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**

Contents

[KHOA KỸ THUẬT VÀ CÔNG NGHỆ 1](#_Toc186632083)

[BỘ MÔN CÔNG NGHỆ THÔNG TIN 1](#_Toc186632084)

[BÁO CÁO ĐỒ ÁN CHUYÊN NGÀNH 1](#_Toc186632085)

[KHOA KỸ THUẬT VÀ CÔNG NGHỆ 2](#_Toc186632086)

[BỘ MÔN CÔNG NGHỆ THÔNG TIN 2](#_Toc186632087)

[BÁO CÁO ĐỒ ÁN CHUYÊN NGÀNH 2](#_Toc186632088)

[TÓM TẮT NIÊN LUẬN vi](#_Toc186632089)

[CHƯƠNG 1: TỔNG QUAN 1](#_Toc186632090)

[1.1. Tổng quan về vấn đề 1](#_Toc186632091)

[1.2. Hướng giải quyết 2](#_Toc186632092)

[CHƯƠNG 2: NGHIÊN CỨU LÝ THUYẾT 4](#_Toc186632093)

[2.1. Ngôn ngữ lập trình C# ( C Sharp ) 4](#_Toc186632094)

[2.2. ASP.NET Framework và sự khác nhau giữa ASP.NET Core và ASP.NET 5](#_Toc186632095)

[2.3. Mô hình MVC (Model-View-Controller) 7](#_Toc186632096)

[2.3.1. Khái quát về cấu trúc mô hình MVC 8](#_Toc186632097)

[2.3.2. Cách hoạt động của mô hình MVC 8](#_Toc186632098)

[2.3.3. Cấu trúc mô hình MVC 9](#_Toc186632099)

[2.3.4. So sánh mô hình MVC và các mô hình khác 10](#_Toc186632100)

[2.4. Một số phương thức, khái niệm, công nghệ/công cụ... khác được sử dụng 11](#_Toc186632101)

[CHƯƠNG 3: THỰC NGHIỆM 12](#_Toc186632102)

[3.1. Thiết kế mô hình ERD (Entity Relationship Diagram) 12](#_Toc186632103)

[3.2. Thiết kế mô hình UseCase 13](#_Toc186632104)

[3.2.1. UseCase khách hàng 13](#_Toc186632105)

[3.2.2. UseCase quản trị viên 16](#_Toc186632106)

[3.3. Thiết kế CSDL 18](#_Toc186632107)

[3.3.1. Thực thể sản phẩm (Products) 18](#_Toc186632108)

[3.3.2. Thực thể người dùng (Users) 19](#_Toc186632109)

[3.3.3. Thực thể danh mục (Categories) 20](#_Toc186632110)

[3.3.4. Thực thể thương hiệu (Brands) 20](#_Toc186632111)

[3.3.5. Thực thể đặt hàng (Orders) 21](#_Toc186632112)

[3.3.6. Thực thể chi tiết đơn hàng (OrderDetails) 22](#_Toc186632113)

[3.3.7. Thực thể đánh giá (Ratings) 23](#_Toc186632114)

[3.3.8. Thực thể địa chỉ đơn hàng (BillingAddresses) 24](#_Toc186632115)

[3.4. 25](#_Toc186632116)

[CHƯƠNG 4: KẾT LUẬN 26](#_Toc186632117)

[TÀI LIỆU THAM KHẢO 27](#_Toc186632118)

**DANH MỤC HÌNH ẢNH**

# 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:** Cơ sở lý thuyết.
* **Chương 2:** Thực nghiệm.
* **Chương 3:** 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.

**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.

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:**

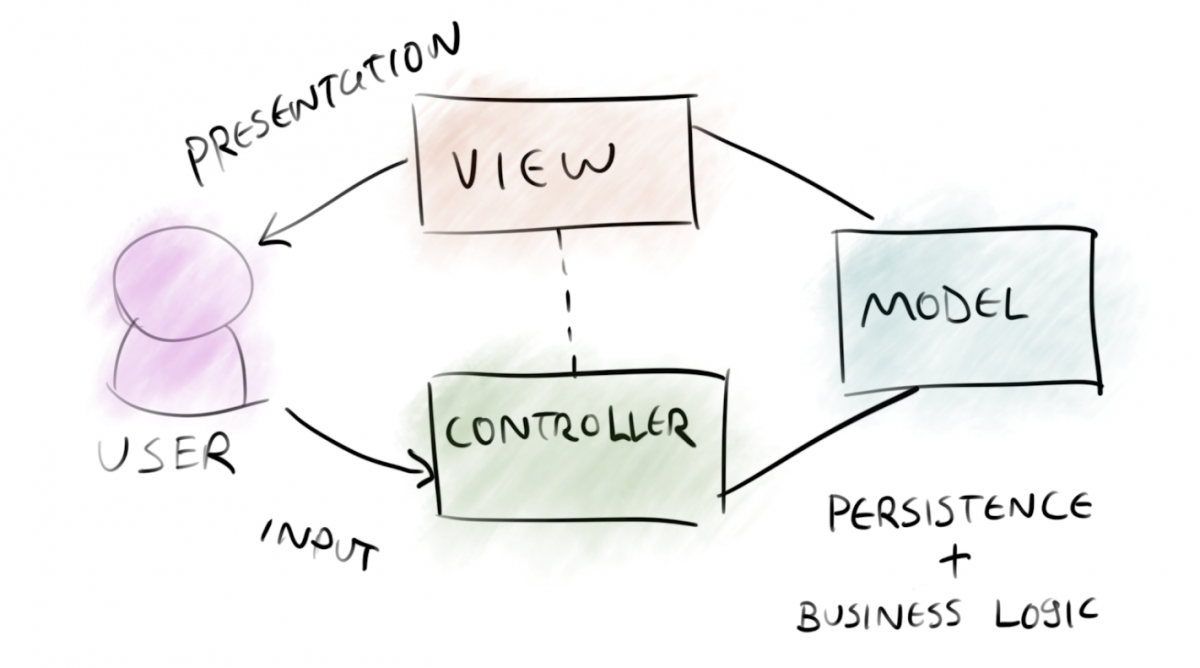
+ ASP.NET: Các tính năng như Web API và SignalR có thể được sử dụng nhưng không được tối ưu hóa cho các ứng dụng hiện đại.

+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.

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).

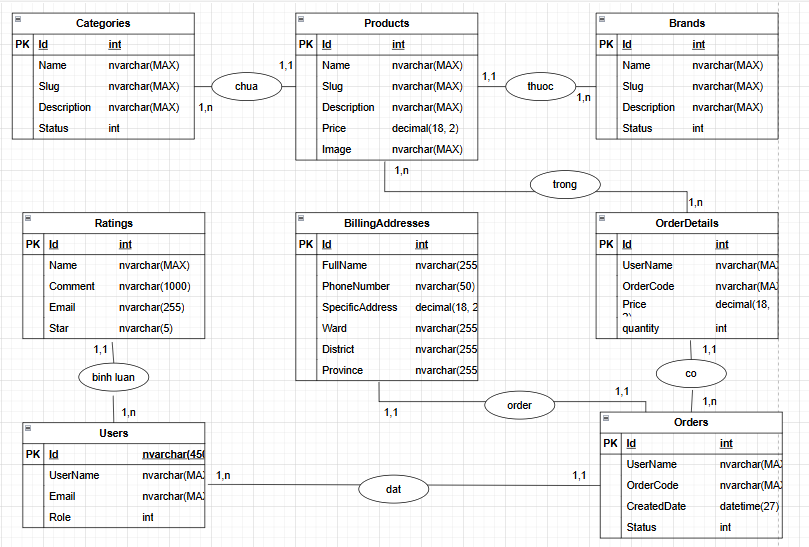
+ **IdentityUser**: một lớp được sử dụng trong **ASP.NET Core Identity**, một framework của Microsoft giúp quản lý xác thực và phân quyền cho các ứng dụng web. Đây là lớp mặc định đại diện cho một tài khoản người dùng trong hệ thống.

+ **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.

+ **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.

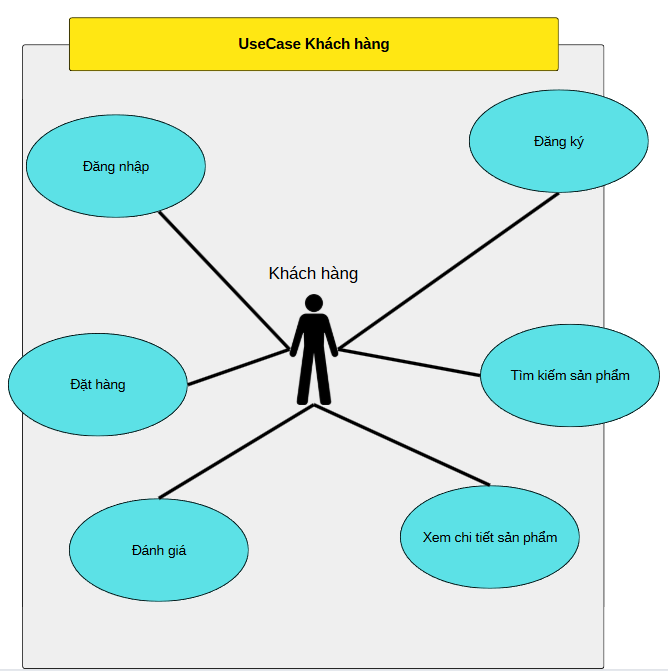
## Thiết kế mô hình UseCase

Use Case là một kỹ thuật thường được sử dụng trong hệ thống và kỹ thuật phần mềm để các đối tượng người dùng có thể nắm rõ được các yêu cầu chức năng của hệ thống. Ngoài ra, Use Case còn là một kỹ thuật có khả năng mô tả sự tương tác giữa hệ thống và người dùng trong cùng một môi trường và vì cùng một mục đích cụ thể.

### UseCase khách hàng

**Actor:**

**Khách hàng**: Là người sử dụng hệ thống để thực hiện các hoạt động mua sắm trên website.



Hình 3.2. UseCase khách hàng

**Các Use Case chính:**

1. **Đăng ký:**

* **Mô tả**: Khách hàng tạo tài khoản mới để sử dụng các tính năng của hệ thống.
* **Mục tiêu**: Cho phép khách hàng trở thành người dùng đã đăng ký.
* **Mối quan hệ**: Khách hàng có thể thực hiện hành động này trước khi đăng nhập.

1. **Đăng nhập (Login):**

* **Mô tả**: Khách hàng nhập thông tin tài khoản để truy cập hệ thống.
* **Mục tiêu**: Xác thực danh tính khách hàng để cho phép sử dụng các tính năng khác.
* **Mối quan hệ**: Khách hàng cần đăng nhập để đặt hàng hoặc đánh giá sản phẩm.

1. **Tìm kiếm sản phẩm:**

* **Mô tả**: Khách hàng tìm kiếm sản phẩm theo từ khóa, danh mục, hoặc các tiêu chí khác.
* **Mục tiêu**: Giúp khách hàng tìm sản phẩm phù hợp một cách nhanh chóng.
* **Mối quan hệ**: Thực hiện trước khi xem chi tiết sản phẩm.

1. **Xem chi tiết sản phẩm:**

* **Mô tả**: Khách hàng xem thông tin chi tiết về sản phẩm, bao gồm mô tả, giá cả, hình ảnh, và đánh giá.
* **Mục tiêu**: Cung cấp đủ thông tin để khách hàng quyết định mua hàng.
* **Mối quan hệ**: Có thể dẫn đến hành động đặt hàng hoặc đánh giá sản phẩm.

1. **Đặt hàng:**

* **Mô tả**: Khách hàng thêm sản phẩm vào giỏ hàng và tiến hành thanh toán để hoàn thành đơn hàng.
* **Mục tiêu**: Cho phép khách hàng mua sản phẩm.
* **Mối quan hệ**: Khách hàng cần đăng nhập trước khi đặt hàng.

1. **Đánh giá:**

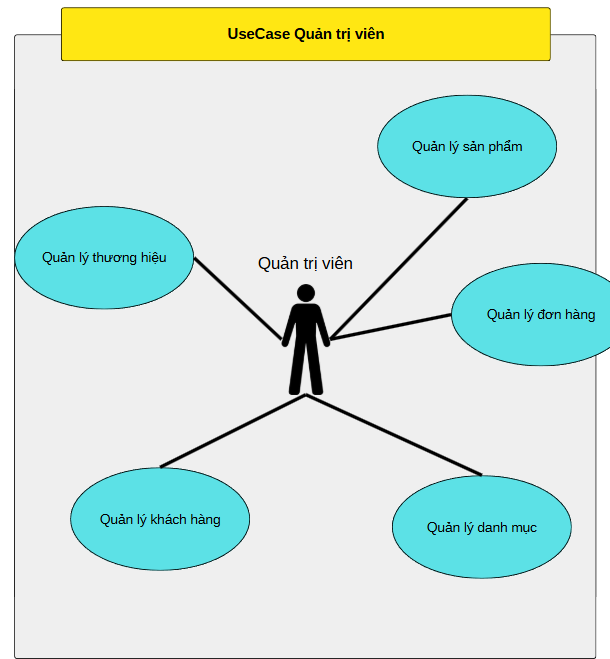
* **Mô tả**: Khách hàng để lại đánh giá về sản phẩm đã mua, bao gồm nhận xét và số sao.
* **Mục tiêu**: Giúp khách hàng khác có thêm thông tin về sản phẩm và cải thiện chất lượng dịch vụ.
* **Mối quan hệ**: Khách hàng cần đăng nhập và mua sản phẩm trước khi đánh giá.

**Mối quan hệ giữa các Use Case:**

* **Đăng ký** là bước đầu tiên mà khách hàng thực hiện nếu chưa có tài khoản.
* **Đăng nhập** là bước cần thiết trước khi khách hàng thực hiện các hành động khác như đặt hàng hoặc đánh giá sản phẩm.
* **Tìm kiếm sản phẩm** và **Xem chi tiết sản phẩm** là các hành động thường xảy ra trước khi khách hàng đặt hàng.
* **Đánh giá** có thể thực hiện sau khi khách hàng đã mua và sử dụng sản phẩm.

### UseCase quản trị viên

**Actor:**

**Quản trị viên**: Là người quản lý hệ thống, thực hiện các tác vụ liên quan đến quản lý sản phẩm, khách hàng, đơn hàng, thương hiệu, và danh mục.

Hình 3.3. UseCase quản trị viên

**Các Use Case chính:**

1. **Quản lý sản phẩm:**

* **Mô tả**: Thêm mới, sửa đổi hoặc xóa sản phẩm trong hệ thống.
* **Mục tiêu**: Cập nhật thông tin sản phẩm để đảm bảo dữ liệu luôn chính xác.
* **Liên quan đến bảng dữ liệu**: Products.

1. **Quản lý đơn hàng:**

* **Mô tả**: Xem, xử lý hoặc hủy đơn hàng của khách hàng.
* **Mục tiêu**: Đảm bảo đơn hàng được xử lý đúng hạn và hiệu quả.
* **Liên quan đến bảng dữ liệu**: Orders, OrderDetails.

1. **Quản lý khách hàng:**

* **Mô tả**: Quản lý thông tin tài khoản khách hàng, bao gồm thêm, sửa, hoặc xóa.
* **Mục tiêu**: Đảm bảo dữ liệu khách hàng đầy đủ và chính xác.
* **Liên quan đến bảng dữ liệu**: Users.

1. **Quản lý danh mục:**

* **Mô tả**: Thêm mới, chỉnh sửa hoặc xóa danh mục sản phẩm.
* **Mục tiêu**: Sắp xếp các sản phẩm vào đúng danh mục để thuận tiện cho khách hàng.
* **Liên quan đến bảng dữ liệu**: Categories.

1. **Quản lý thương hiệu:**

* Mô tả: Thêm mới, sửa đổi hoặc xóa các thương hiệu sản phẩm.
* Mục tiêu: Đảm bảo các sản phẩm được liên kết với thương hiệu chính xác.
* Liên quan đến bảng dữ liệu: Brands.

1. **Mối quan hệ giữa các Use Case:**

* **Quản lý danh mục** và **Quản lý thương hiệu** có thể hỗ trợ cho **Quản lý sản phẩm**, vì mỗi sản phẩm cần liên kết với danh mục và thương hiệu tương ứng.
* **Quản lý đơn hàng** yêu cầu thông tin từ **Quản lý khách hàng** để liên hệ xử lý.
* **Quản lý sản phẩm** là trung tâm trong việc tổ chức thông tin hàng hóa.

## 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,

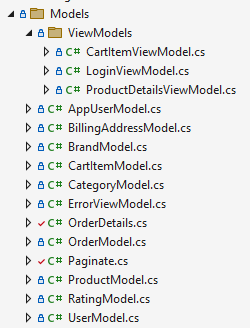
[Province] [nvarchar](255) NOT NULL

)

## 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 3.4. 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 3.5. 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).

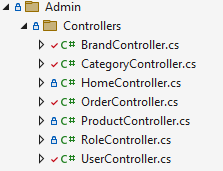
AccountController:

* **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 3.6. 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:

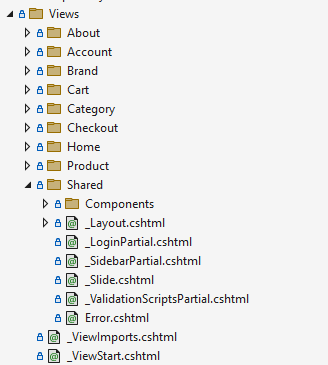
* **Index(int pg = 1)**:
  + Hiển thị danh sách thương hiệu với phân trang, mỗi trang hiển thị tối đa 10 thương hiệu.
  + Truyền dữ liệu phân trang (Paginate) đến View.
* **Create() [GET]**: Hiển thị form để thêm thương hiệu mới.
* **Create(BrandModel brand) [POST]**:
  + Thêm thương hiệu mới vào cơ 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:

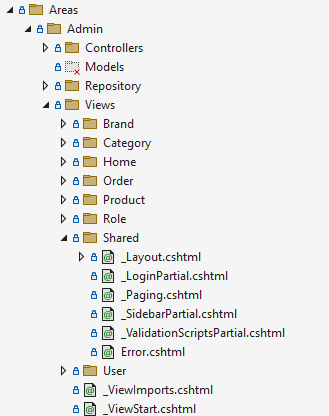
### Cấu trúc Views của trang chính



Hình 3.7. 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 3.8. 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.

### Thiết kế giao diện

giao diện trang chính

giao diện trang Admin

# KẾT QUẢ NGHIÊN CỨU

# KẾT LUẬN

# TÀI LIỆU THAM KHẢO