Error:', error));

```

* Cập Nhật Thông Tin Sản Phẩm (PUT Update Product):

// Thay {id} và cập nhật thông tin sản phẩm

const productIdToUpdate = 1;

const updatedProduct = {

id: productIdToUpdate,

name: 'Updated Product',

price: 150,

description: 'An updated product',

// Thêm các thông tin khác

};

fetch(`https://your-api-url/api/products/${productIdToUpdate}`, {

method: 'PUT',

headers: {

'Content-Type': 'application/json',

},

body: JSON.stringify(updatedProduct),

})

.then(response => {

if (response.status === 204) {

console.log('Product updated successfully.');

} else {

console.error('Failed to update product.');

}

})

.catch(error => console.error('Error:', error));

```

* Xóa Sản Phẩm (DELETE Product):

// Thay {id} bằng ID cụ thể của sản phẩm cần xóa

const productIdToDelete = 1;

fetch(`https://your-api-url/api/products/${productIdToDelete}`, {

method: 'DELETE',

})

.then(response => {

if (response.status === 204) {

console.log('Product deleted successfully.');

} else {

console.error('Failed to delete product.');

}

})

.catch(error => console.error('Error:', error));

Lưu ý rằng bạn cần thay thế `'https://your-api-url'` bằng URL thực tế của API của bạn. Ngoài ra, có thể bạn muốn sử dụng một thư viện như Axios để thực hiện các yêu cầu API một cách thuận tiện hơn.

## Yêu cầu bổ sung

Hoàn thiện giao diện cho tất cả các trang Thêm/Xóa/Sửa/Đọc với API của Product ở phần trên.

# 

# TÀI LIỆU THAM KHẢO

1. Bài giảng Lập trình C# trên Windows, ThS. Nguyễn Hà Giang, 2010.
2. C# and .NET programing, msdn.microsoft.com, 2012.
3. Pro C# 2005 and the .NET 2.0 Platform, Andrew Troelsen, Apress, 2005.
4. C# 2.0 Practical Guide for Programmers, Michel de Champlain, Brian G. Patrick, Morgan Kaufmann publishers. 2005.
5. Windows Forms Programming with C#, Erik Brown, Manning Publications, 2008.
6. Microsoft Visual C# 2010 Step by Step, Microsoft Press, 2010.
7. Windows Forms 2.0 Programming, Chris Sells, Michael Weinhardt, Additon Wesley Professional, 2003.
8. Teach yourself .NET Windows Forms in 21 Days, Chris Payne, SAMS, 2003.
9. Source code tham khảo ở http://www.wrox.com.
10. Các topic lập trình ở www.codeguru.com, [www.codeproject.com](http://www.codeproject.com).
11. https://dotnettutorials.net/