KHOA KỸ THUẬT VÀ CÔNG NGHỆ

**BỘ MÔN CÔNG NGHỆ THÔNG TIN**

****

**BÁO CÁO ĐỒ ÁN CHUYÊN NGÀNH**

**HỌC KỲ I, NĂM HỌC 2024-2025**

**ÁP DỤNG FRAMEWORK ASP.NET CORE**

**XÂY DỰNG WEBSITE BÁN MÁY TÍNH**

Giảng viên hướng dẫn:

Ngô Thanh Huy

Sinh viên thực hiện:

Trần Bá Hiếu – 110121024

***Trà Vinh, .. tháng .. năm 2024***

KHOA KỸ THUẬT VÀ CÔNG NGHỆ

**BỘ MÔN CÔNG NGHỆ THÔNG TIN**

****

**BÁO CÁO ĐỒ ÁN CHUYÊN NGÀNH**

**HỌC KỲ I, NĂM HỌC 2024-2025**

**ÁP DỤNG FRAMEWORK ASP.NET CORE**

**XÂY DỰNG WEBSITE BÁN MÁY TÍNH**

Giảng viên hướng dẫn:

Ngô Thanh Huy

Sinh viên thực hiện:

Trần Bá Hiếu – 110121024

***Trà Vinh, .. tháng .. năm 2024***

**NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN**

*Trà Vinh, ngày …. . tháng …… năm ……*

**Giáo viên hướng dẫn**

*(Ký tên và ghi rõ họ tên)*

**NHẬN XÉT CỦA THÀNH VIÊN HỘI ĐỒNG**

*Trà Vinh, ngày …. . tháng …… năm ……*

**Thành viên hội đồng**

*(Ký tên và ghi rõ họ tên)*

**LỜI CẢM ƠN**

Lời nói đầu, nhóm chúng em xin cảm ơn thầy Ngô Thanh Huy đã hỗ trợ, hướng dẫn tận tình chúng em trong thời gian xây dựng đồ án chuyên ngành, những kiến thức mà thầy đã dạy và hướng dẫn em sẽ là hành trang quý báo trên con đường học vấn và phát triển sự nghiệp tương lai rộng mở của em. Thầy đã luôn kiên nhẫn, nhiệt tình trong việc truyền đạt kiến thức và kinh nghiệm quý báo, giúp chúng em vượt qua những khó khăn và thử thách trong quá trình học tập và nghiên cứu.

Những lời khuyên, góp ý của thầy không chỉ là kim chỉ nam cho sự phát triển của đồ án chuyên ngành này mà còn là nguồn động viên, khích lệ tinh thần lớn lao cho em.

Em xin hứa sẽ tiếp tục nỗ lực không ngừng để không phụ lòng thầy đã dành cho em.

Xin chân thành cảm ơn thầy.

Trà Vinh, .. Tháng .. năm 2024

**Trần Bá Hiếu**

**MỤC LỤC**

**Danh mục** **Trang**

[KHOA KỸ THUẬT VÀ CÔNG NGHỆ 1](#_Toc187237925)

[BỘ MÔN CÔNG NGHỆ THÔNG TIN 1](#_Toc187237926)

[BÁO CÁO ĐỒ ÁN CHUYÊN NGÀNH 1](#_Toc187237927)

[KHOA KỸ THUẬT VÀ CÔNG NGHỆ 2](#_Toc187237928)

[BỘ MÔN CÔNG NGHỆ THÔNG TIN 2](#_Toc187237929)

[BÁO CÁO ĐỒ ÁN CHUYÊN NGÀNH 2](#_Toc187237930)

[TÓM TẮT NIÊN LUẬN x](#_Toc187237931)

[CHƯƠNG 1: TỔNG QUAN 1](#_Toc187237932)

[1.1. Tổng quan về vấn đề 1](#_Toc187237933)

[1.2. Hướng giải quyết 2](#_Toc187237934)

[CHƯƠNG 2: NGHIÊN CỨU LÝ THUYẾT 4](#_Toc187237935)

[2.1. Ngôn ngữ lập trình C# ( C Sharp ) 4](#_Toc187237936)

[2.2. ASP.NET Framework và sự khác nhau giữa ASP.NET Core và ASP.NET 5](#_Toc187237937)

[2.3. Mô hình MVC (Model-View-Controller) 7](#_Toc187237938)

[2.3.1. Khái quát về cấu trúc mô hình MVC 8](#_Toc187237939)

[2.3.2. Cách hoạt động của mô hình MVC 8](#_Toc187237940)

[2.3.3. Cấu trúc mô hình MVC 9](#_Toc187237941)

[2.3.4. So sánh mô hình MVC và các mô hình khác 10](#_Toc187237942)

[2.4. Một số phương thức, khái niệm, công nghệ/công cụ... khác được sử dụng 11](#_Toc187237943)

[CHƯƠNG 3: THỰC NGHIỆM 12](#_Toc187237944)

[3.1. Thiết kế mô hình ERD (Entity Relationship Diagram) 12](#_Toc187237945)

[3.2. Thiết kế CSDL 13](#_Toc187237946)

[3.2.1. Thực thể sản phẩm (Products) 13](#_Toc187237947)

[3.2.2. Thực thể người dùng (Users) 15](#_Toc187237948)

[3.2.3. Thực thể danh mục (Categories) 15](#_Toc187237949)

[3.2.4. Thực thể thương hiệu (Brands) 16](#_Toc187237950)

[3.2.5. Thực thể đặt hàng (Orders) 17](#_Toc187237951)

[3.2.6. Thực thể chi tiết đơn hàng (OrderDetails) 18](#_Toc187237952)

[3.2.7. Thực thể đánh giá (Ratings) 19](#_Toc187237953)

[3.2.8. Thực thể địa chỉ đơn hàng (BillingAddresses) 20](#_Toc187237954)

[3.3. Cấu trúc Models của Website 21](#_Toc187237955)

[3.4. Cấu trúc Controllers của Website 25](#_Toc187237956)

[3.4.1. Cấu trúc Controllers của trang chính 25](#_Toc187237957)

[3.4.2. Cấu trúc Controllers của trang Admin 28](#_Toc187237958)

[3.5. Cấu trúc Views của Website 33](#_Toc187237959)

[3.5.1. Cấu trúc Views của trang chính 33](#_Toc187237960)

[3.5.2. Cấu trúc Views của trang Admin 34](#_Toc187237961)

[CHƯƠNG 4: KẾT QUẢ NGHIÊN CỨU 35](#_Toc187237962)

[4.1. Phần trang chính 35](#_Toc187237963)

[4.1.1. Giao diện phần Header 35](#_Toc187237964)

[4.1.2. Giao diện Banner 36](#_Toc187237965)

[4.1.3. Giao diện Footer 37](#_Toc187237966)

[4.1.4. Tổng thể giao diện trang chủ 37](#_Toc187237967)

[4.1.5. Giao diện trang đăng nhập và chức năng đăng nhập 42](#_Toc187237968)

[4.1.6. Giao diện trang đăng ký và chức năng đăng ký 44](#_Toc187237969)

[4.1.7. Giao diện trang sản phẩm 46](#_Toc187237970)

[4.1.8. Giao diện trang chi tiết sản phẩm 47](#_Toc187237971)

[4.1.9. Giao diện Form giỏ hàng 50](#_Toc187237972)

[4.1.10. Giao diện và chức năng trang hoàn tất đặt hàng 51](#_Toc187237973)

[4.1.11. Giao diện và chức năng tìm kiếm sản phẩm 53](#_Toc187237974)

[4.2. Phần Admin 55](#_Toc187237975)

[4.2.1. Tổng thể trang Admin 55](#_Toc187237976)

[4.2.2. Giao diện trang quản lý sản phẩm 56](#_Toc187237977)

[4.2.3. Giao diện trang quản lý danh mục 58](#_Toc187237978)

[4.2.4. Giao diện trang quản lý thương hiệu 60](#_Toc187237979)

[4.2.5. Giao diện trang Role 61](#_Toc187237980)

[4.2.6. Giao diện trang quản lý người dùng 62](#_Toc187237981)

[4.2.7. Giao diện trang quản lý đơn hàng 63](#_Toc187237982)

[CHƯƠNG 5: KẾT LUẬN 65](#_Toc187237983)

[5.1. Ưu điểm 65](#_Toc187237984)

[5.2. Nhược điểm 65](#_Toc187237985)

[5.3. Hướng phát triển 65](#_Toc187237986)

[TÀI LIỆU THAM KHẢO 67](#_Toc187237987)

**DANH MỤC HÌNH ẢNH**

[Hình 2.1. Mô hình MVC 8](#_Toc187237994)

[Hình 3.1. Mô hình ERD 12](#_Toc187237995)

[Hình 3.2. Cấu trúc Model 22](#_Toc187237996)

[Hình 3.3. Cấu trúc Controller trang chính 25](#_Toc187237997)

[Hình 3.4. Cấu trúc Controllers trang Admin 28](#_Toc187237998)

[Hình 3.5. Cấu trúc Views Trang chính 33](#_Toc187237999)

[Hình 3.6. Cấu trúc Views Trang Admin 34](#_Toc187238000)

[Hình 4.1. Giao diện Header 35](#_Toc187238001)

[Hình 4.2. Giao diện Header sau khi đăng nhập 36](#_Toc187238002)

[Hình 4.3. Giao diện banner 36](#_Toc187238003)

[Hình 4.4. Giao diện footer 37](#_Toc187238004)

[Hình 4.5 Giao diện trang chủ 40](#_Toc187238005)

[Hình 4.6. Phần sản phẩm của chúng tôi 41](#_Toc187238006)

[Hình 4.7. Giao diện trang đăng nhập 42](#_Toc187238007)

[Hình 4.8. Thông báo lỗi khi tên đăng nhập hoặc mật khẩu 43](#_Toc187238008)

[Hình 4.9. Giao diện trang đăng nhập 44](#_Toc187238009)

[Hình 4.10. Giao diện trang sản phẩm 46](#_Toc187238010)

[Hình 4.11. Giao diện chi tiết sản phẩm 47](#_Toc187238011)

[Hình 4.12. Form mô tả sản phẩm 47](#_Toc187238012)

[Hình 4.13. Form đánh giá 48](#_Toc187238013)

[Hình 4.14. Mục các sản phẩm liên quan 49](#_Toc187238014)

[Hình 4.15. Thông báo thêm vào giỏ hàng thành công 49](#_Toc187238015)

[Hình 4.16. Giao diện Form giỏ hàng 50](#_Toc187238016)

[Hình 4.17. Giao diện form hoàn tất đặt hàng 51](#_Toc187238017)

[Hình 4.18. Email xác nhận đặt hàng hoàn tất 52](#_Toc187238018)

[Hình 4.19. Giao diện chức năng tìm kiếm sản phẩm 53](#_Toc187238019)

[Hình 4.20. Giao diện kết quả sau khi tìm kiếm 54](#_Toc187238020)

[Hình 4.21. Giao diện tổng thể trang Admin 55](#_Toc187238021)

[Hình 4.22. Giao diện trang quản lý sản phẩm 56](#_Toc187238022)

[Hình 4.23. Giao diện thêm sản phẩm 56](#_Toc187238023)

[Hình 4.24. Giao diện trang sửa sản phẩm 57](#_Toc187238024)

[Hình 4.25. Giao diện trang quản lý danh mục 58](#_Toc187238025)

[Hình 4.26. Giao diện trang thêm danh mục 59](#_Toc187238026)

[Hình 4.27. Giao diện trang sửa danh mục 59](#_Toc187238027)

[Hình 4.28. Giao diện trang quản lý thương hiệu 60](#_Toc187238028)

[Hình 4.29. Giao diện trang thêm thương hiệu 60](#_Toc187238029)

[Hình 4.30. Giao diện trang sửa thương hiệu 61](#_Toc187238030)

[Hình 4.31. Giao diện trang Role 61](#_Toc187238031)

[Hình 4.32. Giao diện trang quản lý người dùng 62](#_Toc187238032)

[Hình 4.33. Form chỉnh sửa người dùng 62](#_Toc187238033)

[Hình 4.34. Giao diện trang quản lý đơn hàng 63](#_Toc187238034)

[Hình 4.35. Form xem chi tiết đơn hàng 63](#_Toc187238035)

[Hình 4.36. File PDF khi xuất hóa đơn 64](#_Toc187238036)

**DANH MỤC BẢNG BIỂU**

[Bảng 3.1. Sản phẩm 13](#_Toc187238037)

[Bảng 3.2. Người dùng 15](#_Toc187238038)

[Bảng 3.3. Danh mục 15](#_Toc187238039)

[Bảng 3.4. Thương hiệu 16](#_Toc187238040)

[Bảng 3.5. Đơn hàng 17](#_Toc187238041)

[Bảng 3.6. Chi tiết đơn hàng 18](#_Toc187238042)

[Bảng 3.7. Đánh giá 19](#_Toc187238043)

[Bảng 3.8. Chi tiết đơn hàng 20](#_Toc187238044)

# TÓM TẮT NIÊN LUẬN

**Vấn đề nghiên cứu:**

**-** Áp dụng framework Asp.net Core xây dựng website bán máy tính.

**Các hướng tiếp cận:**

+ Tìm hiểu về ngôn ngữ C# và Asp.net core framework.

+ Tìm hiểu mô hình MVC.

+ Tiến hành xây dựng View, Model và Controller.

+ Kiểm tra hiệu năng của Website.

**Một số kết quả đạt được:**

+ Website hoạt động ổn định đảm bảo các chức năng cần thiết.

+ Kiểm thử và đánh giá website khi hoạt động.

**Kết cấu của bài báo cáo:**

* **Chương 1:** Tổng quan.
* **Chương 2:** Nghiên cứu lý thuyết.
* **Chương 3:** Thực nghiệm.
* **Chương 4:** Kết quả nghiên cứu.
* **Chương 5:** Kết luận.

# TỔNG QUAN

## Tổng quan về vấn đề

Trong thời đại công nghệ hiện nay, việc phát triển các website thương mại điện tử trở nên rất quan trọng, đặc biệt là đối với các ngành hàng như bán máy tính, nơi người tiêu dùng ngày càng có nhu cầu mua sắm trực tuyến. Đồ án này tập trung vào việc xây dựng một website bán máy tính sử dụng framework ASP.NET Core, một công nghệ mạnh mẽ và phổ biến để phát triển các ứng dụng web hiện đại.

Mục tiêu của đề tài là thiết kế và phát triển một website bán máy tính, giúp khách hàng dễ dàng tìm kiếm và mua sản phẩm qua các tính năng như lọc sản phẩm, giỏ hàng, thanh toán trực tuyến, và quản lý tài khoản người dùng. Website sẽ được xây dựng với giao diện thân thiện và dễ sử dụng, đồng thời đảm bảo tính bảo mật và hiệu suất cao.

Công nghệ ASP.NET Core sẽ được áp dụng để phát triển website, vì đây là một framework mã nguồn mở, dễ dàng triển khai và có khả năng mở rộng linh hoạt. ASP.NET Core cung cấp các công cụ mạnh mẽ như Entity Framework để tương tác với cơ sở dữ liệu, Razor Pages để tạo giao diện động và các API RESTful để hỗ trợ kết nối giữa frontend và backend.

Trong quá trình phát triển, một số vấn đề quan trọng cần được giải quyết, bao gồm:

* **Quản lý sản phẩm**: Cần xây dựng hệ thống quản lý danh mục sản phẩm, bao gồm thông tin chi tiết về máy tính, giá cả, và các lựa chọn cấu hình.
* **Quản lý đơn hàng**: Hệ thống phải đảm bảo quá trình đặt hàng và thanh toán diễn ra suôn sẻ và an toàn.
* **Tối ưu hóa trải nghiệm người dùng**: Website cần được tối ưu hóa để nhanh chóng và dễ sử dụng trên cả desktop và mobile.
* **Bảo mật và xác thực**: Cần có hệ thống bảo mật để bảo vệ thông tin người dùng và giao dịch.

Từ đó, đề tài này không chỉ giúp nâng cao kỹ năng lập trình web với ASP.NET Core mà còn mang lại một giải pháp thực tiễn cho việc phát triển các website thương mại điện tử trong lĩnh vực bán máy tính.

## Hướng giải quyết

Để giải quyết các vấn đề đã đề cập trong tổng quan, mô hình MVC (Model-View-Controller) của ASP.NET Core sẽ được áp dụng một cách linh hoạt. Mô hình MVC giúp phân chia rõ ràng các thành phần của ứng dụng, làm cho việc phát triển và bảo trì dễ dàng hơn, đồng thời tối ưu hóa trải nghiệm người dùng và khả năng mở rộng của hệ thống.

*Một vài lớp ví dụ để có thể có cái nhìn tổng quan hơn về hướng giải quyết cho việc xây dựng Website:*

**Mô hình (Model)**: Mô hình đại diện cho dữ liệu của ứng dụng và các quy tắc nghiệp vụ. Trong dự án website bán máy tính, chúng ta sẽ xây dựng các lớp Model để quản lý thông tin về sản phẩm (máy tính), khách hàng, đơn hàng và các chức năng khác.

* Sản phẩm: Một lớp Product sẽ chứa các thuộc tính như tên, mô tả, giá, ảnh, và các đặc tính kỹ thuật khác của máy tính.
* Đơn hàng: Lớp Order sẽ lưu trữ thông tin về các đơn hàng, bao gồm các sản phẩm trong giỏ hàng, trạng thái thanh toán và thông tin vận chuyển.
* Khách hàng: Lớp User sẽ lưu trữ thông tin người dùng, bao gồm tên, email, mật khẩu, và địa chỉ giao hàng.

**Giao diện người dùng (View)**: Giao diện người dùng sẽ được xây dựng bằng Razor Views trong ASP.NET Core, giúp tách biệt logic hiển thị khỏi logic xử lý. Các trang chính của website sẽ bao gồm:

* Trang chủ: Hiển thị các sản phẩm nổi bật và các danh mục sản phẩm.
* Trang chi tiết sản phẩm: Cung cấp thông tin chi tiết về một sản phẩm, bao gồm các đặc tính kỹ thuật, hình ảnh, và nút "Thêm vào giỏ hàng".
* Giỏ hàng và thanh toán: Người dùng có thể xem và chỉnh sửa các sản phẩm trong giỏ hàng, sau đó thanh toán trực tuyến.
* Trang quản lý đơn hàng: Người quản trị có thể xem và quản lý các đơn hàng đã được đặt.

**Bộ điều khiển (Controller)**: Các Controller sẽ xử lý logic và tương tác giữa Model và View. Ví dụ:

* ProductController: Xử lý các yêu cầu về danh sách sản phẩm, chi tiết sản phẩm, và tìm kiếm sản phẩm.
* CartController: Xử lý các chức năng liên quan đến giỏ hàng, như thêm sản phẩm, xóa sản phẩm, tính tổng tiền.
* OrderController: Xử lý quá trình thanh toán và quản lý đơn hàng.
* AccountController: Quản lý đăng nhập, đăng ký và thông tin tài khoản người dùng.

**Tính năng quản lý và bảo mật**: Để đảm bảo bảo mật cho website, các tính năng như xác thực người dùng, phân quyền và bảo vệ thông tin khách hàng sẽ được triển khai. ASP.NET Core cung cấp các tính năng như ASP.NET Identity để xử lý việc đăng nhập, đăng ký, và phân quyền người dùng.

**Xác thực và phân quyền**: Sử dụng ASP.NET Core Identity để xử lý việc đăng ký và đăng nhập người dùng. Phân quyền sẽ đảm bảo rằng chỉ người quản trị mới có thể truy cập vào các tính năng quản lý sản phẩm và đơn hàng.

**Bảo mật thanh toán**: Để bảo vệ các giao dịch thanh toán, việc tích hợp với các cổng thanh toán như PayPal hoặc Stripe sẽ được thực hiện để đảm bảo tính bảo mật cao.

**Tối ưu hóa hiệu suất**: Để đảm bảo website hoạt động mượt mà, đặc biệt khi có lượng người dùng lớn, các kỹ thuật tối ưu hóa như caching, lazy loading, và phân trang sẽ được áp dụng.

# NGHIÊN CỨU LÝ THUYẾT

## Ngôn ngữ lập trình C# ( C Sharp )

C Sharp hay được biết đến C# là một ngôn ngữ lập trình do Microsoft phát triển, thuộc dòng ngôn ngữ lập trình C. Nó được sử dụng chủ yếu trong môi trường .NET, giúp xây dựng các ứng dụng trên Windows, web, và di động. C# dễ học và có cú pháp rõ ràng, giống như các ngôn ngữ lập trình khác như Java và C++ nhưng lại đơn giản hơn, dễ hiểu hơn.

C# hỗ trợ các đặc điểm như lập trình hướng đối tượng, giúp người lập trình dễ dàng quản lý và tái sử dụng mã nguồn. Nó cũng có tính năng quản lý bộ nhớ tự động thông qua garbage collection, giảm bớt những lo ngại về việc giải phóng bộ nhớ thủ công.

Với tính ổn định và khả năng tích hợp tốt với các công nghệ khác của Microsoft, C# được sử dụng rộng rãi trong việc phát triển các ứng dụng doanh nghiệp, phần mềm desktop, ứng dụng web và game.

Tóm lại, C# là một ngôn ngữ mạnh mẽ nhưng dễ tiếp cận, phù hợp với nhiều loại ứng dụng và là công cụ lý tưởng cho các dự án phần mềm hiện đại. **[1]**

**Ví dụ khai báo Model cho thực thể User:**

public class UserModel

{

public int Id { get; set; }

[Required(ErrorMessage = "Yêu cầu nhập Tên đăng nhập")]

public string UserName { get; set; }

[DataType(DataType.Password), Required(ErrorMessage = "Yêu cầu nhập Mật khẩu")]

public string Password { get; set; }

[Required(ErrorMessage = "Yêu cầu nhập Email"), EmailAddress]

public string Email { get; set; }

}

## ASP.NET Framework và sự khác nhau giữa ASP.NET Core và ASP.NET

ASP.NET Core là một framework phát triển ứng dụng web mã nguồn mở và đa nền tảng (cross-platform) được Microsoft phát triển. Nó là phiên bản cải tiến của ASP.NET và được thiết kế lại để chạy trên nhiều hệ điều hành khác nhau như Windows, Linux và macOS, thay vì chỉ giới hạn trên Windows như trước đây. [2]

Dưới đây là sự khác biệt chính giữa ASP.NET Core và ASP.NET để có cái nhìn tổng quát hơn về Framework này:

1. **Đa nền tảng (Cross-Platform):**

+ ASP.NET: Chỉ chạy trên hệ điều hành Windows.

+ ASP.NET Core: Được thiết kế để chạy trên nhiều nền tảng khác nhau, bao gồm Windows, Linux và macOS. Điều này giúp các nhà phát triển có thể xây dựng và triển khai ứng dụng web của mình trên bất kỳ nền tảng nào.

1. **Hiệu suất:**

+ ASP.NET: Mặc dù có hiệu suất tốt nhưng không được tối ưu như ASP.NET Core.

+ ASP.NET Core: Cải thiện rất nhiều về hiệu suất so với ASP.NET. Nó sử dụng một mô hình xử lý nhẹ và hiện đại hơn, giúp các ứng dụng chạy nhanh hơn và sử dụng tài nguyên hệ thống hiệu quả hơn.

1. **Kiến trúc:**

+ ASP.NET: Được xây dựng dựa trên mô hình Web Forms, MVC, và Web API, có một số phần không linh hoạt và dễ bị lỗi khi phát triển ứng dụng lớn.

+ ASP.NET Core: Hỗ trợ một kiến trúc sạch hơn và linh hoạt hơn, chủ yếu tập trung vào MVC và API, loại bỏ những phần thừa thãi như Web Forms, và giúp cấu trúc ứng dụng dễ duy trì và mở rộng hơn.

1. **Tích hợp Dependency Injection:**

*Dependency Injection là một kỹ thuật mà trong đó một đối tượng (class) không tự tạo ra các phụ thuộc của mình (dependencies), mà thay vào đó, các phụ thuộc này sẽ được "chuyển vào" đối tượng từ bên ngoài thông qua một cơ chế nào đó (chẳng hạn như qua constructor, setter hoặc interface).*

+ ASP.NET: Dependency Injection (DI) không phải là phần cốt lõi và phải cài đặt thêm để sử dụng.

+ ASP.NET Core: DI được tích hợp sẵn trong framework, giúp quản lý các phụ thuộc giữa các lớp dễ dàng hơn, và giảm sự phụ thuộc chặt chẽ giữa các thành phần trong ứng dụng.

1. **Cấu hình và Tùy biến:**

+ ASP.NET: Cấu hình chủ yếu thông qua các tệp cấu hình như web.config.

+ ASP.NET Core: Sử dụng một hệ thống cấu hình linh hoạt, có thể đọc cấu hình từ nhiều nguồn như tệp JSON, tệp môi trường, biến môi trường, hoặc thậm chí từ các dịch vụ trong đám mây.

1. **Khởi động ứng dụng:**

+ ASP.NET: Khởi động và cấu hình ứng dụng khá phức tạp, nhất là khi cần thực hiện các thay đổi lớn.

+ ASP.NET Core: Quá trình khởi động ứng dụng đơn giản hơn và linh hoạt hơn, nhờ vào việc sử dụng các tệp cấu hình và cách quản lý dịch vụ dễ dàng hơn.

1. **Tối ưu hóa cho Microservices:**

+ ASP.NET: Không được thiết kế tối ưu cho mô hình microservices.

+ ASP.NET Core: Được thiết kế đặc biệt để hỗ trợ phát triển các ứng dụng microservices, với các tính năng như hỗ trợ Docker và Kubernetes, giúp dễ dàng triển khai các dịch vụ độc lập.

1. **Quản lý bộ nhớ và Garbage Collection:**

*Garbage Collection (GC) là một cơ chế tự động trong lập trình nhằm quản lý bộ nhớ, đảm nhận việc giải phóng bộ nhớ mà chương trình không còn sử dụng nữa.*

+ ASP.NET: Hệ thống garbage collection trong ASP.NET có thể kém tối ưu hơn khi phải xử lý các ứng dụng phức tạp.

+ ASP.NET Core: Tối ưu hóa hơn về quản lý bộ nhớ và garbage collection, cải thiện hiệu suất và độ ổn định trong các ứng dụng quy mô lớn.

1. **Cộng đồng và Mã nguồn mở:**

+ ASP.NET: Là sản phẩm đóng của Microsoft, mặc dù có cộng đồng đóng góp nhưng không mã nguồn mở hoàn toàn.

+ ASP.NET Core: Là mã nguồn mở hoàn toàn và phát triển với sự tham gia của cộng đồng. Điều này giúp ASP.NET Core phát triển nhanh chóng và liên tục nhận được các cải tiến từ cộng đồng lập trình viên.

1. **Tính năng Web API và SignalR:**

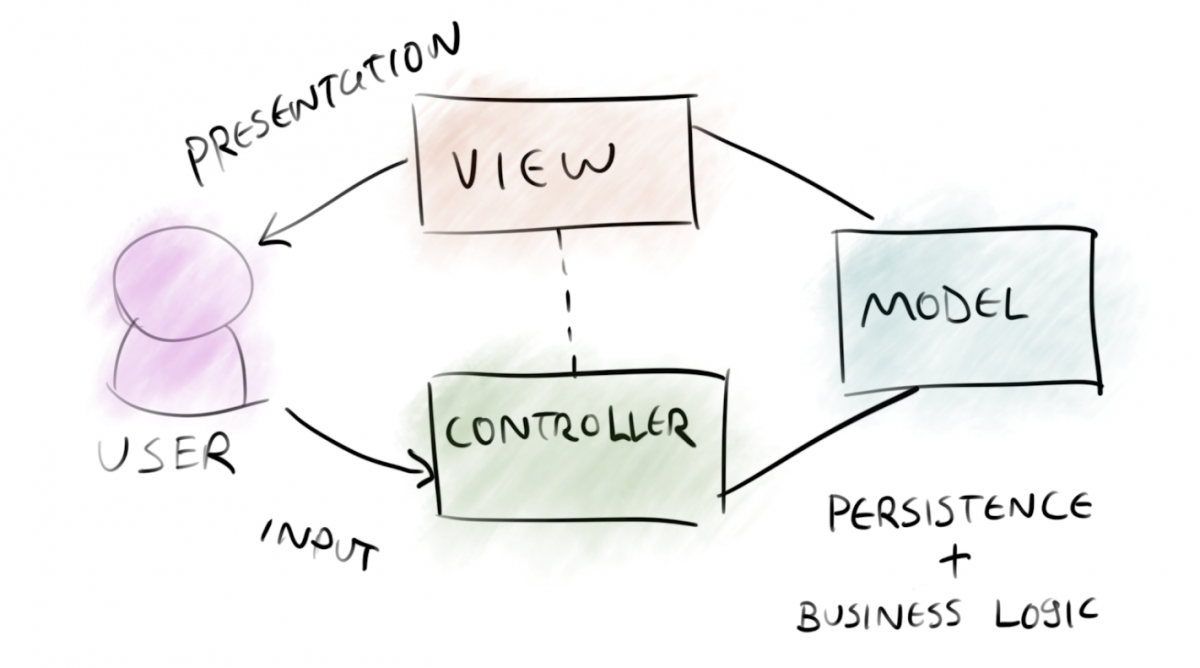
+ ASP.NET: Các tính năng như Web API và SignalR có thể được sử dụng nhưng không được tối ưu hóa cho các ứng dụng hiện đại.

+ASP.NET Core: Cung cấp các cải tiến và tính năng mới cho Web API và SignalR, giúp phát triển các ứng dụng realtime và API RESTful hiệu quả hơn.

🡺 Nhìn chung có thể thấy ASP.NET Core là phiên bản hiện đại hơn, nhanh hơn, và linh hoạt hơn của ASP.NET Nó giúp bạn phát triển các ứng dụng web với hiệu suất tối ưu, khả năng tương thích đa nền tảng, và dễ dàng bảo trì, mở rộng trong tương lai. Nếu bạn bắt đầu phát triển các ứng dụng web mới, ASP.NET Core là sự lựa chọn tốt nhất, còn ASP.NET vẫn có thể hữu ích trong những hệ thống cũ hoặc những ứng dụng không yêu cầu tính năng đa nền tảng.

## Mô hình MVC (Model-View-Controller)

Mô hình MVC (Model-View-Controller) là một kiểu kiến trúc phần mềm phổ biến trong phát triển ứng dụng web, giúp tách biệt các thành phần của ứng dụng để dễ dàng quản lý, bảo trì và mở rộng. Mô hình này phân chia ứng dụng thành ba phần riêng biệt: Model, View, và Controller, mỗi phần đảm nhiệm một chức năng cụ thể. **[2]**

Hình 2.1. Mô hình MVC

### Khái quát về cấu trúc mô hình MVC

Mô hình MVC giúp chia nhỏ ứng dụng thành ba phần cơ bản, nhằm tăng tính mô-đun và dễ dàng mở rộng. Cách phân chia này cũng giúp tăng tính tái sử dụng mã nguồn và dễ dàng kiểm thử.

**Model (M):** Là phần mô tả dữ liệu và logic nghiệp vụ của ứng dụng. Model quản lý dữ liệu, các quy tắc, và trạng thái của ứng dụng.

**View (V):** Là phần giao diện người dùng, giúp hiển thị dữ liệu từ Model ra màn hình. View chịu trách nhiệm về việc trình bày và giao diện của ứng dụng.

**Controller (C):** Là phần trung gian giữa Model và View. Controller nhận các yêu cầu từ người dùng (như nhấn nút, nhập liệu), xử lý các yêu cầu đó và thay đổi Model hoặc cập nhật View.

### Cách hoạt động của mô hình MVC

Khi một người dùng tương tác với ứng dụng (ví dụ: nhấn nút, điền biểu mẫu), các bước sau đây sẽ xảy ra:

1. Controller nhận yêu cầu từ người dùng (ví dụ: nhấn nút, chọn tùy chọn).
2. Controller xử lý yêu cầu và yêu cầu Model lấy hoặc cập nhật dữ liệu (có thể là từ cơ sở dữ liệu hoặc từ các nguồn khác).
3. Model thực hiện logic nghiệp vụ, cập nhật hoặc trả về dữ liệu cho Controller.
4. Controller nhận dữ liệu từ Model, quyết định loại giao diện hoặc hành động cần thực hiện, và yêu cầu View cập nhật giao diện người dùng.
5. View hiển thị dữ liệu cho người dùng, hoàn tất quá trình.

### Cấu trúc mô hình MVC

Model

Mô tả: Là đại diện cho dữ liệu và trạng thái của ứng dụng. Model không quan tâm đến cách hiển thị dữ liệu hoặc cách thức tương tác của người dùng với dữ liệu đó.

**Chức năng:**

* Quản lý dữ liệu, kết nối cơ sở dữ liệu.
* Thực thi các thao tác nghiệp vụ (chẳng hạn như tính toán, xác thực dữ liệu).
* Thông báo cho Controller khi dữ liệu đã thay đổi.

Ví dụ: Một lớp Product có các thuộc tính như tên sản phẩm, giá cả, số lượng trong kho.

View

Mô tả: Là giao diện người dùng, nơi dữ liệu từ Model được hiển thị cho người dùng. View chỉ chịu trách nhiệm trình bày dữ liệu, không chứa logic nghiệp vụ.

**Chức năng:**

* Hiển thị thông tin mà Controller cung cấp.
* Cung cấp các thành phần giao diện người dùng (button, form, text field) để người dùng tương tác.

Ví dụ: Một trang web hiển thị danh sách sản phẩm dưới dạng bảng hoặc thẻ.

Controller

Mô tả: Là cầu nối giữa Model và View. Controller nhận các yêu cầu từ người dùng và xử lý chúng.

**Chức năng:**

* Xử lý yêu cầu người dùng (thường là các sự kiện từ giao diện).
* Cập nhật Model (thêm, sửa, xóa dữ liệu).
* Lựa chọn View và chuyển dữ liệu tới View.

**Đoạn mã ví dụ về Controller thêm sản phẩm:**

[HttpPost]

public IActionResult Create(Product product)

{

if (ModelState.IsValid)

{

\_context.Add(product);

\_context.SaveChanges();

return RedirectToAction(nameof(Index));

}

return View(product);

}

### So sánh mô hình MVC và các mô hình khác

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Mô hình** | **Cấu trúc chính** | **Ưu điểm** | **Nhược điểm** | **Phù hợp với ứng dụng** |
| MVC | Model, View, Controller | Tách biệt rõ ràng, dễ bảo trì, dễ kiểm thử | Phức tạp hơn cho ứng dụng nhỏ, cần mã nguồn kết nối | Ứng dụng web, giao diện phức tạp |
| MVVM | Model, View, ViewModel | Tách biệt logic UI và nghiệp vụ, dễ kiểm thử, data binding | Phức tạp hơn, cần hiểu về ViewModel | Ứng dụng desktop, mobile, WPF, Xamarin |
| MVP | Model, View, Presenter | Dễ kiểm thử, logic nghiệp vụ tách biệt, giúp kiểm soát giao diện | Cần kết nối giữa Presenter và View | Ứng dụng desktop, mobile |
| SOA | Services, Clients | Tái sử dụng, phân tán tốt, mở rộng dễ dàng | Phức tạp trong quản lý, yêu cầu giao tiếp giữa các dịch vụ | Các hệ thống phân tán, Microservices |
| Clean Architecture | Entities, Use Cases, Interface Adapters, Frameworks | Tách biệt các tầng rõ ràng, dễ bảo trì và mở rộng, kiểm thử độc lập, linh hoạt trong việc thay đổi công nghệ | Phức tạp và dư thừa cho ứng dụng đơn giản | Các ứng dụng lớn, enterprise, microservices, hệ thống phức tạp |

## Một số phương thức, khái niệm, công nghệ/công cụ... khác được sử dụng

+ Phương thức **HTTP (HTTP Methods)** là các hành động được sử dụng trong giao thức HTTP (HyperText Transfer Protocol) để tương tác với tài nguyên trên một server. Mỗi phương thức HTTP xác định kiểu yêu cầu mà client gửi tới server để thực hiện một tác vụ cụ thể trên tài nguyên (như thêm, sửa, xóa, hoặc lấy dữ liệu).

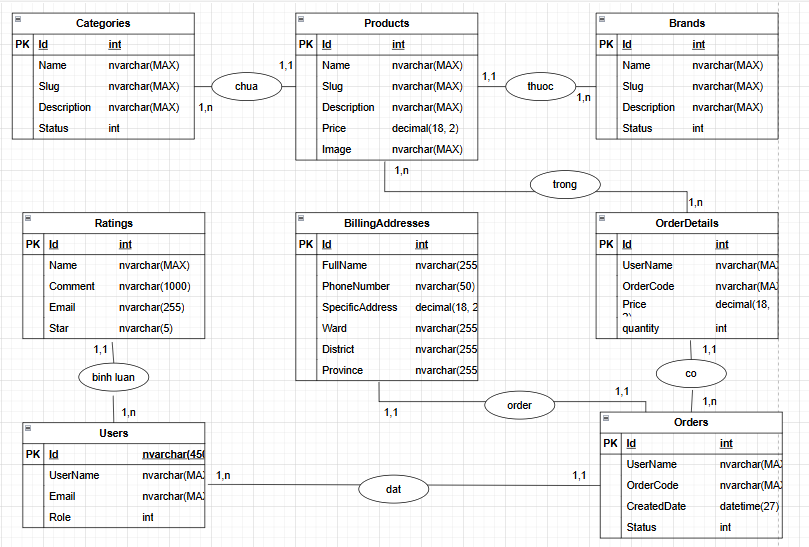
+ **IdentityUser**: một lớp được sử dụng trong **ASP.NET Core Identity**, một framework của Microsoft giúp quản lý xác thực và phân quyền cho các ứng dụng web. Đây là lớp mặc định đại diện cho một tài khoản người dùng trong hệ thống. [4]

+ **SweetAlert**: một thư viện JavaScript mã nguồn mở được sử dụng để tạo các hộp thoại (dialog) hoặc thông báo (alert) đẹp mắt, tùy chỉnh và thân thiện với người dùng. Thư viện này là một sự thay thế trực quan và cải tiến cho hộp thoại alert, confirm, và prompt mặc định của trình duyệt. [4]

+ **PdfSharpCore**: một thư viện mã nguồn mở dành cho .NET Core, được sử dụng để tạo, chỉnh sửa, và xử lý tài liệu PDF. Đây là một phiên bản mở rộng và tương thích với .NET Core của thư viện **PDFsharp**, ban đầu được thiết kế cho .NET Framework.

# THỰC NGHIỆM

## Thiết kế mô hình ERD (Entity Relationship Diagram)

Mô hình ERD (Entity Relationship Diagram) trong quản lý bán hàng là một cấu trúc dữ liệu được hình ảnh hóa để mô phỏng và thiết kế các thực thể (chẳng hạn như khách hàng, sản phẩm, đơn hàng) và mối quan hệ giữa chúng trong quá trình quản lý bán hàng của doanh nghiệp. Mô hình ERD cung cấp một cái nhìn toàn diện về các thành phần chính của hệ thống quản lý bán hàng và cách chúng tương tác với nhau.

Hình 3.1. Mô hình ERD

**1. Categories (Danh mục sản phẩm)**

Quan hệ:

+ Một danh mục (*Categories*) có thể chứa nhiều sản phẩm (*Products*).

**2. Products (Sản phẩm)**

Quan hệ:

+ Một sản phẩm thuộc về một danh mục (*Categories*).

+ Một sản phẩm có thể liên kết với một thương hiệu (*Brands*).

**3. Brands (Thương hiệu)**

Quan hệ:

+ Một thương hiệu có thể có nhiều sản phẩm (*Products*).

**4. Ratings (Đánh giá)**

Quan hệ:

+ Một sản phẩm có thể có nhiều đánh giá (*Ratings*).

**5. Users (Người dùng)**

Quan hệ:

+ Một người dùng có thể thực hiện nhiều đơn hàng (*Orders*).

**6. BillingAddresses (Địa chỉ thanh toán)**

Quan hệ:

+ Một địa chỉ thanh toán có thể gắn với nhiều đơn hàng (*Orders*).

**7. Orders (Đơn hàng)**

Quan hệ:

+ Một đơn hàng có thể chứa nhiều chi tiết đơn hàng (*OrderDetails*).

**8. OrderDetails (Chi tiết đơn hàng)**

Quan hệ:

+ Một đơn hàng có nhiều chi tiết đơn hàng, mỗi chi tiết đại diện cho một sản phẩm cụ thể.

## Thiết kế CSDL

Sử dụng ngôn ngữ SQL và ứng dụng thực hiện là SQL Server.

### Thực thể sản phẩm (Products)

*Dùng để chứa các dữ liệu có liên quan đến sản phẩm của website.*

Bảng 3.1. Sản phẩm

|  |  |  |
| --- | --- | --- |
| **SẢN PHẨM** | | |
| **Tên cột** | **Thuộc tính** |  |
| Id | Int | PK |
| Name | Nvarchar(Max) |  |
| Slug | Nvarchar(Max) |  |
| Description | Nvarchar(Max) |  |
| Price | Decimal(18, 2) |  |
| Image | Nvarchar(Max) |  |
| CategoryId | Int | FK |
| BrandId | Int | FK |

Đoạn mã SQL:

CREATE TABLE [dbo].[Products](

[Id] [bigint] IDENTITY(1,1) NOT NULL,

[Name] [nvarchar](max) NOT NULL,

[Slug] [nvarchar](max) NULL,

[Description] [nvarchar](max) NOT NULL,

[Price] [decimal](18, 2) NOT NULL,

[BrandId] [int] NOT NULL,

[CategoryId] [int] NOT NULL,

[Image] [nvarchar](max) NULL,

)

### Thực thể người dùng (Users)

*Dùng để chứa các dữ liệu có liên quan đến người dùng của website.*

*Áp dụng công nghệ của IdentityUser để hiện thực hóa việc tạo dữ liệu người dùng.*

Bảng 3.2. Người dùng

|  |  |  |
| --- | --- | --- |
| **NGƯỜI DÙNG** | | |
| **Tên cột** | **Thuộc tính** |  |
| Id | Int | PK |
| UserName | Nvarchar(Max) |  |
| Email | Nvarchar(Max) |  |
| Role | Int |  |

Đoạn mã SQL:

CREATE TABLE [dbo].[Users](

[Id] [int] IDENTITY(1,1) NOT NULL,

[UserName] [nvarchar](max) NOT NULL,

[Email] [nvarchar](max) NULL,

[Role] [int] NOT NULL

)

### Thực thể danh mục (Categories)

*Dùng để chứa các dữ liệu có liên quan đến danh mục sản phẩm của website.*

Bảng 3.3. Danh mục

|  |  |  |
| --- | --- | --- |
| **DANH MỤC** | | |
| **Tên cột** | **Thuộc tính** |  |
| Id | Int | PK |
| Name | Nvarchar(Max) |  |
| Slug | Nvarchar(Max) |  |
| Description | Nvarchar(Max) |  |
| Status | Int |  |

Đoạn mã SQL:

CREATE TABLE [dbo].[Categories](

[Id] [int] IDENTITY(1,1) NOT NULL,

[Name] [nvarchar](max) NOT NULL,

[Description] [nvarchar](max) NOT NULL,

[Slug] [nvarchar](max) NULL,

[Status] [int] NOT NULL

)

### Thực thể thương hiệu (Brands)

*Dùng để chứa các dữ liệu có liên quan đến thương hiệu sản phẩm của website.*

Bảng 3.4. Thương hiệu

|  |  |  |
| --- | --- | --- |
| **THƯƠNG HIỆU** | | |
| **Tên cột** | **Thuộc tính** |  |
| Id | Int | PK |
| Name | Nvarchar(Max) |  |
| Slug | Nvarchar(Max) |  |
| Description | Nvarchar(Max) |  |
| Status | Int |  |

Đoạn mã SQL:

CREATE TABLE [dbo].[Brands](

[Id] [int] IDENTITY(1,1) NOT NULL,

[Name] [nvarchar](max) NOT NULL,

[Description] [nvarchar](max) NOT NULL,

[Slug] [nvarchar](max) NULL,

[Status] [int] NOT NULL

)

### Thực thể đặt hàng (Orders)

*Dùng để chứa các dữ liệu có liên quan đến đơn đặt hàng của website.*

Bảng 3.5. Đơn hàng

|  |  |  |
| --- | --- | --- |
| **ĐƠN HÀNG** | | |
| **Tên cột** | **Thuộc tính** |  |
| Id | Int | PK |
| OrderCode | Nvarchar(Max) |  |
| UserName | Nvarchar(Max) |  |
| CreatedDate | DateTime |  |
| Status | Int |  |

Đoạn mã SQL:

CREATE TABLE [dbo].[Orders](

[Id] [int] IDENTITY(1,1) NOT NULL,

[OrderCode] [nvarchar](max) NULL,

[UserName] [nvarchar](max) NULL,

[CreatedDate] [datetime2](7) NOT NULL,

[Status] [int] NOT NULL

)

### Thực thể chi tiết đơn hàng (OrderDetails)

*Dùng để chứa các dữ liệu có liên quan đến chi tiết của đơn đặt hàng của website.*

Bảng 3.6. Chi tiết đơn hàng

|  |  |  |
| --- | --- | --- |
| **CHI TIẾT ĐƠN HÀNG** | | |
| **Tên cột** | **Thuộc tính** |  |
| Id | Int | PK |
| OrderCode | Nvarchar(Max) |  |
| UserName | Nvarchar(Max) |  |
| ProductId | Int | FK |
| Quantity | Int |  |

Đoạn mã SQL:

CREATE TABLE [dbo].[OrderDetails](

[Id] [int] IDENTITY(1,1) NOT NULL,

[UserName] [nvarchar](max) NULL,

[OrderCode] [nvarchar](max) NULL,

[ProductId] [bigint] NOT NULL,

[Price] [decimal](18, 2) NOT NULL,

[Quantity] [int] NOT NULL

)

### Thực thể đánh giá (Ratings)

*Dùng để chứa các dữ liệu có liên quan đến các đánh giá từ khách hàng của website.*

Bảng 3.7. Đánh giá

|  |  |  |
| --- | --- | --- |
| **ĐÁNH GIÁ** | | |
| **Tên cột** | **Thuộc tính** |  |
| Id | Int | PK |
| ProductId | Int | FK |
| Comment | Nvarchar(255) |  |
| Name | Nvarchar(255) |  |
| Email | Nvarchar(255) |  |
| Star | Nvarchar(5) |  |

Đoạn mã SQL:

CREATE TABLE [dbo].[Ratings](

[Id] [int] IDENTITY(1,1) NOT NULL,

[ProductId] [int] NOT NULL,

[Comment] [nvarchar](1000) NULL,

[Name] [nvarchar](255) NULL,

[Email] [nvarchar](255) NULL,

[Star] [nvarchar](5) NULL

)

### Thực thể địa chỉ đơn hàng (BillingAddresses)

*Dùng để chứa các dữ liệu có liên quan đến địa chỉ đặt hàng thuộc đơn hàng của website.*

Bảng 3.8. Chi tiết đơn hàng

|  |  |  |
| --- | --- | --- |
| **ĐỊA CHỈ ĐƠN HÀNG** | | |
| **Tên cột** | **Thuộc tính** |  |
| Id | Int | PK |
| OrderId | Int | FK |
| FullName | Nvarchar(255) |  |
| PhoneNumber | Nvarchar(50) |  |
| Ward | Nvarchar(255) |  |
| District | Nvarchar(255) |  |
| Province | Nvarchar(255) |  |
| SpecificAddress | Nvarchar(500) |  |

Đoạn mã SQL:

CREATE TABLE [dbo].[BillingAddresses](

[Id] [int] IDENTITY(1,1) NOT NULL,

[OrderId] [int] NOT NULL,

[FullName] [nvarchar](255) NOT NULL,

[PhoneNumber] [nvarchar](50) NOT NULL,

[SpecificAddress] [nvarchar](500) NOT NULL,

[Ward] [nvarchar](255) NOT NULL,

[District] [nvarchar](255) NOT NULL,

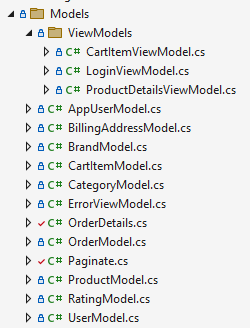
[Province] [nvarchar](255) NOT NULL

)

## Cấu trúc Models của Website

Việc tạo Model này để chứa các biến dữ liệu và truyền xuống CSDL thông qua kết nối của lớp AppDbContext.

Bao gồm 12 Model chính và 3 ViewModels:



Hình 3.2. Cấu trúc Model

**- Model:**

+ **ProductModel**: Chứa các biến dữ liệu của sản phẩm, trong đó có khai báo các biến của các danh sách khác như Category, Brands và List của RatingModel để thực hiện việc truyền dữ liệu của các thao tác của controller. Trong model này còn áp dụng IformFile để thực hiện việc lưu trữ dữ liệu hình ảnh của sản phẩm.

+ **UserModel**: Chứa các biến dữ liệu của người dùng, như Id, UserName, Password được xử lý mã hóa....

+ **AppUserModel:** Được thừa kế từ IdentityUser để thực hiện việc phân quyền của người dùng.

+ **CategoryModel:** Chứa các biến dữ liệu của danh mục, như Id, UserName, Password được xử lý mã hóa....

+ **CategoryModel**: Chứa các biến dữ liệu của danh mục, như Id, Name, Description, Slug, Status để xử lý thông tin danh mục trong hệ thống.

+ **ErrorViewModel**: Chứa thông tin xử lý lỗi. Bao gồm RequestId và ShowRequestId để hỗ trợ hiển thị lỗi khi cần thiết.

+ **OrderDetails**: Chứa thông tin chi tiết của đơn hàng, bao gồm Id, UserName, OrderCode, ProductId, Price, Quantity. Trong đó, ProductId là khóa ngoại liên kết với ProductModel.

+ **OrderModel**: Chứa thông tin của đơn hàng, bao gồm Id, OrderCode, UserName, CreatedDate, Status và một danh sách BillingAddresses để lưu trữ thông tin thanh toán liên quan.

+ **Paginate**: Hỗ trợ phân trang với các biến dữ liệu như TotalItems, PageSize, CurrentPage, TotalPages, StartPage và EndPage. Giúp hiển thị dữ liệu theo từng trang một cách hiệu quả.

+ **CartItemModel**: Lưu trữ thông tin của sản phẩm trong giỏ hàng, bao gồm ProductId, ProductName, Quantity, Price, Total, Image. Tổng giá trị giỏ hàng được tính toán tự động dựa trên Quantity và Price.

+ **BillingAddressModel**: Chứa thông tin địa chỉ thanh toán, bao gồm Id, OrderId, FullName, PhoneNumber, SpecificAddress, Ward, District, Province. Quan hệ với OrderModel thông qua OrderId.

+ **BrandModel**: Chứa các biến dữ liệu của thương hiệu, bao gồm Id, Name, Description, Slug, Status để quản lý thông tin thương hiệu trong hệ thống.

+ **RatingModel**: Lưu trữ thông tin đánh giá của sản phẩm với các thuộc tính như Id, ProductId, UserName, RatingValue, Comment, CreatedDate. ProductId liên kết với ProductModel.

**- ViewModel:**

**+ CartItemViewModel:**

Chứa các biến dữ liệu của giỏ hàng, bao gồm:

* CartItems: Danh sách CartItemModel, đại diện cho từng sản phẩm trong giỏ hàng.
* GrandTotal: Tổng giá trị của giỏ hàng, được tính dựa trên số lượng và giá của từng sản phẩm.
* Hiển thị thông tin chi tiết về giỏ hàng của người dùng, bao gồm danh sách sản phẩm, số lượng, giá cả, và tổng tiền.
* Được sử dụng trong View để tạo giao diện giỏ hàng, hỗ trợ người dùng kiểm tra và quản lý sản phẩm trước khi thanh toán.
* Hỗ trợ xử lý thêm, xóa, cập nhật sản phẩm trong giỏ hàng.

**+ LoginViewModel:**

* Chứa các biến dữ liệu cần thiết cho việc đăng nhập, bao gồm:
* Id: Mã định danh của người dùng.
* UserName: Tên đăng nhập của người dùng (cần nhập).
* Password: Mật khẩu đăng nhập (được xử lý với DataType.Password để tăng cường bảo mật).
* ReturnUrl: Đường dẫn trả về sau khi đăng nhập thành công (hỗ trợ điều hướng người dùng đến trang mong muốn).
* Thu thập và xác thực thông tin đăng nhập của người dùng.
* Hỗ trợ điều hướng người dùng sau khi đăng nhập thành công (ReturnUrl).
* Được sử dụng trong giao diện đăng nhập để xử lý logic xác thực tài khoản.

**+ ProductDetailsViewModel:**

Chứa các biến dữ liệu để hiển thị chi tiết sản phẩm, bao gồm:

* ProductDetails: Thông tin chi tiết sản phẩm (đối tượng ProductModel), bao gồm tên, giá, mô tả, hình ảnh, v.v.
* Ratings: Danh sách các đánh giá sản phẩm (RatingModel), hiển thị nhận xét và đánh giá của người dùng.
* Comment: Bình luận mà người dùng muốn gửi về sản phẩm.
* Name: Tên của người gửi bình luận.
* Email: Email của người gửi bình luận.
* Star: Đánh giá sao của sản phẩm.
* Hiển thị giao diện chi tiết của một sản phẩm cụ thể, bao gồm mô tả, hình ảnh, giá cả và các đánh giá từ khách hàng khác.
* Cho phép người dùng thêm nhận xét, đánh giá sản phẩm theo sao và gửi bình luận.
* Cung cấp dữ liệu cần thiết để tạo trang chi tiết sản phẩm thân thiện và đầy đủ thông tin.

## Cấu trúc Controllers của Website

Trong website đề này tài Controller của Admin được tách ra thành 1 Project nằm trong Solution của Website để tiện cho việc xây dựng View cho trang quản lý tách biệt với trang chính của Website và đơn giản hóa việc phân quyền hơn. Do đó Controller sẽ có 2 cấu trúc riêng và thực hiện riêng các chức năng, tuy nhiên vẫn dựa vào một cấu trúc Model duy nhất.

### Cấu trúc Controllers của trang chính

Hình 3.3. Cấu trúc Controller trang chính

HomeController:

* **Index()**: Hiển thị trang chủ với danh sách sản phẩm, bao gồm thông tin danh mục (Category) và thương hiệu (Brand).
* **Privacy()**: Hiển thị trang chính sách bảo mật.
* **Error(int statuscode)**: Xử lý các lỗi HTTP. Hiển thị trang lỗi tương ứng, ví dụ:

**+ 404**: Hiển thị trang "Not Found".

+ Các lỗi khác: Hiển thị thông tin lỗi dựa trên ErrorViewModel.

ProductController:

* **Index()**: Hiển thị trang danh sách sản phẩm.
* **Details(int? Id)**: Hiển thị chi tiết sản phẩm dựa trên Id. Bao gồm sản phẩm liên quan và danh sách đánh giá.
* **Search(string searchTerm)**: Tìm kiếm sản phẩm dựa trên từ khóa trong tên hoặc mô tả.
* **CommentProduct(RatingModel rating)**: Xử lý đánh giá sản phẩm, thêm đánh giá vào cơ sở dữ liệu và hiển thị thông báo thành công hoặc lỗi.

CheckoutController:

* **Checkout()**: Kiểm tra thông tin người dùng, tạo mã đơn hàng mới và hiển thị trang nhập thông tin thanh toán.
* **SaveBillingAddress(BillingAddressModel billingAddress)**:
* Xử lý lưu thông tin thanh toán và đơn hàng vào cơ sở dữ liệu.
* Lưu chi tiết đơn hàng từ giỏ hàng.
* Gửi email thông báo xác nhận đơn hàng đến người dùng.

CategoryController:

* **GetCategories()**: Trả về danh sách các danh mục dưới dạng một PartialView, giúp hiển thị danh sách danh mục nhanh chóng trên giao diện.
* **Index(string Slug)**:
* Lấy danh sách sản phẩm thuộc danh mục được chỉ định thông qua Slug.
* Hiển thị danh sách sản phẩm trong danh mục đó trên giao diện.
* Nếu không tìm thấy danh mục, chuyển hướng về trang chính (Index).

CartController:

* **Index()**: Hiển thị giỏ hàng với danh sách sản phẩm, số lượng và tổng giá trị giỏ hàng.
* **Checkout()**: Chuyển hướng đến trang thanh toán.
* **Add(int Id)**:
  + Kiểm tra xem người dùng đã đăng nhập chưa.
  + Thêm sản phẩm vào giỏ hàng nếu chưa có hoặc tăng số lượng nếu sản phẩm đã tồn tại trong giỏ hàng.
  + Lưu trạng thái giỏ hàng vào Session và trả về thông báo thành công.
* **Decrease(int Id)**: Giảm số lượng sản phẩm trong giỏ hàng. Nếu số lượng là 1, xóa sản phẩm khỏi giỏ hàng.
* **Increase(int Id)**: Tăng số lượng sản phẩm trong giỏ hàng.
* **Remove(int Id)**: Xóa sản phẩm có Id cụ thể ra khỏi giỏ hàng.
* **Clear(int Id)**: Xóa toàn bộ sản phẩm trong giỏ hàng và làm trống Session.

BrandController:

* **Index(string Slug)**:
* Lấy thông tin thương hiệu dựa trên Slug.
* Hiển thị danh sách sản phẩm thuộc thương hiệu tương ứng.
* Nếu thương hiệu không tồn tại, chuyển hướng về trang danh sách mặc định (Index).

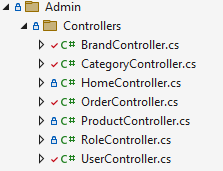
AccountController:

* **Login(string returnUrl)**: Hiển thị trang đăng nhập. Nhận đường dẫn trả về sau khi đăng nhập thành công.
* **Login(LoginViewModel loginVM)**:
* Xác thực thông tin đăng nhập từ người dùng.
* Chuyển hướng về returnUrl nếu đăng nhập thành công, hoặc hiển thị thông báo lỗi nếu thông tin không hợp lệ.
* **Create()**: Hiển thị trang tạo tài khoản người dùng.
* **Create(UserModel user)**:
* Tạo tài khoản mới dựa trên thông tin người dùng cung cấp.
* Nếu tạo tài khoản thành công, hiển thị thông báo và chuyển hướng đến trang đăng nhập.
* Nếu có lỗi, hiển thị danh sách lỗi.
* **Logout(string returnUrl = "/")**:
* Đăng xuất người dùng khỏi hệ thống.
* Chuyển hướng về returnUrl (mặc định là trang chủ).

ViewController:

* **Index**: Trả về trang About.

### Cấu trúc Controllers của trang Admin



Hình 3.4. Cấu trúc Controllers trang Admin

UserController:

* + **Index()**: Hiển thị danh sách người dùng cùng vai trò của họ bằng cách liên kết dữ liệu từ các bảng Users, UserRoles, và Roles.
  + **Create() [GET]**: Hiển thị form để tạo người dùng mới, bao gồm danh sách vai trò có sẵn.
  + **Create(AppUserModel user) [POST]**:
* Tạo người dùng mới trong hệ thống.
* Kiểm tra trùng tên người dùng, xác thực thông tin, và gán vai trò nếu hợp lệ.
* Nếu tạo thành công, chuyển hướng về danh sách người dùng; nếu thất bại, hiển thị lỗi.
* **Edit(string id) [GET]**: Hiển thị form chỉnh sửa thông tin người dùng dựa trên id.
* **Edit(string id, AppUserModel user) [POST]**:
  + - Cập nhật thông tin người dùng hiện tại.
    - Xóa vai trò cũ và gán vai trò mới nếu được chỉ định.
    - Hiển thị thông báo thành công hoặc lỗi khi cập nhật.
  + **Delete(string id):**
  + Xóa người dùng khỏi hệ thống dựa trên id.
  + Nếu thành công, chuyển hướng về danh sách người dùng; nếu thất bại, hiển thị trang lỗi.
  + **AddIdentityErrors:** Phương thức hỗ trợ hiển thị danh sách lỗi từ kết quả của các thao tác Identity.

RoleController:

* **Index()**: Hiển thị danh sách các vai trò (Roles) được sắp xếp giảm dần theo Id.
* **Create() [GET]**: Hiển thị form để tạo vai trò mới.
* **Create(IdentityRole model) [POST]**:
  + Kiểm tra xem vai trò đã tồn tại chưa.
  + Nếu chưa tồn tại, tạo vai trò mới.
  + Hiển thị thông báo thành công hoặc lỗi khi thực hiện thao tác.
* **Edit(string id) [GET]**:
  + Lấy thông tin vai trò theo id và hiển thị form chỉnh sửa.
  + Nếu id không hợp lệ hoặc vai trò không tồn tại, hiển thị thông báo lỗi.
* **Edit(IdentityRole model) [POST]**:
  + Cập nhật tên vai trò.
  + Hiển thị thông báo thành công nếu cập nhật thành công, hoặc thông báo lỗi nếu có vấn đề xảy ra.
* **Delete(string id) [GET]**:
  + Xóa vai trò dựa trên id.
  + Hiển thị thông báo thành công hoặc lỗi tùy thuộc vào kết quả thao tác.

ProductController:

* **Index()**: Hiển thị danh sách sản phẩm, bao gồm thông tin danh mục, được sắp xếp giảm dần theo Id.
* **Create() [GET]**: Hiển thị form thêm sản phẩm mới, bao gồm danh sách danh mục và thương hiệu.
* **Create(ProductModel product) [POST]**:
  + Thêm sản phẩm mới vào cơ sở dữ liệu.
  + Kiểm tra trùng lặp Slug, xử lý tải và lưu trữ hình ảnh sản phẩm.
  + Hiển thị thông báo thành công hoặc lỗi tùy thuộc vào kết quả thao tác.
* **Edit(long Id) [GET]**: Hiển thị form chỉnh sửa thông tin sản phẩm dựa trên Id, kèm danh sách danh mục và thương hiệu.
* **Edit(ProductModel product) [POST]**:
  + Cập nhật thông tin sản phẩm hiện tại.
  + Kiểm tra trùng lặp Slug, xử lý thay thế ảnh cũ bằng ảnh mới (nếu có).
  + Lưu thay đổi vào cơ sở dữ liệu và hiển thị thông báo kết quả.
* **Delete(long Id)**:
  + Xóa sản phẩm khỏi cơ sở dữ liệu dựa trên Id.
  + Xóa ảnh sản phẩm cũ khỏi thư mục lưu trữ.
  + Hiển thị thông báo thành công hoặc lỗi nếu có vấn đề trong quá trình xóa.

OrderController:

* **Index(int pg = 1)**: Hiển thị danh sách đơn hàng với phân trang, mỗi trang hiển thị tối đa 10 đơn hàng.
* **UpdateOrder(string ordercode, int status)**:
  + Cập nhật trạng thái đơn hàng dựa trên ordercode và trạng thái mới (status).
  + Trả về JSON thông báo kết quả và đường dẫn điều hướng.
* **DeleteOrder(string ordercode)**:
  + Xóa đơn hàng dựa trên ordercode, bao gồm chi tiết đơn hàng và địa chỉ thanh toán liên quan.
  + Hiển thị thông báo thành công hoặc lỗi nếu xảy ra vấn đề.
* **ViewOrder(string ordercode)**:
  + Hiển thị chi tiết đơn hàng dựa trên ordercode, bao gồm danh sách sản phẩm và địa chỉ thanh toán.
  + Sử dụng ViewData để truyền thông tin đơn hàng và địa chỉ thanh toán đến View.
* **ExportOrderToPdf(string ordercode)**:
  + Xuất thông tin đơn hàng thành tài liệu PDF.
  + Bao gồm thông tin khách hàng, chi tiết sản phẩm và tổng giá trị đơn hàng.
  + Sử dụng thư viện PdfSharpCore để tạo tài liệu PDF và trả về file dưới dạng phản hồi tải xuống.

CategoryController:

* **Index(int pg = 1)**:
  + Hiển thị danh sách danh mục với phân trang, mỗi trang hiển thị tối đa 10 danh mục.
  + Tính toán và truyền dữ liệu phân trang (Paginate) đến View.
* **Edit(int Id) [GET]**:
  + Lấy thông tin danh mục dựa trên Id và hiển thị trong form chỉnh sửa.
* **Edit(CategoryModel category) [POST]**:
  + Cập nhật thông tin danh mục.
  + Kiểm tra trùng lặp Slug để tránh xung đột dữ liệu.
  + Hiển thị thông báo thành công hoặc lỗi khi chỉnh sửa.
* **Create() [GET]**: Hiển thị form để thêm danh mục mới.
* **Create(CategoryModel category) [POST]**:
  + Tạo danh mục mới trong cơ sở dữ liệu.
  + Kiểm tra trùng lặp Slug.
  + Hiển thị thông báo kết quả khi thêm mới thành công hoặc lỗi nếu có vấn đề.
* **Delete(int Id)**:
  + Xóa danh mục dựa trên Id.
  + Hiển thị thông báo thành công hoặc lỗi nếu xảy ra vấn đề trong quá trình xóa.

BrandController:

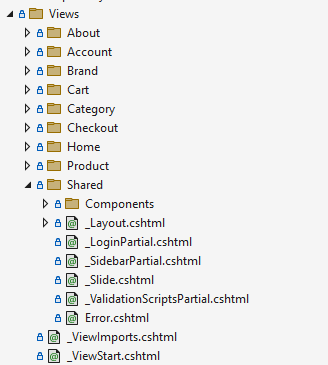
* **Index(int pg = 1)**:
  + Hiển thị danh sách thương hiệu với phân trang, mỗi trang hiển thị tối đa 10 thương hiệu.
  + Truyền dữ liệu phân trang (Paginate) đến View.
* **Create() [GET]**: Hiển thị form để thêm thương hiệu mới.
* **Create(BrandModel brand) [POST]**:
  + Thêm thương hiệu mới vào cơ sở dữ liệu.
  + Kiểm tra trùng lặp Slug để đảm bảo tính duy nhất.
  + Hiển thị thông báo thành công hoặc lỗi khi thao tác thêm mới.
* **Edit(int id) [GET]**:
  + Lấy thông tin thương hiệu dựa trên id và hiển thị trong form chỉnh sửa.
  + Nếu không tìm thấy, trả về lỗi NotFound.
* **Edit(BrandModel brand) [POST]**:
  + Cập nhật thông tin thương hiệu hiện tại.
  + Kiểm tra trùng lặp Slug để đảm bảo tính duy nhất.
  + Hiển thị thông báo kết quả khi chỉnh sửa thành công hoặc lỗi nếu xảy ra vấn đề.
* **Delete(int Id)**:
  + Xóa thương hiệu dựa trên Id.
  + Hiển thị thông báo thành công hoặc lỗi nếu có vấn đề trong quá trình xóa.

## Cấu trúc Views của Website

Ở phần View này sẽ xử lý các tác vụ hiển thị, truyền dữ liệu từ người dùng xuống Controller xử lý.

Tương tự phần Controller, View cũng được chia thành hai phần cho cả hai trang để dễ dàng áp dụng các HTML và Extention riêng của nhau:

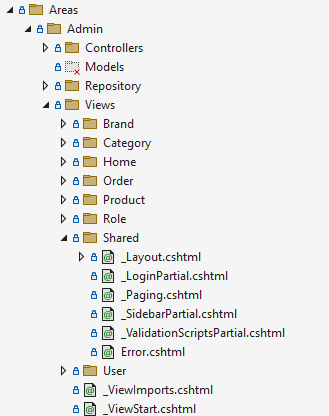
### Cấu trúc Views của trang chính



Hình 3.5. Cấu trúc Views Trang chính

\_Layout.html là trang chính thể hiện giao diện tổng qua của website và chứa các phần tử con khi người dùng truy cập đến thông qua Function **@RenderBody()** thuộc Razer để truyền đến các trang như là: Product, Account... chứa tổng thể của trang html đảm nhận việc hiện thị và xử lý xửa liệu đầu vào.

### Cấu trúc Views của trang Admin



Hình 3.6. Cấu trúc Views Trang Admin

Tương tự như trang chính phần \_Layout.html của trang admin là trang chính thể hiện giao diện tổng qua của website và chứa các phần tử con khi người dùng truy cập đến thông qua Function **@RenderBody()** thuộc Razer để truyền đến các trang như là: Product, Account... chứa tổng thể của trang html đảm nhận việc hiện thị và xử lý xửa liệu đầu vào.

# KẾT QUẢ NGHIÊN CỨU

*(Chi tiết Sourcecode của đồ án sau:* [*https://github.com/BHieeuss/ChuyenNganh-TranBaHieu-110121024*](https://github.com/BHieeuss/ChuyenNganh-TranBaHieu-110121024)*)*

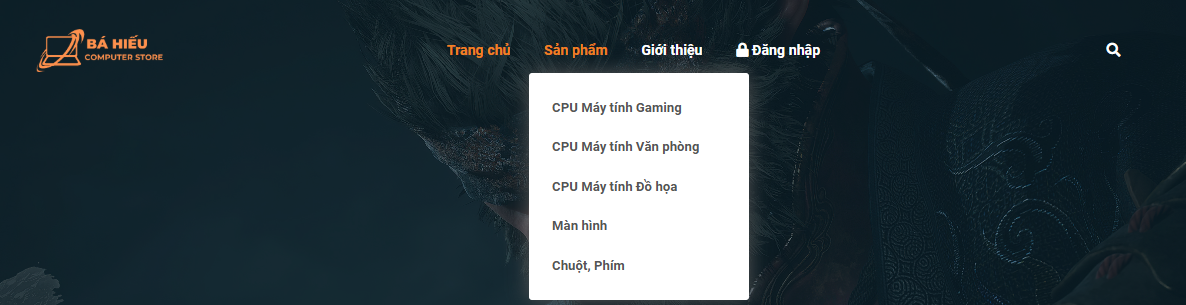
*Lưu ý chương 4:*

*+ Một số phần vì Sourcecode quá dài nên sẽ trực tiếp giải thích quy trình hoặc bỏ qua nếu không cần thiết.*

*+ Các thông báo của website được sử dụng thông qua Extention SweetAlert https://sweetalert2.github.io/.*

## Phần trang chính

### Giao diện phần Header



Hình 4.1. Giao diện Header

Tại Header sẽ bao gồm các phần chính như:

**Logo:** Bên trái là logo của thương hiệu cửa hàng

Các mục trên thanh điều hướng:

**Trang chủ:** Dùng để điều hướng đến trang chủ, được đánh dấu bằng màu cam, thể hiện rằng đây là mục hiện đang được chọn.

**Sản phẩm:** Dùng để điều hướng đến các trang sản phẩm thông qua các danh mục được Dropdown xuống khi người dùng di chuột vào liên kết đến danh mục sản phẩm.

**Giới thiệu:** Dùng để điều hướng đến trang giới thiệu của cửa hàng, mục để tìm hiểu thêm thông tin về cửa hàng.

**Đăng nhập:** Dùng để điều hướng đến trang đăng nhập có biểu tượng ổ khóa, có thể dẫn đến trang đăng nhập. Mục này sẽ thay đổi khi người dùng đã đăng nhập thành công.

Hình 4.2. Giao diện Header sau khi đăng nhập

Phần này chủ yếu xử lý các Route chuyển hướng và thực hiện gọi danh sách danh mục từ Controller của Danh mục để hiển thị. Và các thẻ If hỗ trợ việc xác thực đã đăng nhập thành công.

### Giao diện Banner

Hình 4.3. Giao diện banner

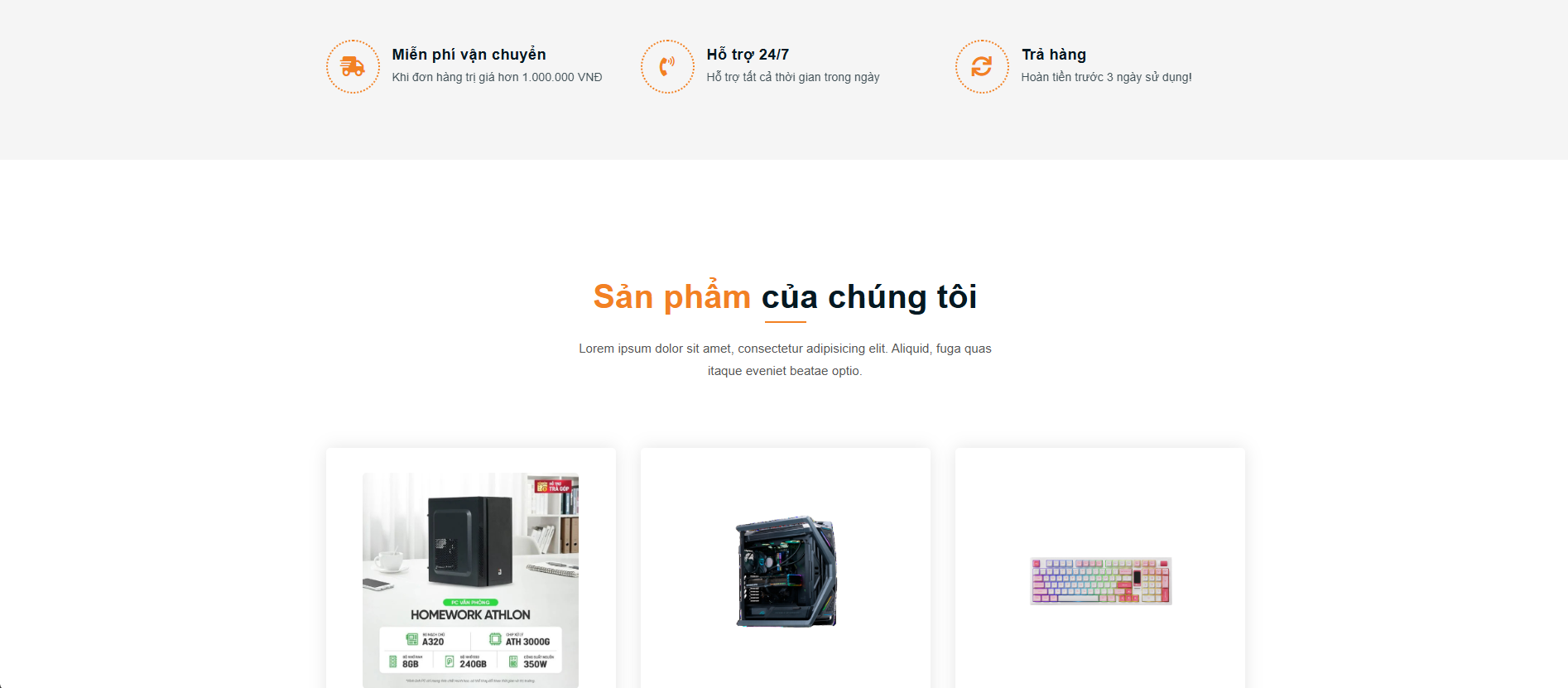
Ở mục xây dựng Banner phần lớn là xây dựng dựa trên các thẻ HTML và CSS không có sự can thiệp của các Controller xử lý tác vụ.

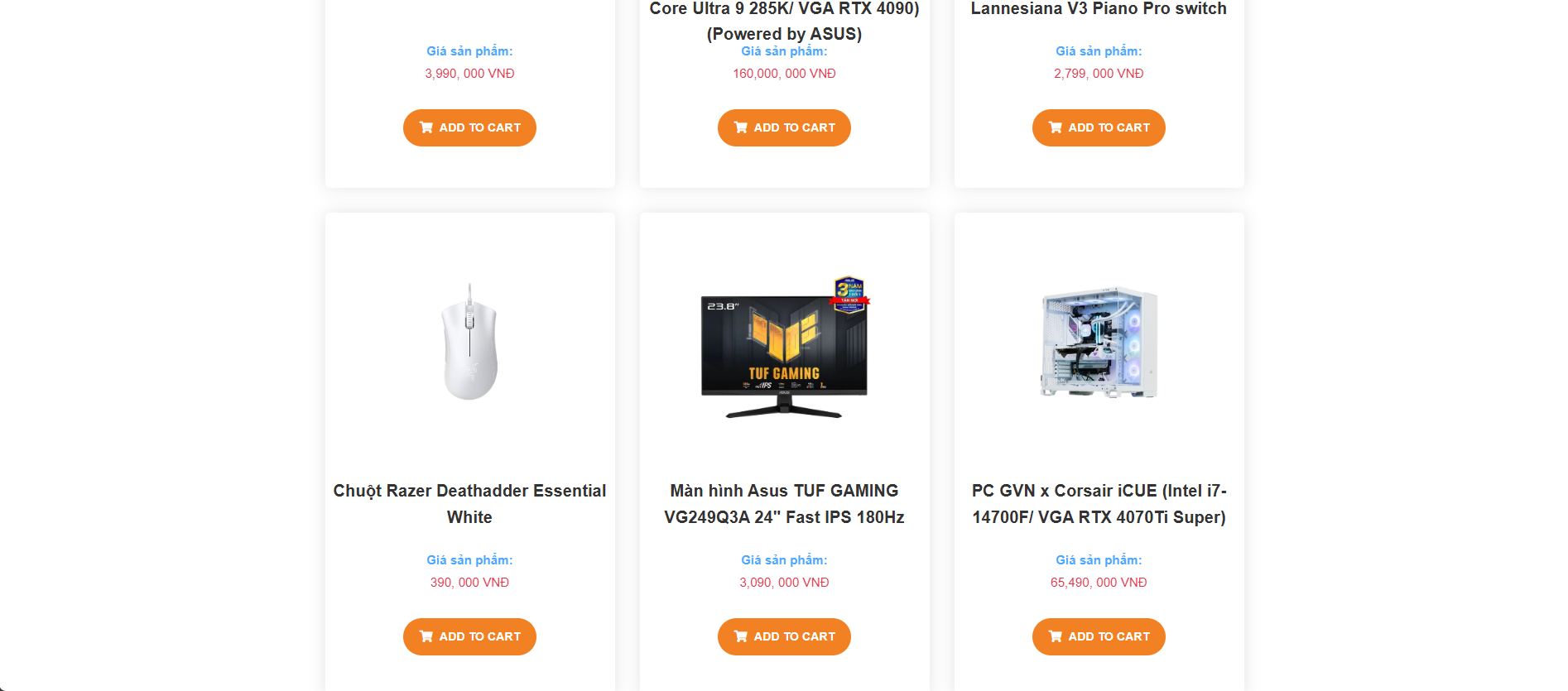
### Giao diện Footer

Hình 4.4. Giao diện footer

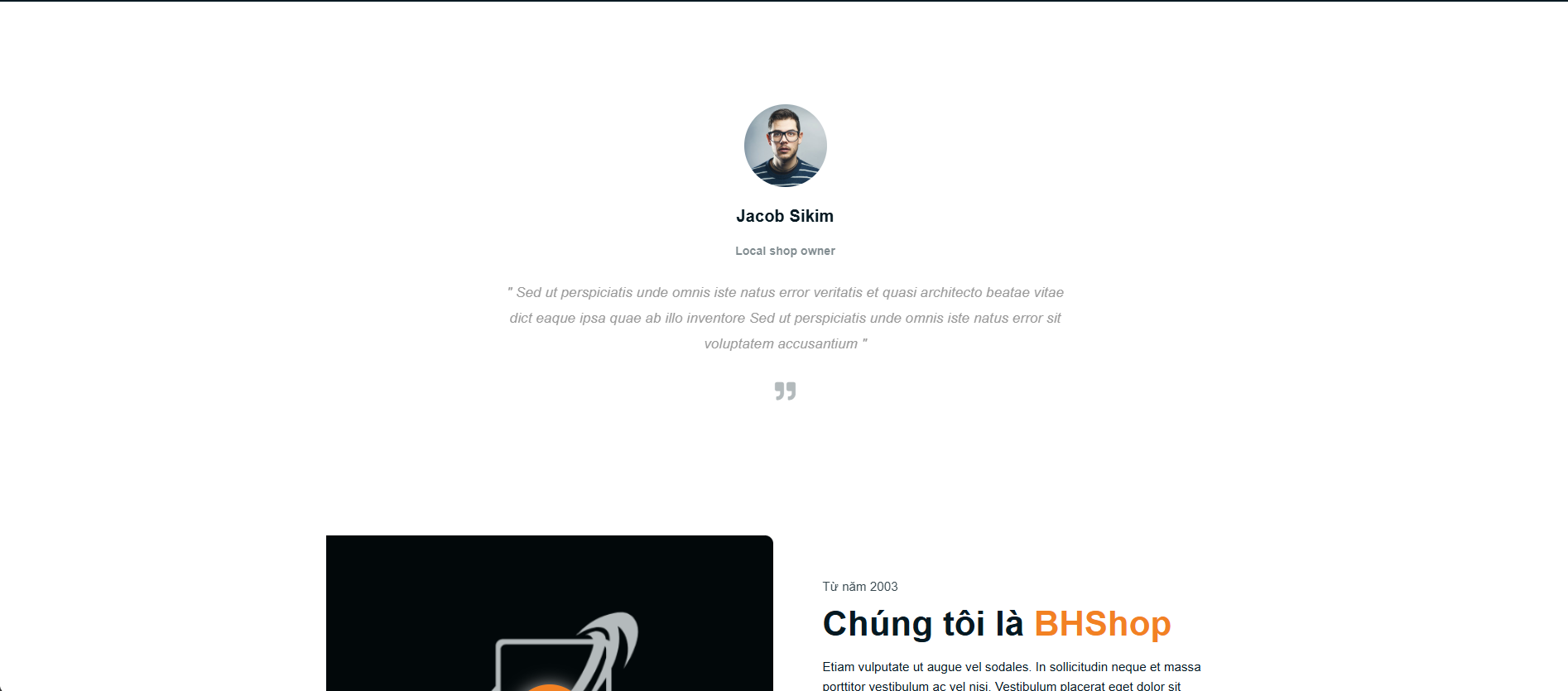
Ở mục xây dựng Footer phần lớn là xây dựng dựa trên các thẻ HTML và CSS không có sự can thiệp của các Controller xử lý tác vụ.

### Tổng thể giao diện trang chủ



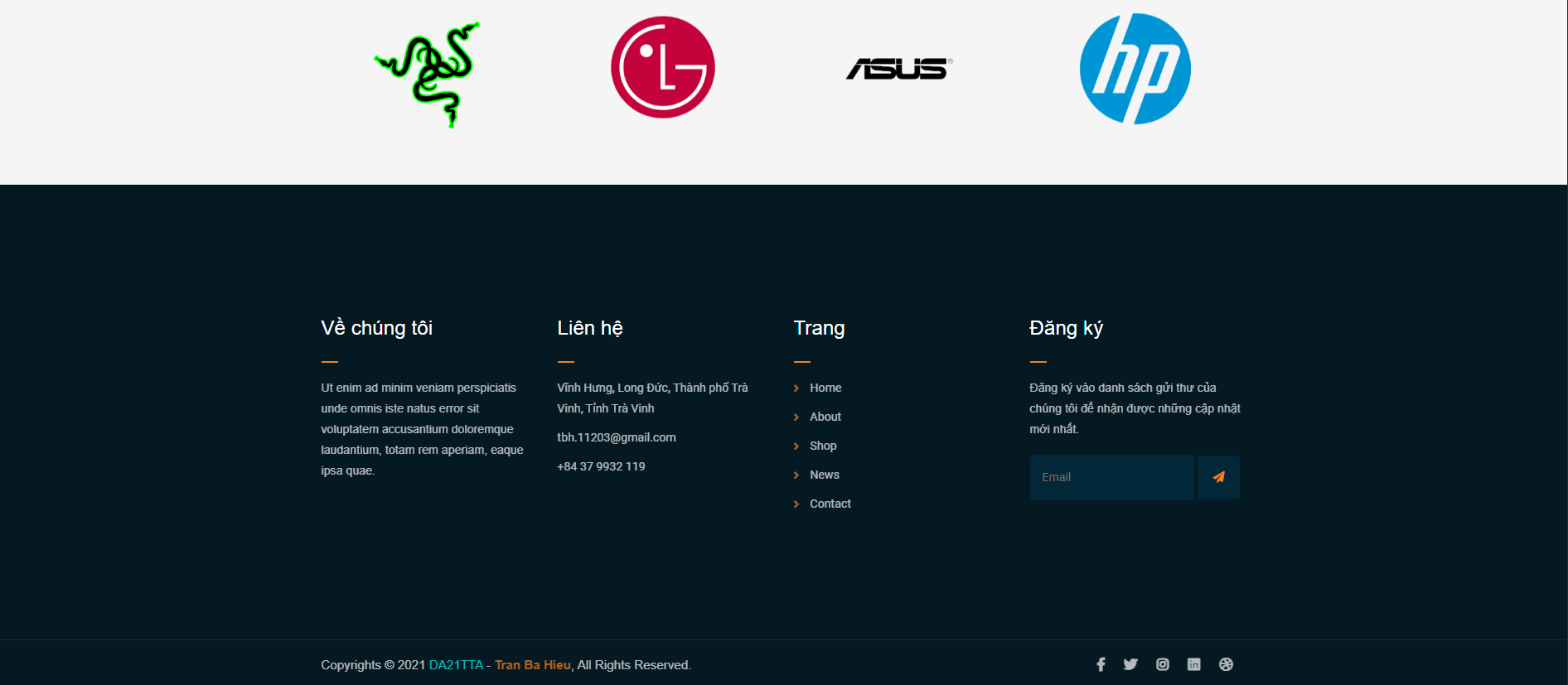








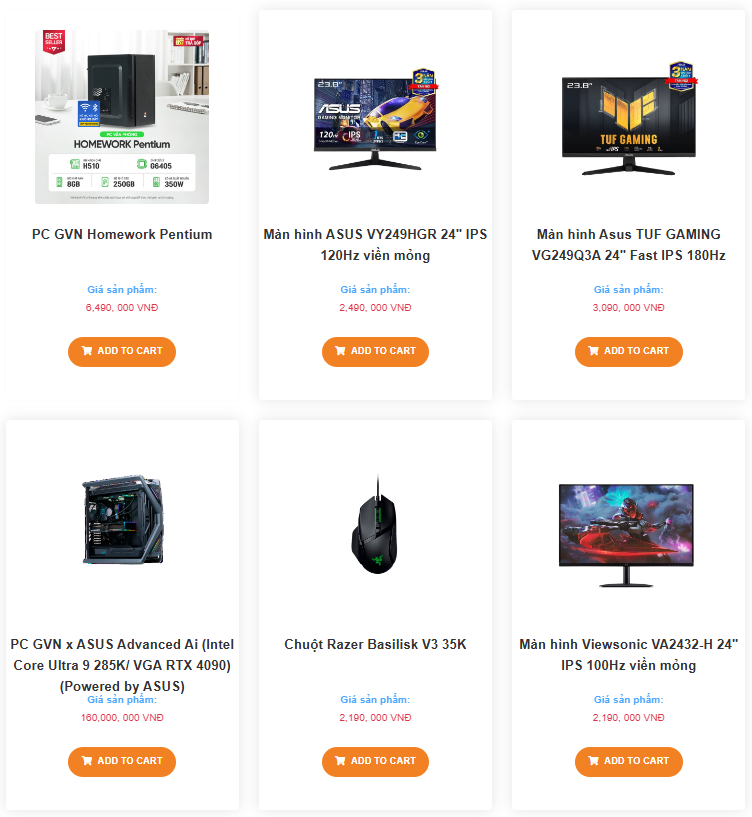




Hình 4.5 Giao diện trang chủ

*( Vì lí do giao diện khá dài nên xin được cắt ghép từ nhiều ảnh)*

Tại trang chủ phần lớn là các khối HTML xử lý việc hiện thị cho người dùng xem giao diện.

 Tại phần này có sự can thiệp của Controller để xử lý việc truyền ngẫu nhiên 6 sản phẩm từ CSDL, chi tiết như sau:

Hình 4.6. Phần sản phẩm của chúng tôi

public IActionResult Index()

{

var randomProducts = \_dataContext.Products

.Include("Category")

.Include("Brand")

.OrderBy(p => Guid.NewGuid())

.Take(6)

.ToList();

return View(randomProducts);

}

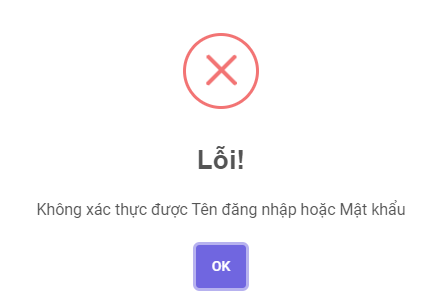
Đầu tiên sẽ khai báu một biến **randomProducts** được get lên từ dataContext đã khai báo, sau đó gọi từ hai Model Category và Brand để lấy danh sách sản phẩm, từ đó lấy ra 6 sản phẩm vào đưa vào một danh sách. Cuối cùng là trả return về View của trang chủ với biến **randomProducts**.

### Giao diện trang đăng nhập và chức năng đăng nhập

Hình 4.7. Giao diện trang đăng nhập

Giao diện này được xây dựng khá đơn giản và tối ưu, dựa trên các thẻ HTML CSS và Bootstrap5.

Phần quan trọng nhất nó sự can thiệp của AccountController thông qua Identity để thực hiện việc xác thực đăng nhập.



Hình 4.8. Thông báo lỗi khi tên đăng nhập hoặc mật khẩu

không đúng

[HttpPost]

public async Task<IActionResult> Login(LoginViewModel loginVM)

{

if(ModelState.IsValid)

{

Microsoft.AspNetCore.Identity.SignInResult result = await \_signInManager.PasswordSignInAsync(loginVM.UserName, loginVM.Password, false,false);

if (result.Succeeded)

{

return Redirect(loginVM.ReturnUrl ?? "/");

}

TempData["error"] = "Không xác thực được Tên đăng nhập hoặc Mật khẩu";

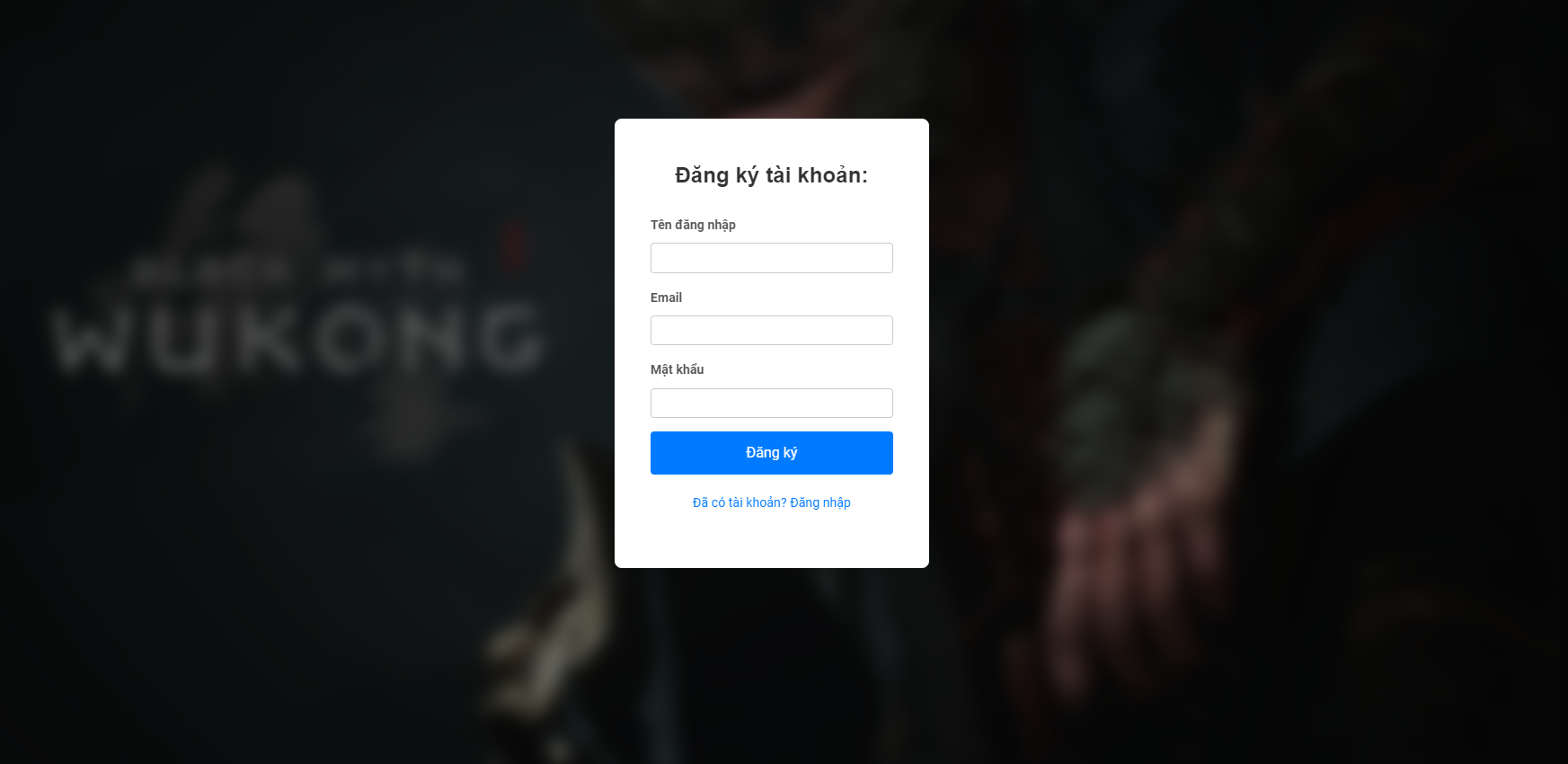
}

return View(loginVM);

}

Phương thức xử lý đăng nhập sẽ kiểm tra tính hợp lệ của dữ liệu từ LoginViewModel, sau đó gọi \_signInManager.PasswordSignInAsync để xác thực người dùng, nếu thành công sẽ chuyển hướng đến ReturnUrl hoặc trang chủ, nếu thất bại sẽ lưu thông báo lỗi vào TempData và trả lại View cùng thông tin đăng nhập.

### Giao diện trang đăng ký và chức năng đăng ký



Hình 4.9. Giao diện trang đăng nhập

Giao diện này được xây dựng khá đơn giản và tối ưu, dựa trên các thẻ HTML CSS và Bootstrap5.

Phần quan trọng nhất nó sự can thiệp của AccountController thông qua Identity để thực hiện việc tạo một User mới.

public IActionResult Create()

{

return View();

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Create(UserModel user)

{

if (ModelState.IsValid)

{

AppUserModel newUser = new AppUserModel { UserName =user.UserName, Email = user.Email};

IdentityResult result = await \_userManager.CreateAsync(newUser, user.Password);

if (result.Succeeded)

{

TempData["success"] = "Tạo tài khoản thành công";

return Redirect("/account/login");

}

foreach(IdentityError error in result.Errors)

{

ModelState.AddModelError("", error.Description);

}

}

return View(user);

}

public async Task<IActionResult> Logout( string returnUrl = "/")

{

await \_signInManager.SignOutAsync();

return Redirect(returnUrl);

}

}

Đầu tiên, phương thức Create (**GET**) sẽ trả về view để hiển thị form tạo tài khoản, tiếp đến phương thức Create (**POST**) kiểm tra tính hợp lệ của dữ liệu từ UserModel, sau đó gọi **\_userManager.CreateAsync** để tạo tài khoản mới, nếu thành công sẽ lưu thông báo thành công vào **TempData** và chuyển hướng đến trang đăng nhập, nếu thất bại sẽ thêm lỗi vào **ModelState** và trả về view kèm thông tin nhập. Cuối cùng, phương thức **Logout** sẽ gọi \_**signInManager**.**SignOutAsync** để đăng xuất người dùng và chuyển hướng về **returnUrl**.

### Giao diện trang sản phẩm

Hình 4.10. Giao diện trang sản phẩm

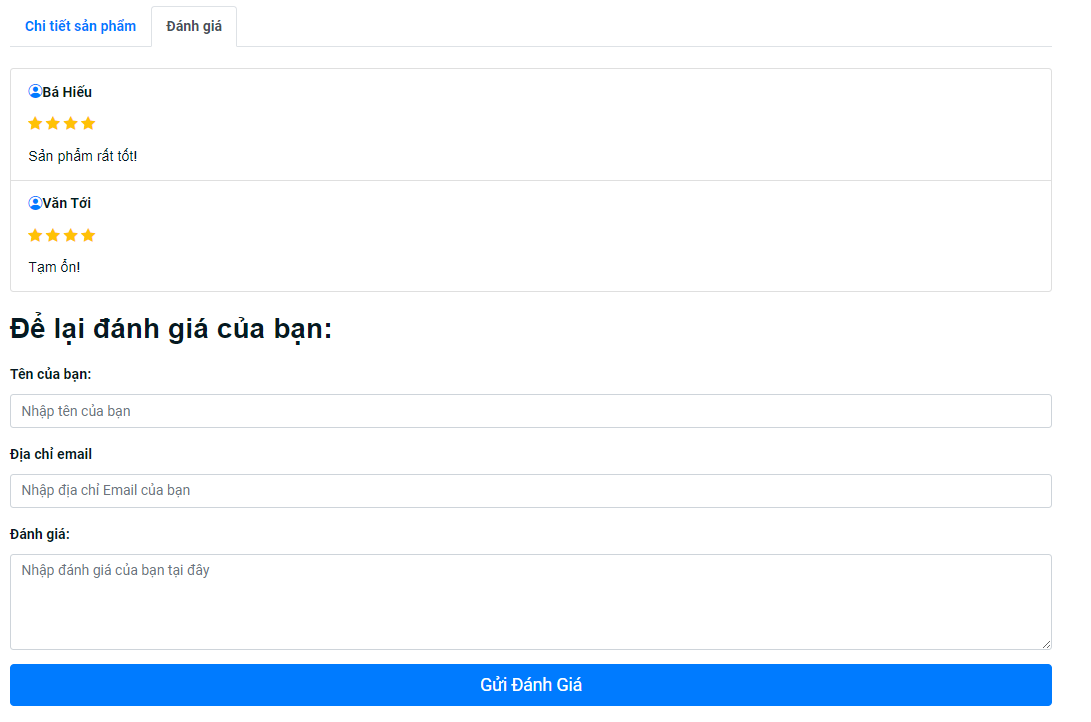
Tại trang này chủ yếu là Get lên danh sách các sản phẩm dựa vào Category.

### Giao diện trang chi tiết sản phẩm

Hình 4.11. Giao diện chi tiết sản phẩm

Trang này được xử lý trừ Controller cho việc get dữ liệu của sản phẩm dựa vào Id và các sản phẩm liên quan, thực hiện việc tạo đánh giá sản phẩm.

Hình 4.12. Form mô tả sản phẩm



Hình 4.13. Form đánh giá

*( Xem mã nguồn chi tiết tại Github )*

Chủ yếu việc tạo đánh giá và các ràng buộc liên quan vào sản phẩm được thêm vào RatingModel

var ratingEntity = new RatingModel

{

ProductId = rating.ProductId,

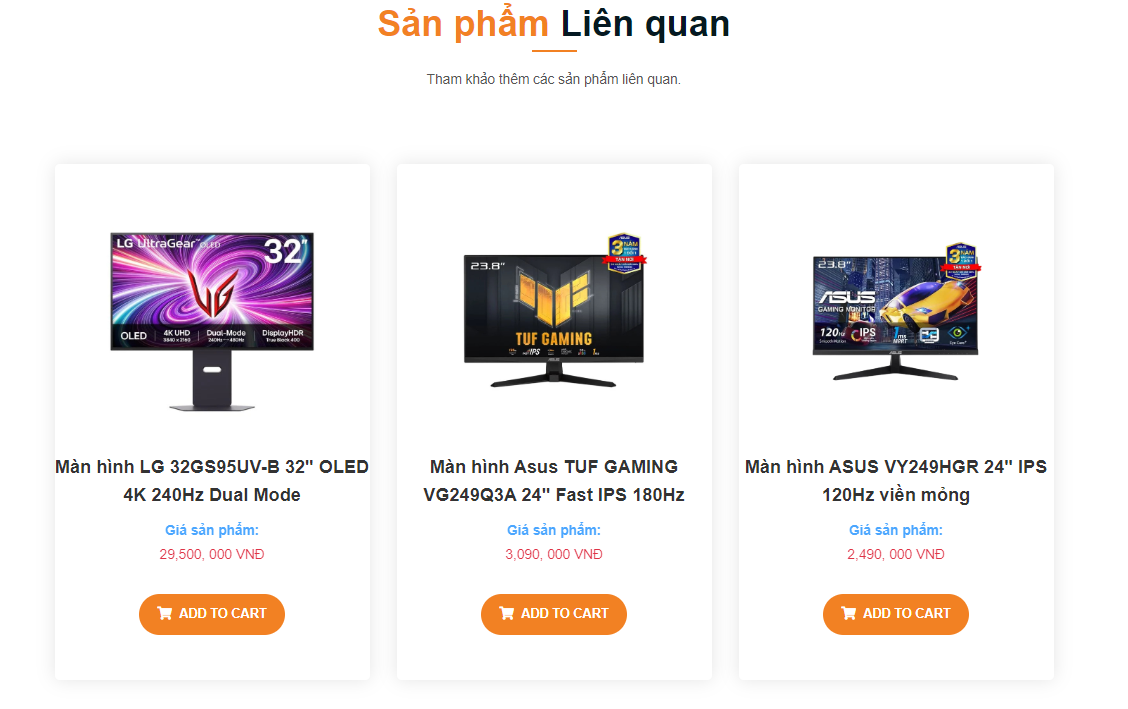
Name = rating.Name,

Email = rating.Email,

Comment = rating.Comment,

Star = rating.Star

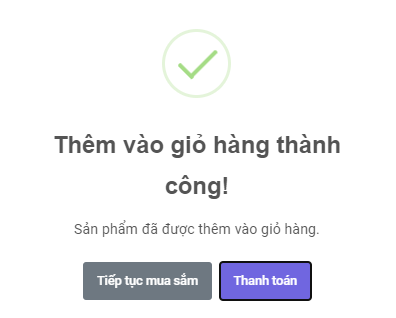
};



Hình 4.14. Mục các sản phẩm liên quan

*( Xem mã nguồn chi tiết tại Github )*

Mục này chủ yếu gọi các dữ liệu từ DataContext dựa vào brand và category để hiện thị.



Hình 4.15. Thông báo thêm vào giỏ hàng thành công

* Tiếp tục mua sắm: Sẽ đưa về trang chủ.
* Thanh toán: Sẽ đưa vào trang giỏ hàng thanh toán.

*( Xem mã nguồn chi tiết tại Github )*

Phương thức Add kiểm tra nếu người dùng chưa đăng nhập (User.Identity.IsAuthenticated == false), thì trả về JSON với thông báo loggedIn = false. Nếu đã đăng nhập, phương thức tìm sản phẩm theo Id từ \_dataContext.Products. Sau đó, lấy danh sách giỏ hàng từ Session (nếu giỏ hàng chưa tồn tại thì tạo danh sách mới), kiểm tra sản phẩm đã có trong giỏ hàng chưa:

* **Nếu chưa có**: Tạo đối tượng CartItemModel mới và thêm vào giỏ hàng.
* **Nếu đã có**: Tăng số lượng (Quantity) của sản phẩm trong giỏ hàng lên 1.

Cuối cùng, giỏ hàng được lưu lại vào Session thông qua SetJson và trả về JSON với loggedIn = true kèm thông báo "Thêm sản phẩm vào giỏ hàng thành công!".

### Giao diện Form giỏ hàng

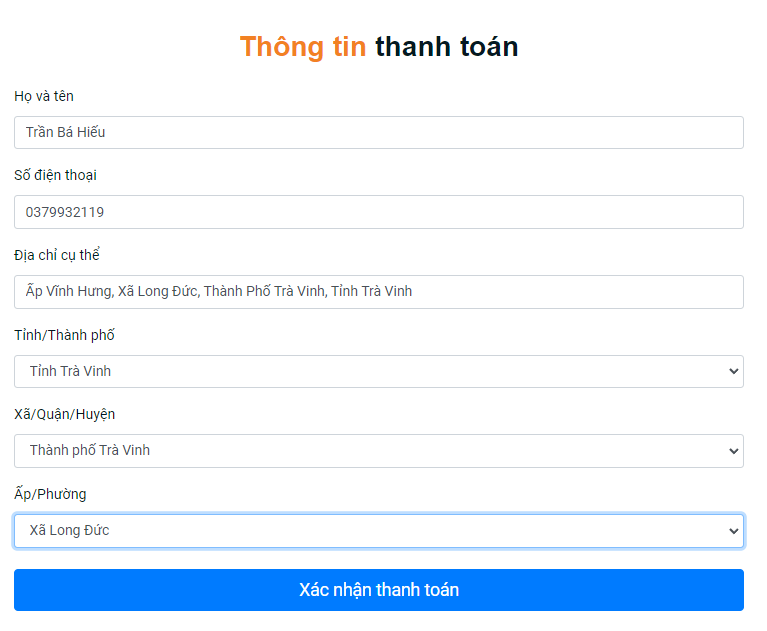
Hình 4.16. Giao diện Form giỏ hàng

Tại trang giỏ hàng sẽ hiển thị các sản phẩm được thêm vào từ người dùng thông qua Cookie và một số nghiệp vụ liên quan để từ Controller để thực hiện việc hiển thị.

*( Xem mã nguồn chi tiết tại Github )*

Khi người dùng hoàn tất việc kiểm tra và ấn vào nút thanh toán sẽ được đưa trên trang Địa chỉ của đơn hàng để thành thành việc hoàn tất đơn hàng.

### Giao diện và chức năng trang hoàn tất đặt hàng



Hình 4.17. Giao diện form hoàn tất đặt hàng

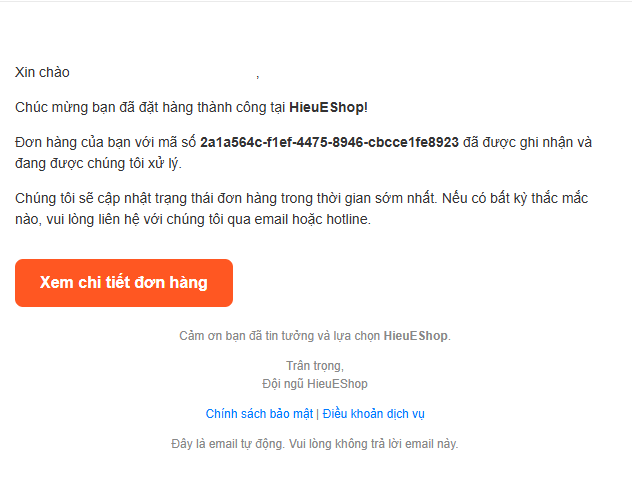
Đây là trang và cũng là bước cuối cùng để thực hiện việc hoàn tất đơn đặt hàng thành công.

Tại trang này có áp dụng API từ bên ngoài cụ thể: [*https://provinces.open-api.vn/*](https://provinces.open-api.vn/)*.*

Api này được gọi vào nhầm đảm bảo việc Select Tỉnh, Xã/Quận/Huyện, Ấp/Phường được chính xác tuyệt đối.

*( Xem mã nguồn chi tiết tại Github )*

Sau khi đặt hàng thành công sẽ nhận được một Email gửi đến Email người dùng xác nhận đơn đặt hàng:



Hình 4.18. Email xác nhận đặt hàng hoàn tất

### Giao diện và chức năng tìm kiếm sản phẩm

Hình 4.19. Giao diện chức năng tìm kiếm sản phẩm

Tại phần này sẽ được thực hiện thông qua ProductController cụ thể như sau:

public async Task<IActionResult> Search(string searchTerm)

{

var products = await \_dataContext.Products

.Where(p => p.Name.Contains(searchTerm) || p.Description.Contains(searchTerm))

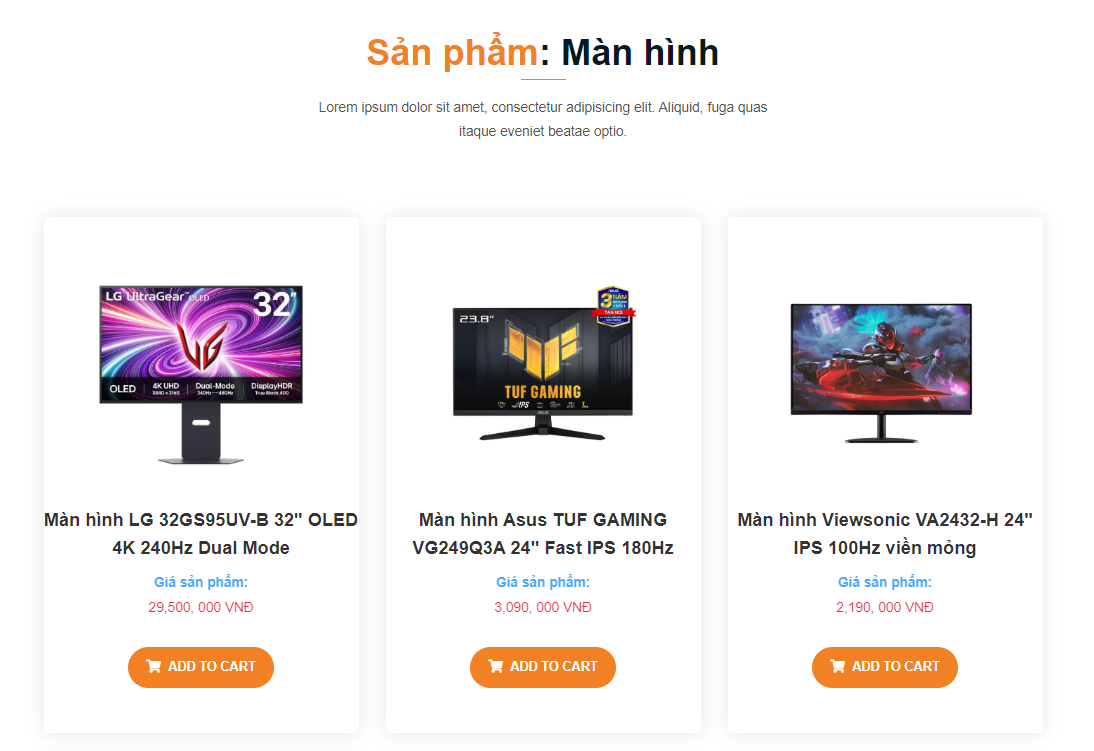
.ToListAsync();

ViewBag.Keyword = searchTerm;

return View(products);

}

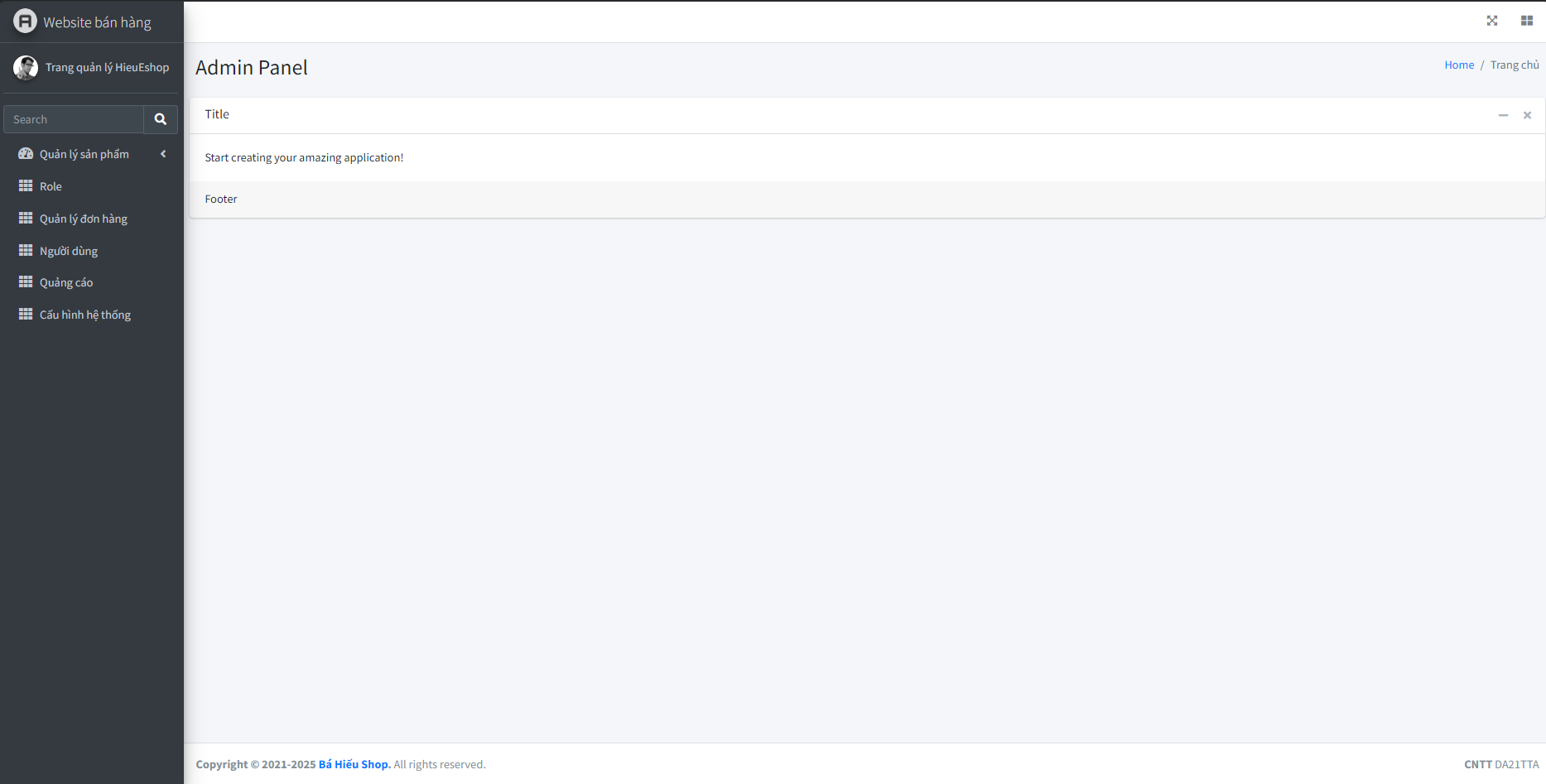
Phương thức **Search** xử lý việc tìm kiếm sản phẩm trong cơ sở dữ liệu dựa trên từ khóa do người dùng nhập vào. Mô tả ngắn gọn:

* **Input**: Nhận từ khóa tìm kiếm (searchTerm) từ người dùng.
* **Xử lý**:
  + Tìm kiếm trong bảng sản phẩm (Products) các mục có tên (Name) hoặc mô tả (Description) chứa từ khóa tìm kiếm (searchTerm) bằng phương thức Contains.
  + Sử dụng ToListAsync để lấy danh sách các sản phẩm phù hợp.
* **Kết quả**:
  + Lưu từ khóa tìm kiếm vào ViewBag.Keyword để hiển thị lại trên giao diện nếu cần.
  + Trả về view cùng danh sách sản phẩm tìm được để hiển thị cho người dùng.

Hình 4.20. Giao diện kết quả sau khi tìm kiếm

## Phần Admin

### Tổng thể trang Admin



Hình 4.21. Giao diện tổng thể trang Admin

Để truy cập vào trang này yêu cầu người dùng phải có Role là Admin và Staff thông qua Identify.

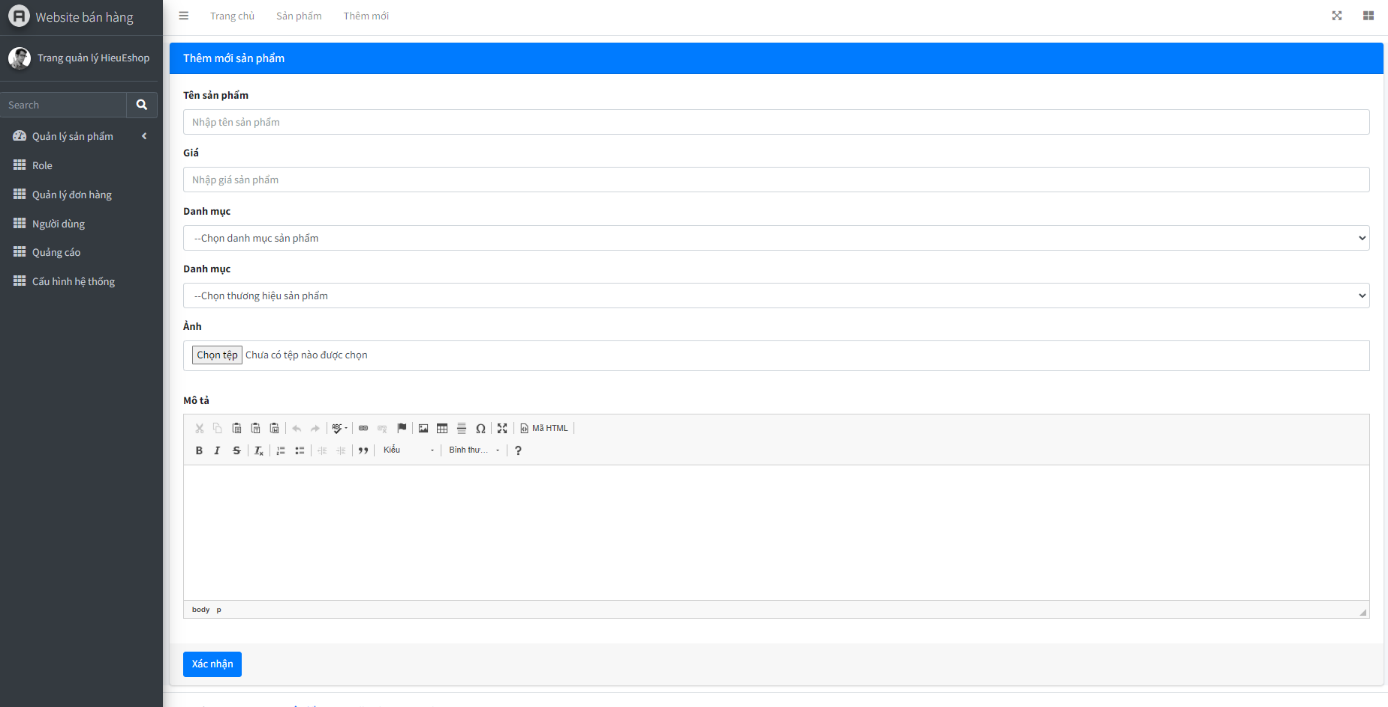
Ví dụ:

[Area("Admin")]

[Authorize(Roles = "Admin")]

### Giao diện trang quản lý sản phẩm

Hình 4.22. Giao diện trang quản lý sản phẩm

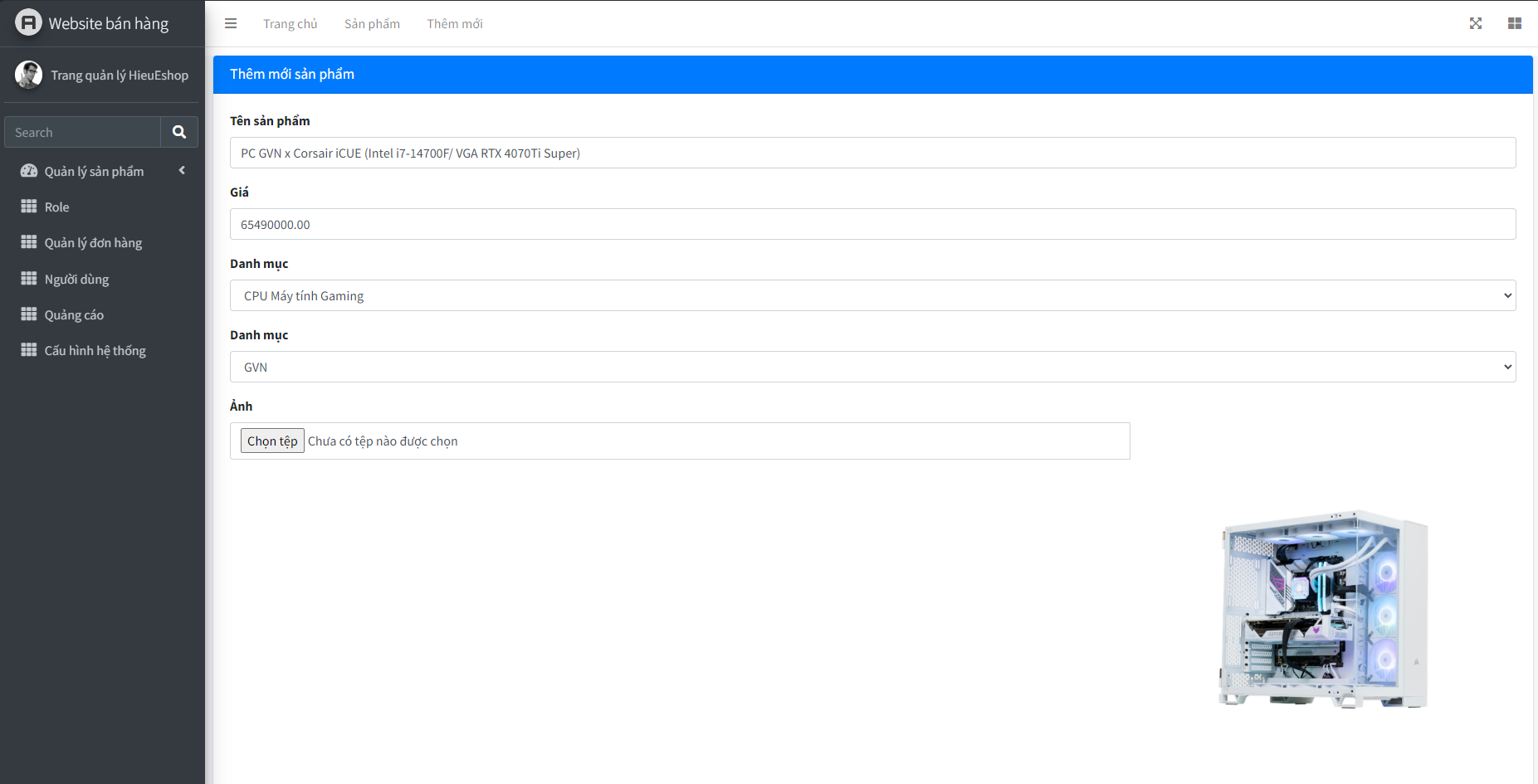
Giao diện trang thêm sản phẩm

Hình 4.23. Giao diện thêm sản phẩm

*( Xem mã nguồn chi tiết tại Github )*

Đầu tiên, phương thức Create sẽ lấy danh sách **Categories** và **Brands** từ dataContext và gán vào ViewBag để hiển thị trên form. Sau đó, kiểm tra dữ liệu từ ProductModel, nếu hợp lệ thì tạo slug cho sản phẩm bằng cách thay dấu cách thành dấu gạch ngang và kiểm tra slug có tồn tại trong cơ sở dữ liệu hay không. Nếu không tồn tại, phương thức sẽ xử lý hình ảnh tải lên, lưu vào thư mục media/products, thêm thông tin sản phẩm vào cơ sở dữ liệu, lưu thay đổi và hiển thị thông báo thành công qua TempData. Cuối cùng, chuyển hướng đến trang danh sách sản phẩm (Index), nếu thất bại sẽ trả về form kèm thông báo lỗi.

Giao diện trang sửa sản phẩm



Hình 4.24. Giao diện trang sửa sản phẩm

*( Xem mã nguồn chi tiết tại Github )*

Đầu tiên, phương thức Edit (GET) sẽ tìm sản phẩm theo Id từ dataContext và gán danh sách **Categories** và **Brands** vào ViewBag để hiển thị trên form chỉnh sửa. Sau đó, phương thức Edit (POST) kiểm tra dữ liệu từ ProductModel, nếu hợp lệ thì:

* **Tạo slug**: Chuyển tên sản phẩm thành định dạng URL-friendly và kiểm tra slug có bị trùng với sản phẩm khác không.
* **Cập nhật ảnh sản phẩm**:
  + Nếu người dùng tải lên ảnh mới: Xóa ảnh cũ khỏi thư mục và lưu ảnh mới vào media/products.
* **Cập nhật thông tin sản phẩm**:
  + Cập nhật các thuộc tính như tên, mô tả, thương hiệu, danh mục, và giá của sản phẩm.
  + Gọi \_dataContext.Update() để lưu thay đổi vào cơ sở dữ liệu.
* Hiển thị thông báo thành công qua TempData và chuyển hướng về trang danh sách sản phẩm (Index).

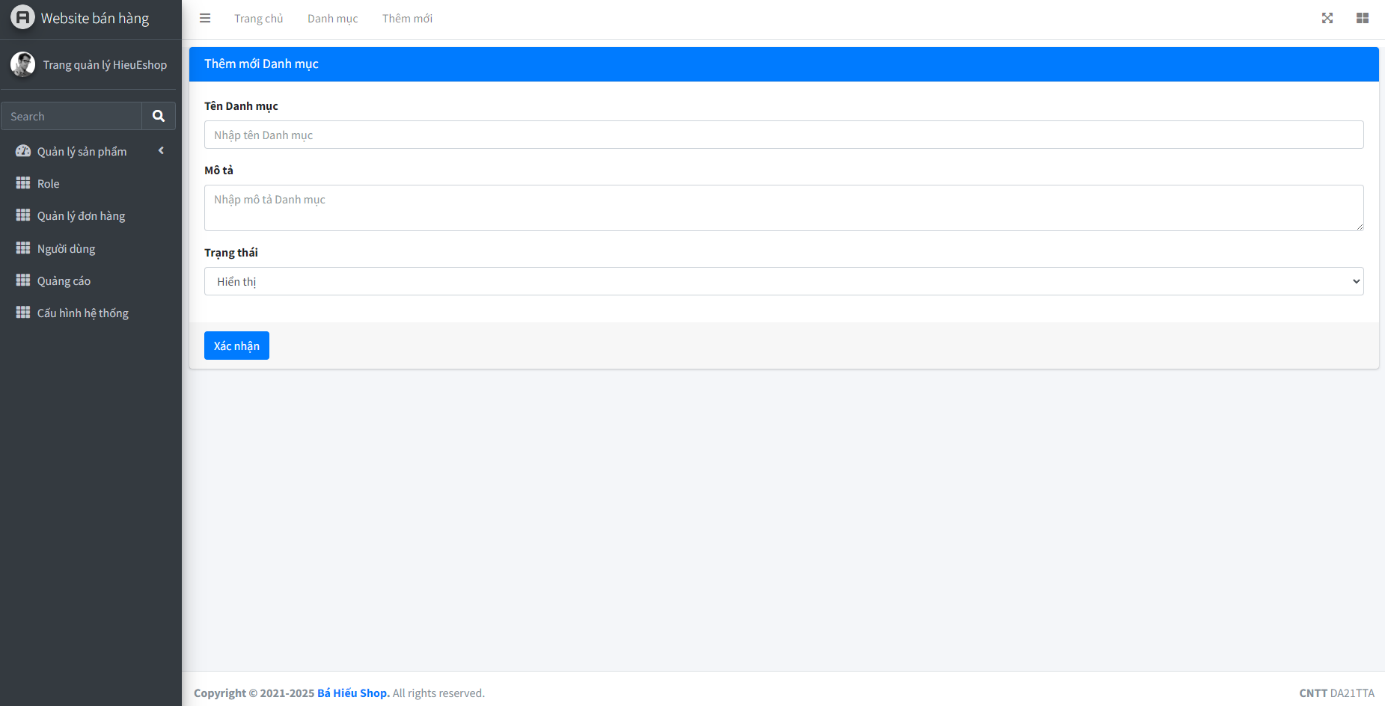
Nếu thất bại, thông báo lỗi sẽ được lưu vào TempData và danh sách lỗi chi tiết trả về dưới dạng BadRequest.

### Giao diện trang quản lý danh mục

Hình 4.25. Giao diện trang quản lý danh mục

*( Xem mã nguồn chi tiết tại Github )*

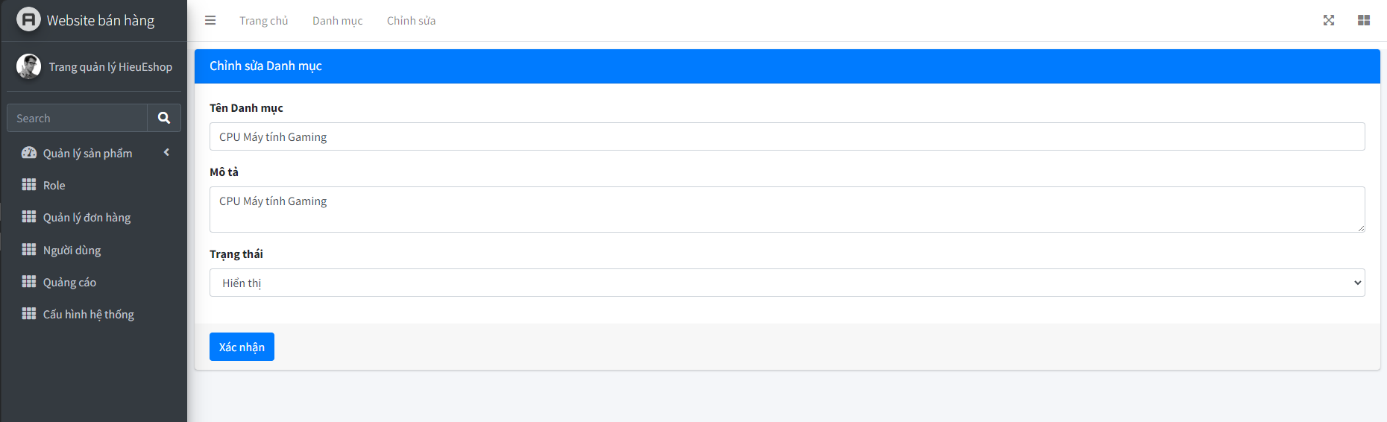
Tương tự phần Product sẽ hiện thị các danh mục được tạo trước đó.

Giao diện trang thêm danh mục

Hình 4.26. Giao diện trang thêm danh mục

*( Xem mã nguồn chi tiết tại Github )*

Tại phần này chủ yếu là xử lý của Controller với phương thức Create. Đầu tiên, phương thức Create sẽ kiểm tra dữ liệu từ CategoryModel, nếu hợp lệ thì tạo slug cho danh mục bằng cách thay dấu cách thành dấu gạch ngang và kiểm tra xem slug có tồn tại trong cơ sở dữ liệu hay không. Nếu slug chưa tồn tại, danh mục mới sẽ được thêm vào cơ sở dữ liệu, lưu thay đổi và hiển thị thông báo thành công qua TempData. Cuối cùng, phương thức chuyển hướng đến trang danh sách danh mục (Index). Nếu thất bại, sẽ trả về form kèm thông báo lỗi và danh sách các lỗi chi tiết.

Giao diện trang sửa danh mục

Hình 4.27. Giao diện trang sửa danh mục

*( Xem mã nguồn chi tiết tại Github )*

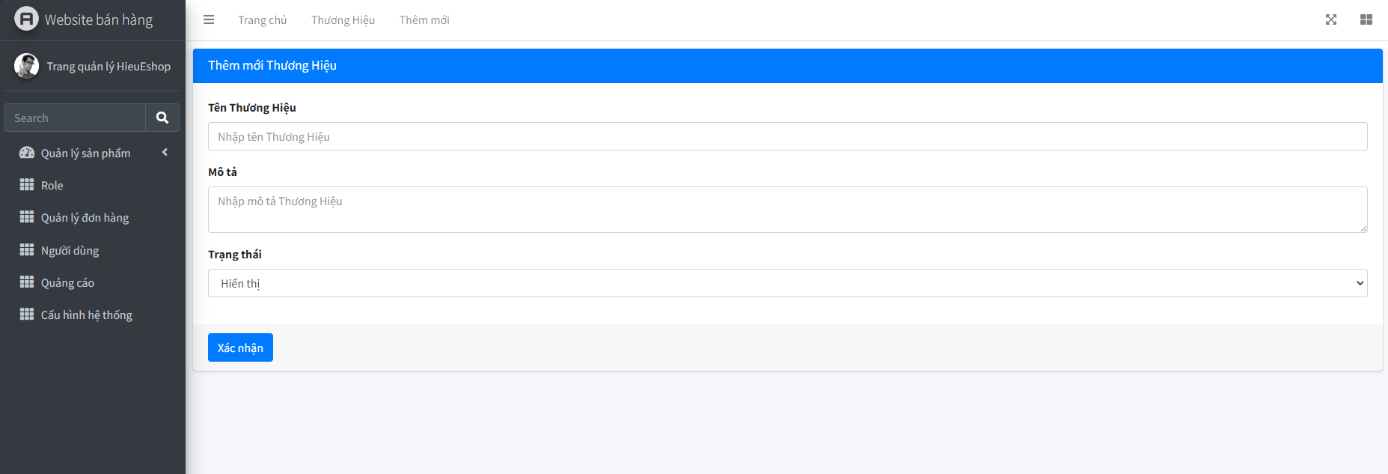
Các xử lý nghiệp vụ trên SourceCode ở phần này sẽ tương tự như phần sửa sản phẩm.

### Giao diện trang quản lý thương hiệu

Hình 4.28. Giao diện trang quản lý thương hiệu

*( Xem mã nguồn chi tiết tại Github )*

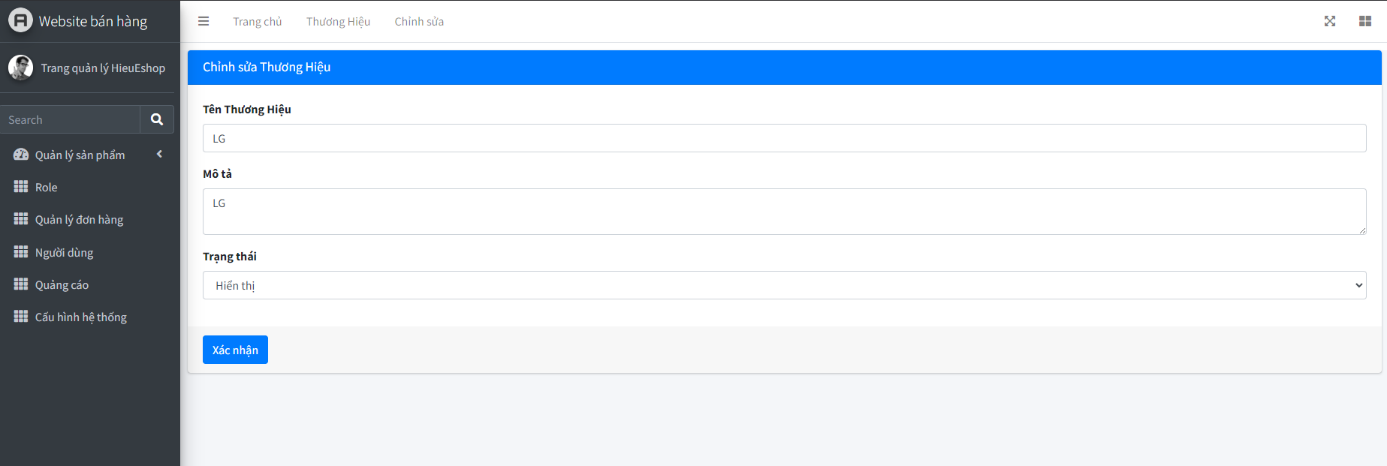
Tương tự hai phần trên sẽ hiện thị các danh mục được tạo trước đó.

Giao diện trang thêm thương hiệu

Hình 4.29. Giao diện trang thêm thương hiệu

*( Xem mã nguồn chi tiết tại Github )*

Các xử lý nghiệp vụ trên SourceCode ở phần này sẽ tương tự như phần danh mục.

Giao diện trang sửa thương hiệu

Hình 4.30. Giao diện trang sửa thương hiệu

*( Xem mã nguồn chi tiết tại Github )*

Các xử lý nghiệp vụ trên SourceCode ở phần này sẽ tương tự như phần danh mục.

### Giao diện trang Role

Hình 4.31. Giao diện trang Role

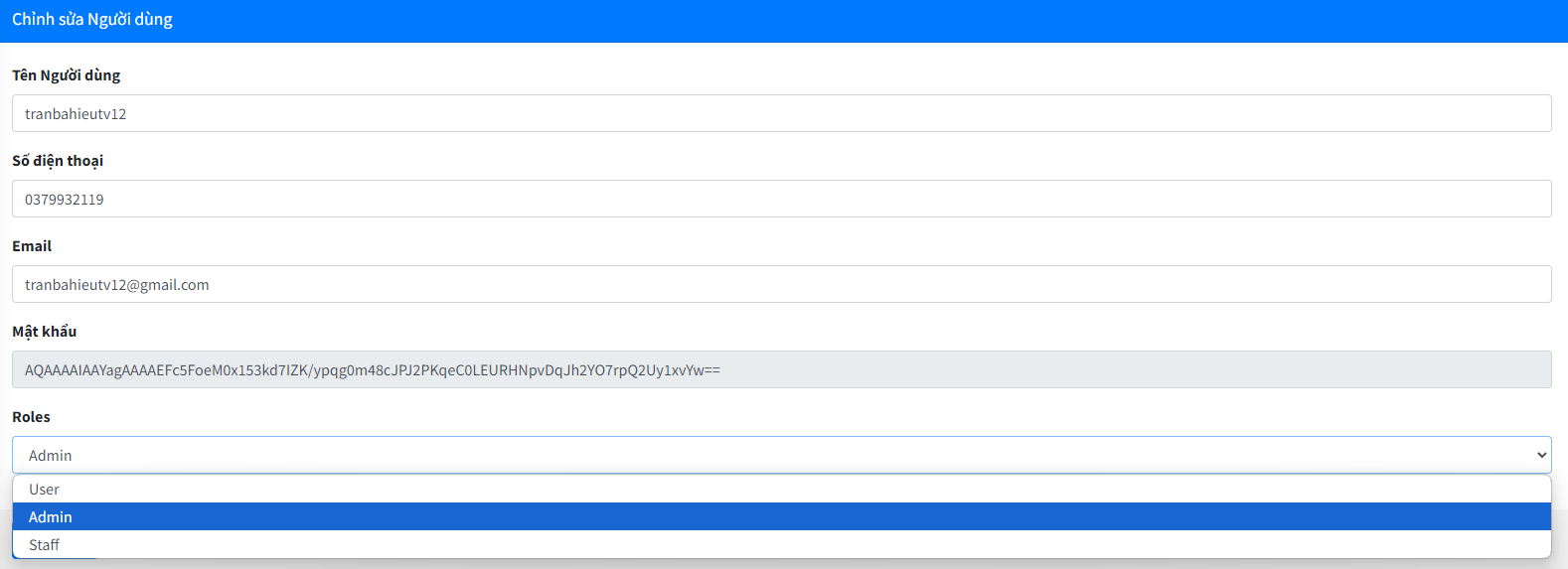
*( Xem mã nguồn chi tiết tại Github )*

Trang này thực hiện chức năng thêm các quyền truy cập cho website.

### Giao diện trang quản lý người dùng

Hình 4.32. Giao diện trang quản lý người dùng

*( Xem mã nguồn chi tiết tại Github )*

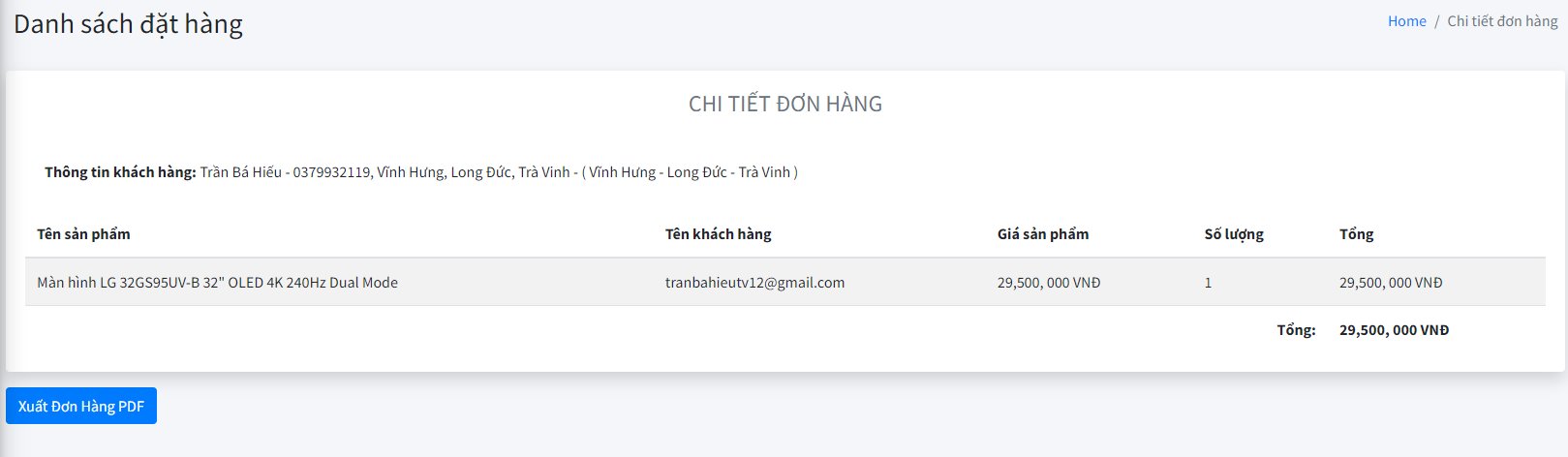
Trang này thực hiện chức năng chính là thiết lập quyền cho người dùng, như chuyển thành nhân viên hoặc chủ cửa hàng cho tài khoản đó.

Hình 4.33. Form chỉnh sửa người dùng

### Giao diện trang quản lý đơn hàng

Hình 4.34. Giao diện trang quản lý đơn hàng

*( Xem mã nguồn chi tiết tại Github )*

Trang này thực hiện chức năng chính xem danh sách đơn đặt hàng và thay đổi trạng thái đơn hàng.

Hình 4.35. Form xem chi tiết đơn hàng

* Chức năng **Xuất Đơn hàng PDF** được kèm theo bằng việc sử dụng Package **PdfSharpCore** Asp .Net core hỗ trợ để tải xuống file Pdf của đơn hàng.



Hình 4.36. File PDF khi xuất hóa đơn

# KẾT LUẬN

## Ưu điểm

1. **Ứng dụng công nghệ tiên tiến:** Framework ASP.NET Core được áp dụng giúp xây dựng một website bán hàng hiện đại, hiệu suất cao, và bảo mật tốt.
2. **Thiết kế linh hoạt:** Hệ thống được thiết kế theo mô hình MVC (Model-View-Controller), dễ dàng mở rộng và bảo trì.
3. **Tối ưu trải nghiệm người dùng:** Giao diện website thân thiện, hỗ trợ tìm kiếm, lọc sản phẩm và thanh toán trực tuyến một cách tiện lợi.
4. **Quản lý hiệu quả:** Hệ thống quản lý sản phẩm, đơn hàng và người dùng được tổ chức rõ ràng, đảm bảo tính toàn vẹn dữ liệu.
5. **Tích hợp bảo mật:** Sử dụng ASP.NET Identity để bảo mật thông tin và phân quyền truy cập.

## Nhược điểm

1. **Phức tạp trong triển khai:** Yêu cầu kiến thức sâu về C# và ASP.NET Core, khiến thời gian học tập và phát triển dài hơn.
2. **Hiệu suất có thể cải thiện:** Khi lưu lượng truy cập lớn, hiệu suất website có thể bị ảnh hưởng nếu không có tối ưu hóa bổ sung như caching.
3. **Phụ thuộc nền tảng:** Website cần môi trường hỗ trợ .NET Core để hoạt động, hạn chế tính linh hoạt trong một số hạ tầng.

## Hướng phát triển

1. **Tối ưu hiệu suất:** Sử dụng các công nghệ caching, load balancing, và lazy loading để cải thiện hiệu năng khi số lượng người dùng tăng.
2. **Tích hợp AI:** Triển khai các tính năng gợi ý sản phẩm dựa trên hành vi người dùng bằng công nghệ trí tuệ nhân tạo.
3. **Mở rộng nền tảng:** Phát triển thêm ứng dụng di động kết nối trực tiếp với hệ thống để tăng cường trải nghiệm người dùng.
4. **Bảo mật nâng cao:** Sử dụng chứng chỉ SSL/TLS và tích hợp công nghệ bảo mật tiên tiến để bảo vệ dữ liệu người dùng.
5. **Đa dạng phương thức thanh toán:** Kết hợp nhiều cổng thanh toán trực tuyến như MoMo, VNPay, hoặc ZaloPay để đáp ứng nhu cầu khách hàng.

# TÀI LIỆU THAM KHẢO

|  |  |
| --- | --- |
| [1] | Tìm hiểu về ngôn ngữ lập trình C#- codegym: https://codegym.vn/blog/c-la-gi-tim-hieu-ve-ngon-ngu-lap-trinh-c/. |
| [2] | Framework Asp.Net là gì ? - Glints: https://glints.com/vn/blog/asp-net-la-gi/. |
| [3] | Tất tần tật về mô hình MVC - Viblo: https://viblo.asia/p/tat-tan-tat-ve-mo-hinh-mvc-Rk74avjAJeO. |
| [4] | Áp dụng IdentifyUser vào MVC - xuanthulab: https://xuanthulab.net/asp-net-razor-su-dung-identity-de-tao-user-xac-thuc-dang-nhap-website.html. |
| [5] | Sweetalert2: https://sweetalert2.github.io/. |