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

Trong suốt quá trình thực hiện đồ án “ShoesShopWeb – Website bán giày trực tuyến”, em nhận được sự hướng dẫn tận tình của quý thầy cô, sự hỗ trợ chuyên môn của bạn bè và sự động viên to lớn từ gia đình. Em xin bày tỏ lòng biết ơn sâu sắc đến quý thầy cô đã trang bị cho em kiến thức nền tảng về ASP.NET Core, Entity Framework Core, cơ sở dữ liệu PostgreSQL và phương pháp luận nghiên cứu khoa học. Những kiến thức và định hướng phương pháp này giúp em xây dựng được khung tiếp cận rõ ràng, xác định mục tiêu phù hợp và triển khai hệ thống theo trình tự khoa học.

Em trân trọng cảm ơn các thầy cô phụ trách môn học và hướng dẫn đồ án đã dành thời gian góp ý, gợi mở các góc nhìn về kiến trúc nhiều lớp, tổ chức dữ liệu và phương thức kiểm thử. Các buổi trao đổi học thuật, phản biện chuyên môn đã giúp em nhận diện sớm hạn chế, điều chỉnh thiết kế, tối ưu hóa luồng xử lý và nâng cao chất lượng mã nguồn. Sự tỉ mỉ trong từng nhận xét và sự tận tâm trong quá trình đồng hành là động lực để em kiên trì hoàn thiện sản phẩm.

Em cũng cảm ơn các bạn đồng môn đã tích cực hỗ trợ trong quá trình phát triển và kiểm thử, từ việc rà soát yêu cầu đến thử nghiệm các chức năng quản lý sản phẩm, giỏ hàng và đơn hàng. Những ý kiến phản hồi thực tiễn từ trải nghiệm người dùng đã giúp em cải thiện giao diện Razor Pages, tối ưu tương tác AJAX và điều chỉnh quy trình đặt hàng theo hướng thân thiện và hiệu quả hơn. Sự hợp tác và tinh thần chia sẻ đã tạo nên môi trường học tập tích cực, thúc đẩy tiến độ và chất lượng của đồ án.

Đặc biệt, em xin gửi lời cảm ơn đến gia đình vì luôn là chỗ dựa tinh thần vững chắc, tạo điều kiện về thời gian và môi trường học tập. Sự động viên kịp thời đã giúp em giữ vững tinh thần trong giai đoạn căng thẳng, vượt qua thách thức về kỹ thuật và quản lý thời gian. Nhờ sự đồng hành của gia đình, em có thể tập trung nghiên cứu, hiện thực hóa các hạng mục theo kế hoạch và hoàn thành đồ án đúng tiến độ.

Cuối cùng, em cảm ơn các thầy cô, bạn bè và những người dùng thử đã dành sự quan tâm và hỗ trợ đối với đề tài. Các góp ý giá trị về bảo mật (hashing mật khẩu, CSRF), về mô hình dữ liệu (biến thể sản phẩm theo màu và kích cỡ) và về cách triển khai (Docker Compose cho PostgreSQL) đã giúp sản phẩm đạt độ ổn định tốt hơn trong môi trường phát triển. Từ những trải nghiệm này, em rút ra nhiều bài học quý báu về tư duy hệ thống, quy trình làm việc và sự cẩn trọng trong kiểm thử, làm nền tảng cho các dự án sau.

MỤC LỤC

DANH MỤC HÌNH ẢNH

TÓM TẮT

Đồ án hiện thực một hệ thống thương mại điện tử chuyên bán giày dép với đầy đủ nghiệp vụ cốt lõi: quản lý danh mục, sản phẩm và biến thể (màu sắc, kích cỡ), giỏ hàng và đặt hàng trực tuyến, cùng cơ chế phân quyền theo vai trò (Admin, Staff, Customer). Bối cảnh lựa chọn đề tài xuất phát từ nhu cầu thực tế của cửa hàng trực tuyến về một nền tảng ổn định, dễ mở rộng và bảo trì, đáp ứng đặc thù sản phẩm thời trang có nhiều biến thể và yêu cầu trải nghiệm nhất quán. Mục tiêu là xây dựng một giải pháp có cấu trúc rõ ràng, đảm bảo tính nhất quán dữ liệu, độ tin cậy trong xử lý nghiệp vụ và trải nghiệm người dùng hợp lý trong môi trường phát triển.

Hệ thống được phát triển bằng ASP.NET Core 9.0, theo kiến trúc nhiều lớp N-Layer (Entity, DAL, BLL, Presentation), sử dụng Razor Pages cho tầng giao diện và Entity Framework Core 9.0 cho truy cập dữ liệu. Cơ sở dữ liệu PostgreSQL 16 được khởi chạy nhất quán thông qua Docker Compose nhằm đơn giản hóa việc thiết lập môi trường, quản lý cấu hình kết nối và đảm bảo khả năng tái lập. Các yêu cầu bảo mật cơ bản được hiện thực bằng cơ chế hashing mật khẩu với BCrypt và bảo vệ CSRF tích hợp. Thiết kế mô hình dữ liệu phản ánh đặc thù sản phẩm giày, với các quan hệ Category–Product–ProductVariant và Order–OrderItem, hỗ trợ thao tác CRUD, tính toàn vẹn dữ liệu và quy trình đặt hàng.

Phương pháp tiếp cận bao gồm khảo cứu tài liệu chính thống, thiết kế kiến trúc và mô hình dữ liệu, hiện thực hóa dịch vụ nghiệp vụ trong BLL, cùng kiểm thử chức năng theo luồng người dùng đại diện. Kết quả cho thấy hệ thống hoạt động ổn định trong môi trường phát triển: các chức năng CRUD vận hành đúng, giỏ hàng có tương tác AJAX, quy trình đặt hàng và quản lý đơn hàng đáp ứng yêu cầu. Báo cáo tổng kết cơ sở lý thuyết, quy trình hiện thực, mô hình cơ sở dữ liệu, kiến trúc triển khai và đánh giá kết quả; đồng thời đề xuất hướng phát triển như tích hợp cổng thanh toán, mở rộng thống kê và triển khai CI/CD.

Tổng kết lại, đồ án đã chứng minh tính khả thi của việc xây dựng một nền tảng bán giày trực tuyến theo kiến trúc nhiều lớp với công nghệ .NET hiện đại, đạt được các mục tiêu về chức năng, bảo mật cơ bản và trải nghiệm người dùng trong môi trường phát triển. Sản phẩm tạo tiền đề cho việc mở rộng theo hướng tích hợp thanh toán, nâng cấp báo cáo thống kê và tự động hóa quy trình triển khai, góp phần nâng cao chất lượng và hiệu quả vận hành của hệ thống trong các giai đoạn tiếp theo.

MỞ ĐẦU

# Lý do chọn đề tài

Trong bối cảnh thương mại điện tử phát triển nhanh, các cửa hàng trực tuyến cần những hệ thống bán hàng chuyên biệt, ổn định và dễ mở rộng. Nhóm sản phẩm thời trang, đặc biệt là giày dép, có đặc thù quản lý biến thể theo màu sắc và kích cỡ, đòi hỏi thiết kế dữ liệu và quy trình nghiệp vụ rõ ràng. Đề tài ShoesShopWeb được lựa chọn để hiện thực một website bán giày trực tuyến có đầy đủ quy trình từ quản lý danh mục, sản phẩm, biến thể đến giỏ hàng và xử lý đơn hàng. Đồng thời, đề tài tạo cơ hội áp dụng công nghệ .NET hiện đại, chuẩn hóa hạ tầng cơ sở dữ liệu và rèn luyện tư duy kiến trúc phần mềm theo mô hình nhiều lớp.

Việc triển khai hệ thống trên nền ASP.NET Core giúp tận dụng hệ sinh thái phong phú, khả năng mở rộng, bảo mật tích hợp và cộng đồng hỗ trợ. Lựa chọn PostgreSQL và Docker Compose mang lại môi trường phát triển nhất quán, dễ tái lập và thuận lợi cho việc chuyển giao. Nhờ đó, đề tài vừa đáp ứng yêu cầu học thuật, vừa có tính ứng dụng thực tế.

# Mục đích nghiên cứu

Mục tiêu tổng quát là thiết kế và hiện thực một website bán giày trực tuyến theo kiến trúc nhiều lớp, bảo đảm tính mô-đun, khả năng mở rộng và dễ bảo trì. Hệ thống cần đáp ứng đầy đủ nghiệp vụ cốt lõi, bao gồm quản lý danh mục, sản phẩm và biến thể (màu sắc, kích cỡ), giỏ hàng và đặt hàng, cùng cơ chế phân quyền theo vai trò để đảm bảo an toàn và kiểm soát truy cập.

Mục tiêu cụ thể là vận dụng Entity Framework Core để mô hình hóa dữ liệu và quản lý migrations, thiết lập kết nối ổn định với PostgreSQL qua Docker Compose, và xây dựng giao diện Razor Pages thân thiện, responsive. Bên cạnh đó, dự án đặt mục tiêu đảm bảo bảo mật cơ bản (hashing mật khẩu, CSRF), cung cấp trải nghiệm người dùng hợp lý và tạo nền tảng sẵn sàng cho việc tích hợp các tính năng nâng cao trong tương lai.

# Đối tượng nghiên cứu

Đối tượng nghiên cứu là hệ thống thương mại điện tử chuyên bán giày, với trọng tâm vào các nghiệp vụ quản lý danh mục, sản phẩm, biến thể, giỏ hàng, đặt hàng và phân quyền người dùng. Hệ thống hướng tới phục vụ ba nhóm vai trò chính: Khách hàng, Nhân viên (Staff) và Quản trị viên (Admin), mỗi vai trò có phạm vi chức năng và mức độ truy cập khác nhau.

Phạm vi nghiên cứu tập trung vào nền tảng ASP.NET Core với Razor Pages cho giao diện, sử dụng Entity Framework Core cho truy cập dữ liệu và PostgreSQL làm cơ sở dữ liệu, triển khai qua Docker Compose để đảm bảo môi trường nhất quán. Các nội dung như tích hợp cổng thanh toán bên thứ ba, tối ưu SEO nâng cao, phân tích dữ liệu chuyên sâu và triển khai CI/CD toàn diện được ghi nhận là hướng phát triển, chưa nằm trong phạm vi hiện thực của đồ án này.

# Phương pháp nghiên cứu

Phương pháp nghiên cứu bao gồm tiếp cận từ nền tảng lý thuyết và thực nghiệm. Trước hết, em khảo cứu các tài liệu chính thức của ASP.NET Core, Entity Framework Core, PostgreSQL và Docker Compose để nắm vững đặc điểm, mô hình vận hành và thực hành tốt. Trên cơ sở đó, em xây dựng kiến trúc hệ thống theo mô hình N-Layer, thiết kế mô hình dữ liệu phù hợp với đặc thù sản phẩm giày (biến thể theo màu, kích cỡ) và các quan hệ giữa các thực thể như danh mục, sản phẩm, đơn hàng, người dùng.

Quá trình hiện thực được triển khai theo hướng gia tăng, chia nhỏ chức năng, tích hợp và kiểm thử theo luồng người dùng đại diện (Customer, Staff, Admin). Các báo cáo tiến độ tuần giúp ghi nhận trạng thái, rủi ro và điều chỉnh kế hoạch đúng thời điểm. Bên cạnh kiểm thử chức năng, em cũng xem xét các yếu tố phi chức năng như bảo mật cơ bản (hashing mật khẩu, CSRF), tính ổn định và trải nghiệm người dùng, nhằm đảm bảo hệ thống đạt yêu cầu ở mức độ triển khai thực tế.

# Ý nghĩa của đề tài

Ý nghĩa học thuật của đề tài thể hiện ở việc vận dụng có hệ thống các kiến thức về kiến trúc phần mềm nhiều lớp, mô hình hóa dữ liệu với EF Core và thiết lập môi trường triển khai bằng Docker Compose. Việc hiện thực hóa một bài toán thương mại điện tử với đặc thù sản phẩm có nhiều biến thể giúp củng cố năng lực phân tích yêu cầu, thiết kế lược đồ cơ sở dữ liệu và tổ chức tầng nghiệp vụ theo nguyên tắc tách biệt trách nhiệm. Quá trình nghiên cứu và kiểm thử theo luồng người dùng đại diện cũng góp phần hoàn thiện kỹ năng đánh giá phi chức năng như bảo mật, tính ổn định và trải nghiệm người dùng.

Ý nghĩa thực tiễn nằm ở khả năng xây dựng một nền tảng bán giày trực tuyến có cấu trúc rõ ràng, dễ mở rộng và dễ bảo trì, phù hợp với nhu cầu cửa hàng thương mại điện tử hiện nay. Hệ thống tạo tiền đề cho việc tích hợp các dịch vụ thanh toán, mở rộng báo cáo thống kê và tự động hóa quy trình triển khai, qua đó nâng cao hiệu quả vận hành và giảm chi phí bảo trì. Đề tài đồng thời gợi mở một quy trình làm việc có thể tái sử dụng cho các dự án web khác trong cùng lĩnh vực.

CHƯƠNG 1: TỔNG QUAN

Chương này trình bày mục tiêu và bối cảnh của đề tài, kiến trúc tổng thể của hệ thống, cùng một số nội dung nền tảng gồm yêu cầu và phạm vi, công nghệ và môi trường, dữ liệu và giả định nghiệp vụ, rủi ro và biện pháp giảm thiểu. Mục tiêu của chương là định vị vấn đề, làm rõ phương hướng hiện thực và chuẩn hóa thuật ngữ sử dụng trong các phần sau.

# Mục tiêu và bối cảnh

Mục tiêu của đề tài là xây dựng một hệ thống thương mại điện tử chuyên bán giày, đáp ứng đầy đủ nghiệp vụ cốt lõi (quản lý danh mục, sản phẩm và biến thể; giỏ hàng; đặt hàng) và phân quyền người dùng theo vai trò (Admin, Staff, Customer). Bối cảnh lựa chọn đề tài xuất phát từ nhu cầu thực tế của cửa hàng trực tuyến về một nền tảng ổn định, dễ mở rộng và dễ bảo trì; đồng thời sản phẩm thời trang có nhiều biến thể theo màu sắc và kích cỡ đặt ra yêu cầu thiết kế dữ liệu và quy trình nghiệp vụ rõ ràng.

# Kiến trúc tổng thể của hệ thống

Hệ thống áp dụng kiến trúc nhiều lớp (N-Layer) gồm: Entity (mô hình dữ liệu, DTOs, Enums), Data Access Layer (DAL – truy cập dữ liệu, repositories, migrations), Business Logic Layer (BLL – dịch vụ nghiệp vụ, validators) và Presentation (Razor Pages – giao diện người dùng, ViewModels, Helpers). Cách tổ chức này tách biệt trách nhiệm, nâng cao khả năng kiểm thử, mở rộng và bảo trì, đồng thời hỗ trợ phân quyền chức năng theo vai trò.

## Yêu cầu hệ thống và phạm vị triển khai

Yêu cầu hệ thống tập trung vào các chức năng hiển thị và quản lý sản phẩm, biến thể (màu/size), giỏ hàng, đơn hàng và phân quyền theo vai trò. Phạm vi triển khai trong đồ án tập trung vào ASP.NET Core 9.0 (Razor Pages) cho giao diện, EF Core 9.0 cho truy cập dữ liệu và PostgreSQL 16 cho cơ sở dữ liệu, chạy qua Docker Compose để đảm bảo môi trường nhất quán. Các nội dung như tích hợp cổng thanh toán, SEO nâng cao và phân tích dữ liệu chuyên sâu được ghi nhận là hướng phát triển tiếp theo.

## Công nghệ và môi trường

Backend sử dụng ASP.NET Core 9.0; giao diện sử dụng Razor Pages kết hợp Bootstrap 5 và JavaScript; truy cập dữ liệu qua Entity Framework Core 9.0. Cơ sở dữ liệu PostgreSQL 16 được khởi chạy nhất quán bằng Docker Compose, cấu hình kết nối quản lý trong `appsettings.json` và biến môi trường. Bảo mật cơ bản bảo đảm bằng hashing mật khẩu (BCrypt) và cơ chế CSRF.

## Dữ liệu và giả định nghiệp vụ

Mô hình dữ liệu phản ánh đặc thù sản phẩm giày với các quan hệ Category–Product–ProductVariant và Order–OrderItem. Giả định nghiệp vụ gồm: biến thể gắn với từng sản phẩm; tồn kho quản lý theo biến thể; đơn hàng gồm nhiều dòng hàng với số lượng và giá đơn vị; vai trò người dùng quyết định phạm vi thao tác (xem/CRUD/duyệt đơn hàng).

1. **Rủi ro và biện pháp giảm thiểu**

Rủi ro chủ yếu gồm: sai lệch mô hình dữ liệu so với yêu cầu nghiệp vụ; lỗi tương tác AJAX ở giỏ hàng; cấu hình môi trường Docker chưa nhất quán; và các vấn đề bảo mật cơ bản. Biện pháp giảm thiểu: kiểm thử theo luồng người dùng đại diện; rà soát lược đồ và migrations; sử dụng cấu hình chuẩn cho Docker Compose; áp dụng hashing mật khẩu và CSRF; theo dõi tiến độ và phản hồi định kỳ để điều chỉnh kịp thời.

CHƯƠNG 2: NGHIÊN CỨU LÝ THUYẾT

## 2.1. Kiến trúc N-Layer

Kiến trúc N-Layer tách biệt rõ trách nhiệm giữa các tầng, qua đó nâng cao khả năng bảo trì, kiểm thử và mở rộng. Tầng Presentation hiện thực giao diện bằng Razor Pages và PageModel, trao đổi dữ liệu qua ViewModel. Tầng Business Logic Layer (BLL) tập trung các quy tắc nghiệp vụ thông qua dịch vụ, kèm validators và helpers. Tầng Data Access Layer (DAL) đảm nhiệm truy cập dữ liệu với Repository, cấu hình DbContext và quản lý migrations. Tầng Entity định nghĩa mô hình dữ liệu, DTO và Enums dùng chung.

Luồng xử lý điển hình: người dùng tương tác trang Razor, PageModel tiếp nhận yêu cầu và gọi dịch vụ ở BLL; dịch vụ áp dụng quy tắc (ví dụ kiểm tra tồn kho, tính giá), phối hợp một hoặc nhiều Repository ở DAL để đọc/ghi dữ liệu, bao bọc giao dịch khi cần nhằm bảo đảm toàn vẹn. Dữ liệu qua lại giữa các tầng bằng ViewModel/DTO để tránh rò rỉ chi tiết mô hình, hạn chế phụ thuộc chặt chẽ và cho phép thay thế từng tầng với tác động tối thiểu.

Ưu điểm của cách tiếp cận này là dễ kiểm thử (có thể mock Repository), mở rộng chức năng theo chiều ngang và kiểm soát tốt thay đổi. Đánh đổi là tăng nhu cầu phối hợp và độ phức tạp; do đó cần tuân thủ phụ thuộc một chiều, áp dụng Dependency Injection và chuẩn hóa hợp đồng giữa các tầng (interfaces, DTOs). Kiến trúc này cũng thuận lợi để tích hợp các mối quan tâm cắt ngang như caching, logging và phân quyền.

**(Ghi chú: Có thể chèn ảnh minh họa luồng tương tác giữa các tầng N-Layer. Tên ảnh gợi ý: `nlayer-overview.png`.)**

## Razor Pages

Razor Pages là mô hình phát triển giao diện theo định hướng trang (page‑centric), trong đó mỗi trang gồm cặp tệp .cshtml và lớp PageModel tương ứng. Cơ chế Model Binding/Validation, Tag Helpers và Routing theo cấu trúc thư mục hỗ trợ xây dựng biểu mẫu, xử lý các phương thức OnGet/OnPost, xác thực dữ liệu và phát sinh antiforgery token mặc định, nhờ đó giảm mã lặp và bảo đảm an toàn cho thao tác gửi biểu mẫu.

Trong phạm vi dự án, Razor Pages được lựa chọn vì phù hợp với các chức năng CRUD và các luồng tương tác tuyến tính của người dùng (danh sách, chi tiết, thêm, sửa, xóa). Mô hình này giúp tách bạch trình bày và xử lý, dễ bảo trì, thuận tiện tích hợp DI/EF Core ở mức PageModel, hỗ trợ cập nhật bất đồng bộ (AJAX/partial render) và cấu hình phân quyền theo từng trang, qua đó giữ cho kiến trúc rõ ràng và nhất quán.

**(Ghi chú: Có thể chèn ảnh mô tả luồng xử lý một yêu cầu trong Razor Pages. Tên ảnh gợi ý: `razor-pages-flow.png`.)**

## Entity Framework Core

Entity Framework Core (EF Core) là ORM của .NET dùng để ánh xạ mô hình hướng đối tượng sang cơ sở dữ liệu quan hệ. Trong dự án, EF Core được sử dụng với DbContext, bộ migrations và LINQ để hiện thực các quan hệ một‑nhiều và nhiều‑nhiều giữa các thực thể như Category, Product, ProductVariant, Order và OrderItem. Cơ chế tracking/no‑tracking và các chiến lược nạp dữ liệu (eager/lazy/explicit) giúp cân bằng giữa tính nhất quán và hiệu năng khi truy vấn, đồng thời giữ cho logic nghiệp vụ tách biệt khỏi chi tiết SQL.

Các mẫu sử dụng chính gồm: truy vấn có phân trang và lọc (search/filter) qua LINQ; thao tác cập nhật/ghi nhận sự thay đổi với ChangeTracker; và quản lý lược đồ cơ sở dữ liệu thông qua migrations nhằm đồng bộ cấu trúc theo vòng đời phát triển. Để tối ưu, các truy vấn chỉ đọc dùng AsNoTracking, các thao tác ghi gói trong giao dịch khi cần đảm bảo toàn vẹn, kết hợp chỉ mục khóa ngoại/phổ biến để cải thiện hiệu năng. EF Core được chọn vì năng suất cao, hỗ trợ hệ sinh thái .NET và nhà cung cấp PostgreSQL ổn định, phù hợp cho yêu cầu CRUD và mở rộng sau này.

**(Ghi chú: Chèn ảnh lược đồ quan hệ giữa các thực thể chính. Tên ảnh gợi ý: `efcore-entities.png`.)**

## PostgreSQL

PostgreSQL 16 là hệ quản trị cơ sở dữ liệu quan hệ tuân thủ ACID với cơ chế MVCC tối ưu cho truy cập đồng thời. Hệ thống chỉ mục đa dạng (B‑tree, GIN, GiST, BRIN) cùng các ràng buộc toàn vẹn (khóa chính/ngoại, unique, check) giúp bảo đảm tính nhất quán dữ liệu và hiệu năng truy vấn ở các kiểu tải khác nhau. Các tính năng như transaction, savepoint, view/materialized view, trigger và policy (Row‑Level Security) tạo nền tảng vững chắc để hiện thực nghiệp vụ an toàn, có kiểm soát.

Về khả năng mở rộng, PostgreSQL hỗ trợ kiểu dữ liệu phong phú (JSONB, mảng, range, enum, composite), full‑text search tích hợp và hệ sinh thái extension mạnh (PostGIS, pg\_trgm, citext, hstore, FDW). JSONB kết hợp chỉ mục GIN cho phép truy vấn thuộc tính bán cấu trúc hiệu quả; pg\_trgm hỗ trợ tìm kiếm theo tương đồng/ký tự, citext giúp so khớp không phân biệt hoa thường. Trong đồ án, PostgreSQL được dùng theo mô hình quan hệ chuẩn với các bảng Category, Product, ProductVariant, User, Role, Order, OrderItem; áp dụng khóa ngoại và chỉ mục trên các cột truy vấn phổ biến (ví dụ khóa ngoại, SKU, tên sản phẩm). Các thao tác đặt hàng được gói trong transaction để bảo đảm toàn vẹn, và Docker Compose được dùng để khởi chạy cơ sở dữ liệu ổn định, dễ tái lập môi trường.

Các điểm nổi bật của PostgreSQL so với SQL Server (trong bối cảnh đồ án):

- Giấy phép/Chi phí: mã nguồn mở, miễn phí; không ràng buộc license trong giai đoạn phát triển và triển khai thử nghiệm.

- JSONB và chỉ mục: JSONB kèm chỉ mục GIN và toán tử phong phú giúp truy vấn thuộc tính linh hoạt; ở SQL Server, JSON lưu dưới dạng văn bản và thường cần cột tính toán để lập chỉ mục hiệu quả.

- Hệ sinh thái extension: phong phú với pg\_trgm (tìm kiếm tương đồng), citext (so khớp không phân biệt hoa thường), FDW, PostGIS… giúp mở rộng tính năng theo nhu cầu; SQL Server ít extension cộng đồng tương đương.

- Concurrency (MVCC): đọc không khóa mặc định, phù hợp các truy vấn liệt kê sản phẩm; SQL Server thường cần bật RCSI/SNAPSHOT để đạt hành vi tương tự.

- Docker & đa nền tảng: image chính thức nhẹ, chạy tốt trên macOS/Linux/Windows và khởi chạy nhanh với Docker Compose; image SQL Server thường nặng hơn.

- Kiểu dữ liệu: hỗ trợ mảng, range, enum, composite… thuận lợi khi mô hình hóa thuộc tính/biến thể; SQL Server thiếu một số kiểu tương đương trực tiếp.

**(Ghi chú: Ảnh Diagram CSDL toàn màn hình sẽ được chèn ở Chương 3.3. Tên ảnh gợi ý: `db-diagram.png`.)**

## Docker Compose

Docker Compose được sử dụng để khởi chạy PostgreSQL trong môi trường phát triển theo cách nhất quán và có thể tái lập. Tệp cấu hình lựa chọn image `postgres:16-alpine` nhằm giảm dung lượng và tăng tốc độ khởi động; chính sách `restart: unless-stopped` giúp container tự khởi động lại khi gặp sự cố. Cách tiếp cận một dịch vụ cơ sở dữ liệu độc lập giúp nhóm phát triển trên macOS/Linux/Windows có cùng nền tảng, hạn chế sai lệch “works‑on‑my‑machine”.

Cấu hình dùng biến môi trường với giá trị mặc định (cú pháp `${VAR:-default}`) để đặt tên CSDL, tài khoản, mật khẩu và cổng lắng nghe; ánh xạ cổng `${POSTGRES\_PORT:-5432}:5432` để linh hoạt khi máy đã dùng cổng 5432. Dữ liệu được lưu bền trên volume `postgres\_data`, thư mục dữ liệu nội bộ được đặt qua `PGDATA` để tách biệt rõ ràng. Healthcheck sử dụng `pg\_isready` nhằm kiểm tra trạng thái sẵn sàng; dịch vụ tham gia mạng bridge `shoesshop\_network` để cô lập lưu lượng. Khi ứng dụng .NET chạy cục bộ, chuỗi kết nối trỏ đến `localhost` và cổng đã ánh xạ, hoặc tham chiếu các biến môi trường tương ứng trong `appsettings.json`.

Trong vận hành, Compose cho phép tùy biến thông qua tệp `.env` mà không cần sửa YAML; khuyến nghị đặt mật khẩu ở biến môi trường thay vì hard‑code. Khi chuyển sang môi trường dài hạn, cần bổ sung cơ chế sao lưu/khôi phục volume, chính sách bảo mật (role/permission) và giám sát tài nguyên container; đồng thời có thể mở rộng Compose để thêm công cụ quản trị tùy chọn (Adminer/pgAdmin) khi cần.

**(Ghi chú: Có thể chèn ảnh minh họa vòng đời container và healthcheck. Tên ảnh gợi ý: `docker-compose-services.png`.)**

## Bảo mật Cơ bản

Hệ thống lưu trữ mật khẩu theo cơ chế băm một chiều với BCrypt, sử dụng salt ngẫu nhiên và tham số chi phí (work factor) để tăng độ khó khi tấn công vét cạn. Cách tiếp cận này giúp vô hiệu hóa khả năng khôi phục mật khẩu gốc từ dữ liệu băm, đồng thời giảm thiểu tác động khi xảy ra rò rỉ cơ sở dữ liệu. Thành phần hiện thực tại `ShoesShopWeb.BLL/Helpers/PasswordHasher.cs` bao gói logic tạo băm và xác minh, tách biệt chi tiết thuật toán khỏi phần còn lại của ứng dụng nhằm dễ bảo trì và kiểm thử.

Đối với bảo vệ khỏi tấn công giả mạo yêu cầu liên trang (CSRF), Razor Pages tích hợp antiforgery token vào biểu mẫu và xác minh trên mỗi yêu cầu POST. Token được gắn với phiên người dùng, được kết xuất tự động qua Tag Helpers và kiểm tra ở phía máy chủ, ngăn chặn kịch bản kẻ tấn công lợi dụng trình duyệt nạn nhân gửi yêu cầu trái phép. Kết hợp với xác thực người dùng và cookie cấu hình an toàn (HttpOnly, Secure), cơ chế này tạo lớp phòng thủ quan trọng cho các thao tác ghi dữ liệu như thêm vào giỏ hàng, cập nhật sản phẩm hay duyệt đơn hàng.

Phân quyền dựa trên vai trò (Role‑Based Authorization) được áp dụng nhất quán ở tầng trình bày, giới hạn truy cập các trang và thao tác nhạy cảm theo ba vai trò chính: Admin, Staff và Customer. Các quy tắc ủy quyền có thể đặt ở mức trang hoặc thư mục trang, giúp kiểm soát rõ phạm vi chức năng mỗi vai trò; đồng thời có thể mở rộng sang mô hình policy‑based để diễn đạt điều kiện phức tạp hơn khi cần. Cách tiếp cận theo vai trò giúp đơn giản hóa quản trị, tăng khả năng tuân thủ và giảm rủi ro lộ lọt dữ liệu nội bộ.

**(Ghi chú: Có thể chèn ảnh sơ đồ quyền truy cập theo vai trò. Tên ảnh gợi ý: `rbac-overview.png`.)**

CHƯƠNG 3: HIỆN THỰC HOÁ NGHIÊN CỨU

## Mô tả bài toán

Bài toán đặt ra là xây dựng một hệ thống thương mại điện tử chuyên bán giày, cho phép người dùng cuối duyệt, tìm kiếm và lựa chọn sản phẩm theo danh mục, màu sắc và kích cỡ, đồng thời hỗ trợ quy trình đặt hàng trọn vẹn. Hệ thống phải đảm bảo trải nghiệm nhất quán từ trang danh sách đến chi tiết sản phẩm, giỏ hàng và thanh toán, với dữ liệu sản phẩm có nhiều biến thể được mô hình hóa đúng đắn để tránh nhầm lẫn về tồn kho và giá. Kiến trúc triển khai cần ổn định, dễ mở rộng và dễ bảo trì nhằm đáp ứng yêu cầu học thuật lẫn ứng dụng thực tế.

Ở góc độ khách hàng, hệ thống cho phép đăng ký/đăng nhập, xem danh sách sản phẩm theo phân trang, lọc và tìm kiếm theo từ khóa, truy cập trang chi tiết để chọn biến thể phù hợp (màu, size) và thêm vào giỏ. Giỏ hàng hỗ trợ cập nhật số lượng theo thời gian thực (AJAX), xóa mục và tiến hành đặt hàng; người dùng có thể xem lại lịch sử đơn hàng sau khi hoàn tất. Các yêu cầu phi chức năng bao gồm giao diện responsive, tính ổn định trong môi trường phát triển và thời gian phản hồi chấp nhận được cho các thao tác thường gặp.

Ở góc độ quản trị, nhân viên và quản trị viên có quyền truy cập bảng điều khiển thống kê cơ bản, quản lý danh mục, sản phẩm và biến thể, quản lý người dùng và đơn hàng. Hệ thống áp dụng phân quyền theo vai trò để bảo vệ các thao tác nhạy cảm, bổ sung cơ chế phòng ngừa xóa dữ liệu ngoài ý muốn và hỗ trợ sinh mã SKU tự động nhằm chuẩn hóa quản lý sản phẩm. Phạm vi hiện thực tập trung vào các nghiệp vụ cốt lõi nêu trên; các chức năng nâng cao như tích hợp cổng thanh toán hoặc báo cáo phân tích chuyên sâu được ghi nhận là hướng phát triển tiếp theo.

## Yêu cầu chức năng

|  |  |  |
| --- | --- | --- |
| **Vai trò** | **Chức năng** | **Mô tả ngắn** |
| Customer | Đăng ký/Đăng nhập | Tạo tài khoản, đăng nhập hệ thống |
| Customer | Xem sản phẩm | Phân trang, lọc, tìm kiếm theo từ khóa |
| Customer | Giỏ hàng | Thêm/Xóa/Sửa số lượng; AJAX cập nhật |
| Customer | Đặt hàng | Tạo đơn hàng; xem lịch sử đơn hàng |
| Staff/Admin | Dashboard | Thống kê sản phẩm/đơn hàng cơ bản |
| Staff/Admin | Quản lý danh mục | CRUD danh mục sản phẩm |
| Staff/Admin | Quản lý sản phẩm | CRUD sản phẩm, biến thể màu/size |
| Staff/Admin | Quản lý khách hàng | Danh sách và thông tin người dùng |
| Staff/Admin | Quản lý đơn hàng | Duyệt, cập nhật trạng thái |

Yêu cầu phi chức năng: hiệu năng chấp nhận được trong môi trường localhost, bảo mật cơ bản (hashing, CSRF), giao diện responsive.

## Mô hình cơ sở dữ liệu

**--------**

Mô tả từng bảng (ví dụ tối giản):

* Category: Id, Name, Description.
* Product: Id, Name, CategoryId, Description, BasePrice, SKU.
* ProductVariant: Id, ProductId, Color, Size, Stock, PriceDelta.
* User: Id, Email, PasswordHash, DisplayName, Role.
* Order: Id, UserId, CreatedAt, Status, Total.
* OrderItem: Id, OrderId, ProductVariantId, Quantity, UnitPrice.

Mô tả chi tiết từng thực thể:

* Category: đại diện nhóm sản phẩm (ví dụ: Sneaker, Sandal, Boots), dùng để tổ chức trình bày và lọc. Trường Name là duy nhất trong phạm vi cửa hàng để tránh trùng lặp; có thể bổ sung Description cho mục đích SEO và hiển thị.
* Product: mô tả sản phẩm cơ sở trước khi phân tách thành biến thể. Mỗi sản phẩm có CategoryId, Name, Description, BasePrice và SKU để phục vụ quản lý kho và tra cứu. SKU được sinh theo quy ước nhằm đảm bảo tính duy nhất và thuận tiện khi đối soát; BasePrice là nền tảng cho phép định giá biến thể.
* ProductVariant: nắm giữ các thuộc tính cụ thể như Color và Size, kèm Stock để quản lý tồn kho theo biến thể, và PriceDelta để điều chỉnh giá bán so với BasePrice. Khóa ngoại ProductId liên kết biến thể với sản phẩm mẹ; có thể áp dụng ràng buộc duy nhất trên cặp (ProductId, Color, Size) để ngăn trùng lặp biến thể.
* User: đại diện khách hàng hoặc nhân sự nội bộ, gồm Email, PasswordHash, DisplayName và Role (hoặc liên kết bảng Role). Email được dùng để đăng nhập; nên chuẩn hóa so khớp không phân biệt hoa thường và đảm bảo duy nhất. PasswordHash lưu theo cơ chế BCrypt; thông tin hồ sơ có thể mở rộng thêm địa chỉ, số điện thoại.
* Role: định nghĩa vai trò hệ thống như Admin, Staff, Customer, phục vụ phân quyền truy cập. Có thể liên kết nhiều‑nhiều với User nếu cần gán đa vai trò; trong phạm vi đồ án, mô hình một vai trò/người dùng là đủ.
* Order: ghi nhận đơn hàng với UserId, thời điểm tạo CreatedAt, Status (ví dụ: Pending, Confirmed, Shipped, Canceled) và Total. Tổng đơn được tính từ các dòng OrderItem trong transaction để bảo đảm toàn vẹn; trạng thái đơn hàng hỗ trợ luồng duyệt/cập nhật của nhân viên.
* OrderItem: dòng hàng thuộc một đơn hàng, liên kết OrderId và ProductVariantId, chứa Quantity và UnitPrice tại thời điểm đặt hàng. Việc lưu UnitPrice ở thời điểm giao dịch giúp đảm bảo tính bất biến của hoá đơn dù giá biến thể thay đổi sau này; ràng buộc Quantity > 0 và kiểm tra Stock là cần thiết để ngăn oversell.

Chỉ mục được đề xuất trên các khóa ngoại (CategoryId, ProductId, OrderId, ProductVariantId) và các cột tìm kiếm phổ biến như Product.Name và Product.SKU để tối ưu liệt kê và tra cứu. Ràng buộc Quantity > 0 cho OrderItem và kiểm soát tồn kho ở mức ProductVariant.Stock giúp ngăn đặt hàng vượt quá số lượng. Trường Order.Total có thể được tính từ tổng Quantity × UnitPrice của các dòng OrderItem, kiểm tra nhất quán trong transaction khi tạo/ cập nhật đơn hàng. Với PostgreSQL, có thể xem xét citext cho email (so khớp không phân biệt hoa thường) và sử dụng chỉ mục phù hợp để cải thiện hiệu năng truy vấn.

## Kiến trúc hệ thống

Công nghệ và môi trường:

- Backend: ASP.NET Core 9.0; BLL thực thi logic nghiệp vụ; DAL giao tiếp database qua EF Core 9.0.

- Frontend: Razor Pages, Bootstrap 5, JavaScript; thư mục `Pages/` chứa các trang như `Products`, `Cart`, `Orders`, `Staff`.

- Database: PostgreSQL 16 chạy bằng Docker Compose; kết nối cấu hình trong `appsettings.json`.

(Ghi chú: Có thể chèn ảnh “Kiến trúc N-Layer của dự án”.)

Kiến trúc áp dụng mô hình nhiều lớp (N‑Layer) để tách biệt trách nhiệm. Tầng Presentation hiện thực giao diện với Razor Pages: mỗi trang gồm file `.cshtml` và lớp `PageModel` xử lý các phương thức `OnGet/OnPost`, trao đổi dữ liệu qua `ViewModels`. Tầng BLL tập trung quy tắc nghiệp vụ (ví dụ: kiểm tra tồn kho, tính giá bán theo `BasePrice` và `PriceDelta`, xác thực dữ liệu đầu vào) dưới dạng services/validators. Tầng DAL sử dụng EF Core với `DbContext`, repository và migrations để truy cập, ánh xạ và quản lý lược đồ cơ sở dữ liệu. Tầng Entity định nghĩa các `Models`, `DTOs`, `Enums` dùng chung giữa các tầng, giúp chuẩn hóa hợp đồng dữ liệu.

Luồng điển hình: yêu cầu từ người dùng tới một trang Razor, `PageModel` gọi service ở BLL để thực thi nghiệp vụ; service phối hợp repository ở DAL để đọc/ghi dữ liệu, gói thao tác ghi trong transaction khi cần (ví dụ tạo đơn hàng kèm các dòng `OrderItem` và trừ tồn `ProductVariant.Stock`). Dữ liệu trả về được chuyển thành `ViewModel` để trình bày, tránh rò rỉ chi tiết mô hình nội bộ. Các mối quan tâm cắt ngang (logging, xử lý lỗi, phân quyền) được đặt ở tầng biên: middleware/filters cho kiểm soát chung, attributes/policies cho ủy quyền trang, logger DI cho ghi nhận sự kiện nghiệp vụ.

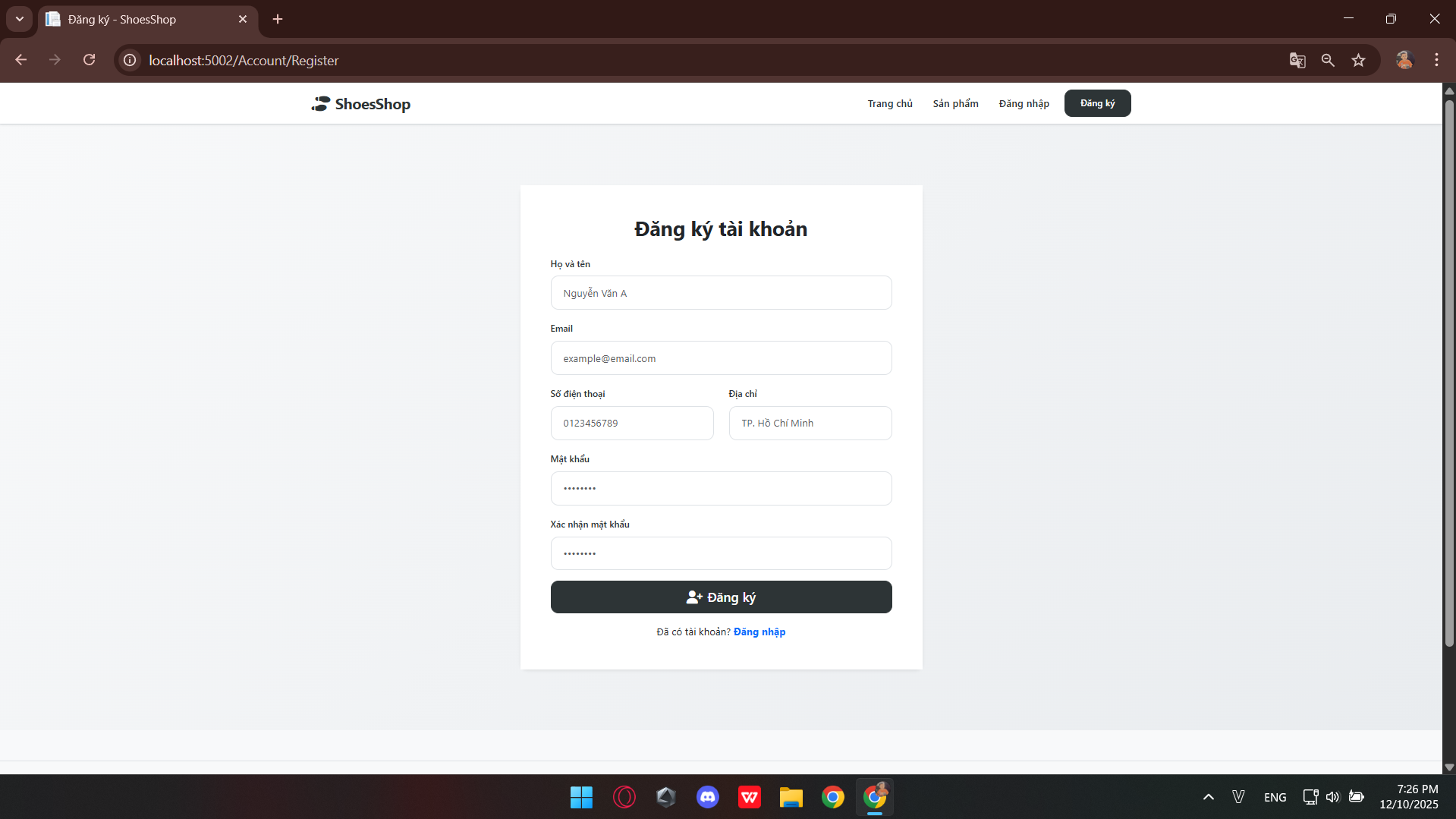
Khía cạnh vận hành: cấu hình kết nối PostgreSQL và các thông số ứng dụng đặt trong `appsettings.json` và biến môi trường; DI đăng ký `DbContext`, services BLL và repository DAL trong `Program.cs`. Migrations quản lý vòng đời lược đồ, đảm bảo đồng bộ với code qua các lần phát triển; static files (CSS/JS/lib) phục vụ từ `wwwroot`, giao diện responsive với Bootstrap. Bảo mật cơ bản gồm hashing mật khẩu (BCrypt), antiforgery token cho POST trong Razor Pages và phân quyền theo vai trò cho khu vực quản trị (`Staff`, `Admin`). Ghi log và xử lý lỗi sử dụng pipeline ASP.NET Core (developer exception page trong dev), kết hợp thông báo thân thiện trên giao diện để nâng cao trải nghiệm.

# CHƯƠNG 4: KẾT QUẢ NGHIÊN CỨU

Sau khi hoàn tất quá trình xây dựng, hệ thống đã được triển khai thành công trên môi trường phát triển (Localhost) theo cấu hình Razor Pages. Các chức năng được hiện thực hóa dựa trên kiến trúc phân lớp, sử dụng UnitOfWork và các Service để xử lý nghiệp vụ. Giao diện người dùng và trang quản trị hoạt động ổn định, đảm bảo quy trình xử lý dữ liệu chặt chẽ từ Frontend xuống Backend. Phần này trình bày chi tiết kết quả thực nghiệm theo từng luồng chức năng, kèm theo các hình ảnh minh họa thực tế từ mã nguồn.

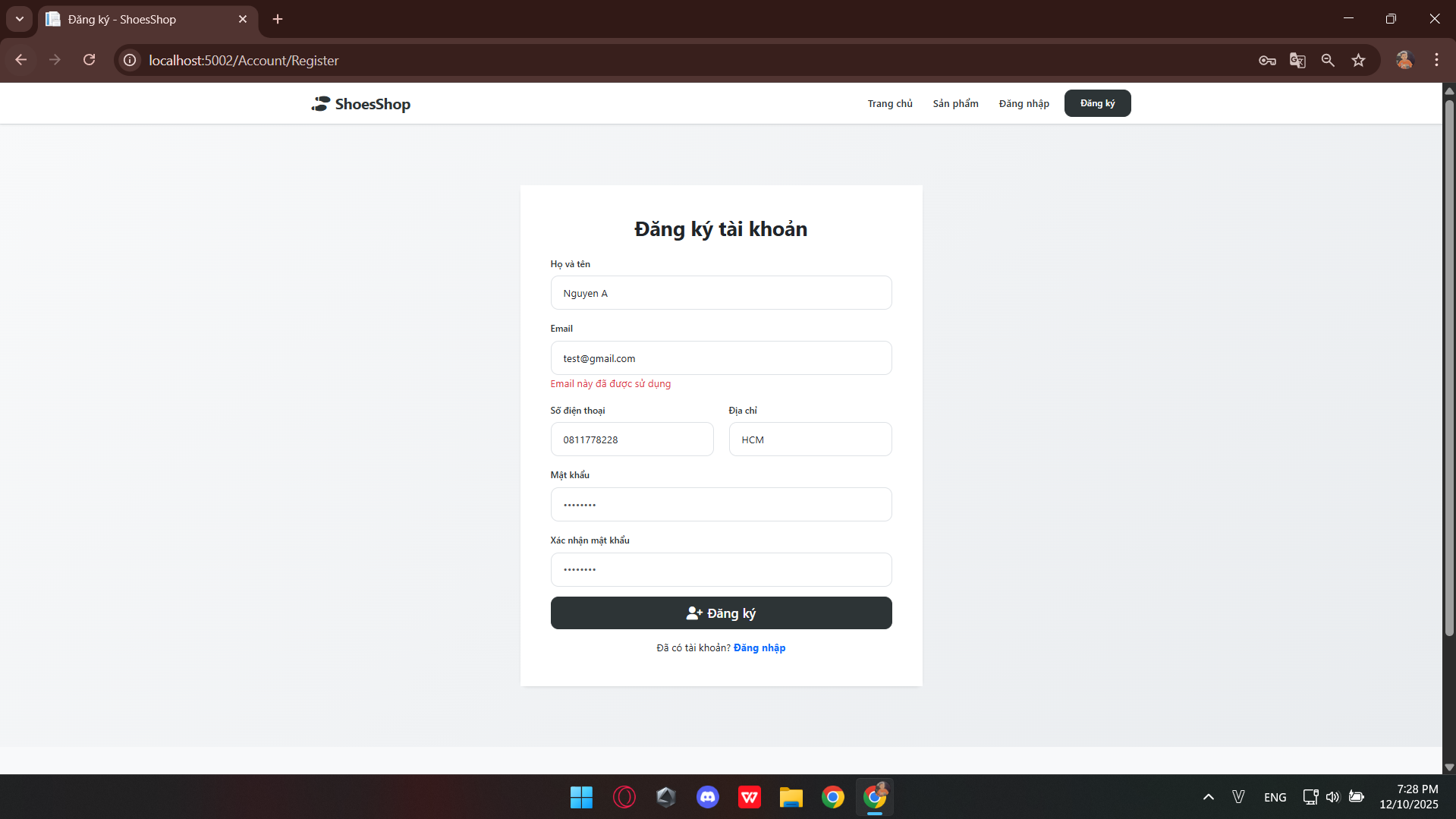
## 4.1. Kết quả theo luồng chức năng người dùng

Chức năng đăng ký tài khoản được hiện thực tại trang Pages/Account/Register.cshtml. Giao diện cung cấp form nhập liệu gồm các trường FullName, Email, Phone, Address và Password. Hệ thống áp dụng validation chặt chẽ thông qua RegisterModel.InputModel. Khi người dùng truy cập trang này (phương thức OnGet), hệ thống sẽ tự động chuyển hướng về trang chủ (/Index) nếu phát hiện người dùng đã đăng nhập.



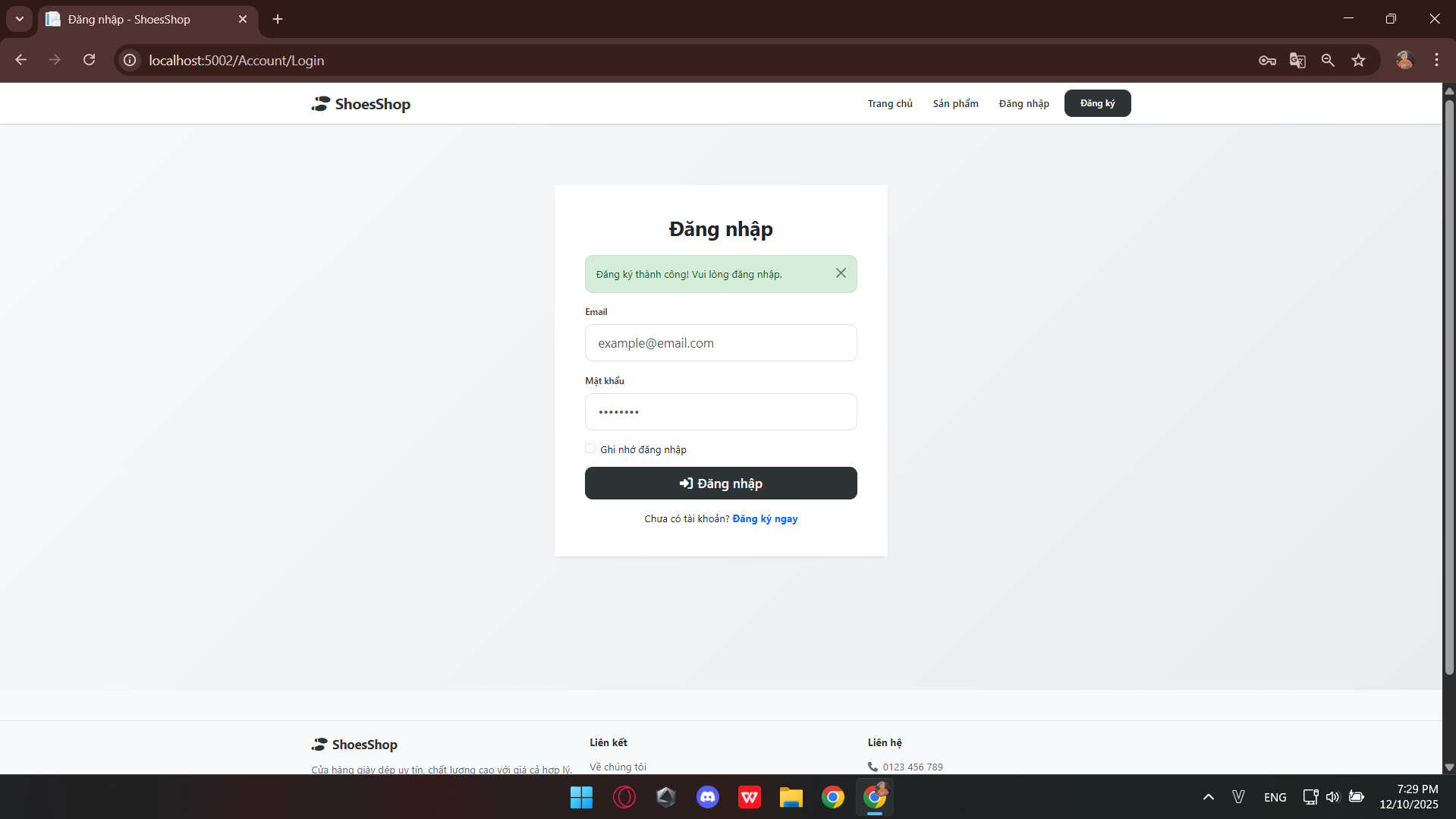
**Hình 4.1 – Form đăng ký:** Giao diện hiển thị các trường thông tin và cơ chế validation.

Khi người dùng gửi dữ liệu (OnPostAsync), hệ thống kiểm tra tính hợp lệ của ModelState và xác thực sự tồn tại của email thông qua \_userService.GetUserEntityByEmailAsync. Nếu phát sinh lỗi (ví dụ: email trùng, định dạng sai), thông báo lỗi sẽ hiển thị ngay bên dưới trường tương ứng.



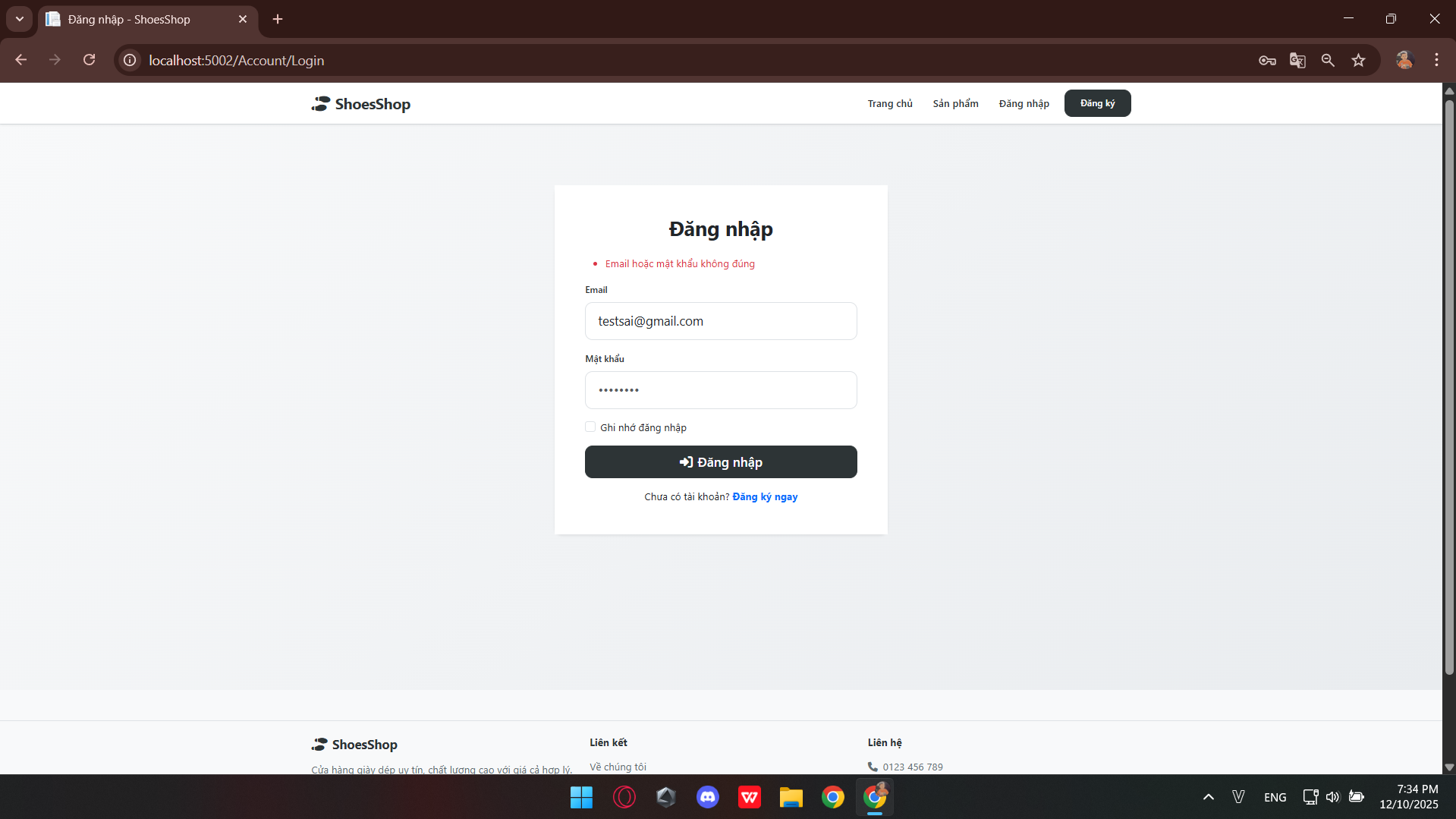
**Hình 4.2 – Lỗi đăng ký:** Giao diện hiển thị thông báo lỗi khi validation thất bại hoặc email đã tồn tại.

Trong trường hợp dữ liệu hợp lệ, RegisterModel tiến hành mã hóa mật khẩu (hashing) và lưu thông tin người dùng mới. TempData sẽ ghi nhận thông báo “Đăng ký thành công! Vui lòng đăng nhập” trước khi chuyển hướng người dùng sang trang đăng nhập.



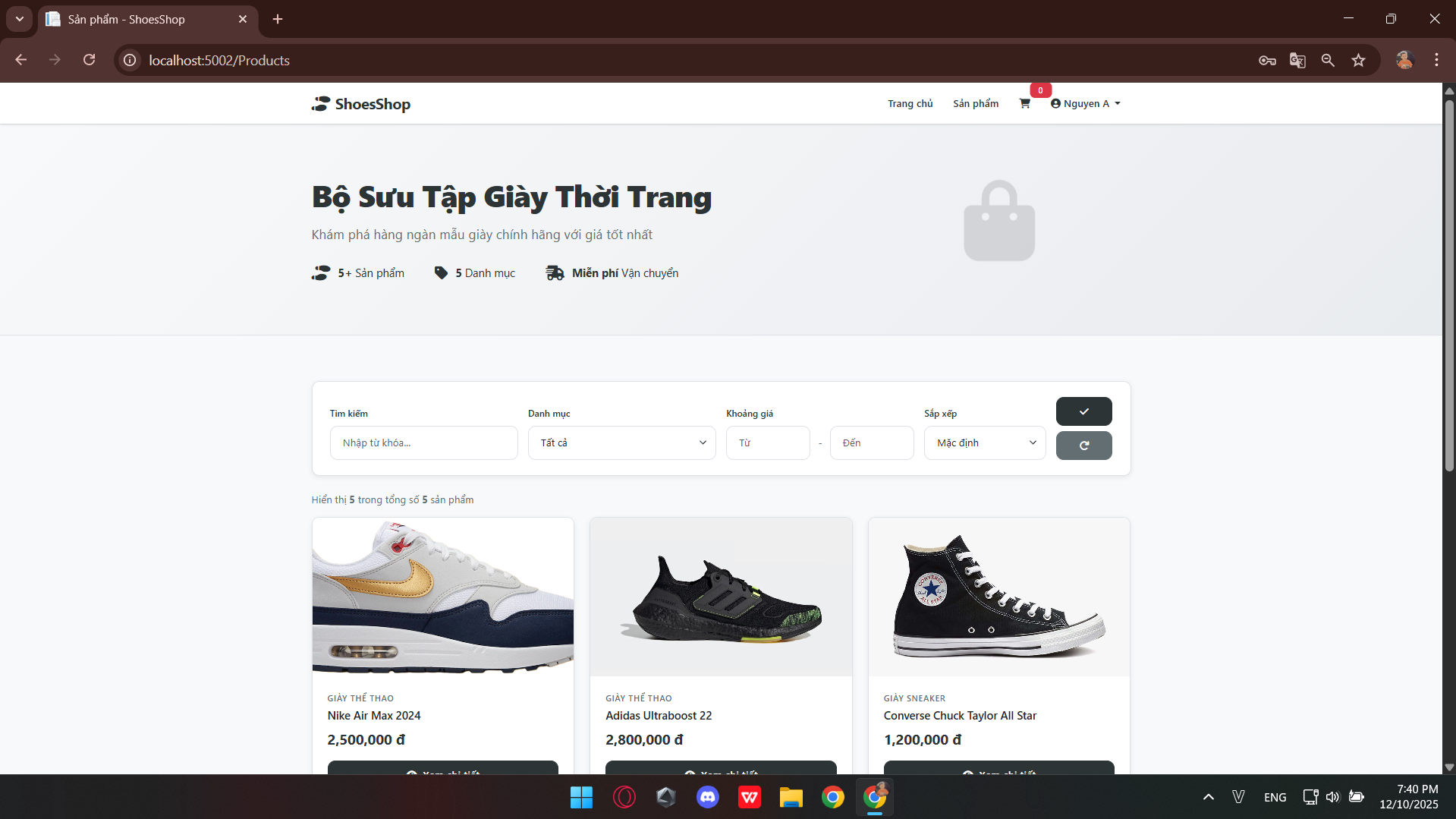
**Hình 4.3 – Đăng ký thành công:** Thông báo thành công hiển thị trước khi chuyển hướng.

Chức năng đăng nhập tại Pages/Account/Login.cshtml hỗ trợ tham số ReturnUrl để điều hướng người dùng quay lại trang trước đó sau khi đăng nhập. Giao diện bao gồm tùy chọn “Ghi nhớ đăng nhập”. Khi submit, dữ liệu được gửi đến LoginModel.OnPostAsync. Nếu thông tin xác thực sai (không tìm thấy user hoặc sai mật khẩu từ \_userService), lỗi sẽ được thêm vào ModelState.



**Hình 4.4 – Lỗi đăng nhập:** Thông báo “Email hoặc mật khẩu không đúng”.

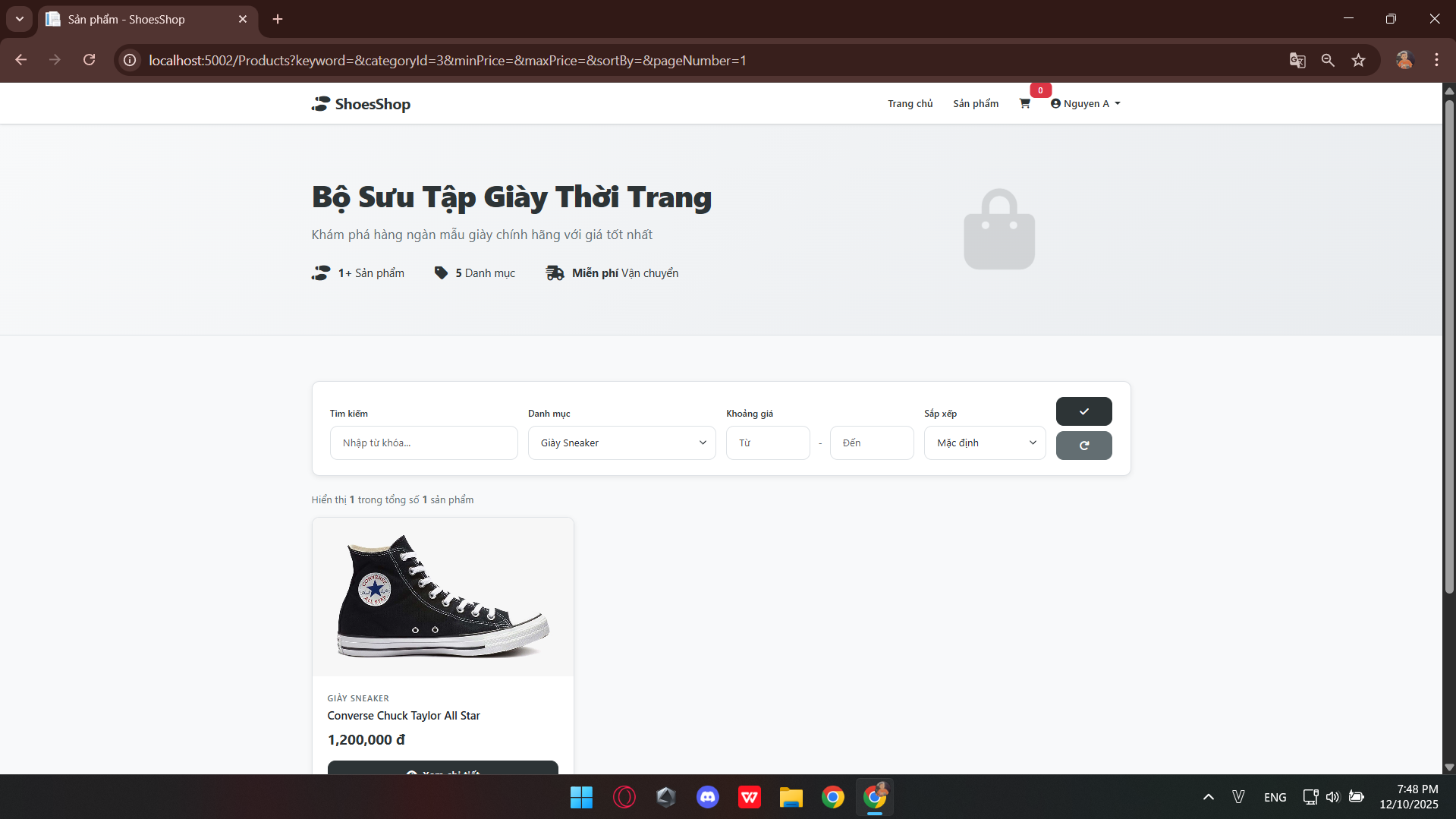
Sau khi đăng nhập thành công (HttpContext.SignInAsync), thanh điều hướng (Navbar) sẽ đọc ClaimTypes.Name từ cookie để hiển thị tên người dùng, thay thế nút đăng nhập bằng menu cá nhân (My Orders, Logout).



**Hình 4.5 – Navbar sau đăng nhập:** Hiển thị tên người dùng và menu chức năng.

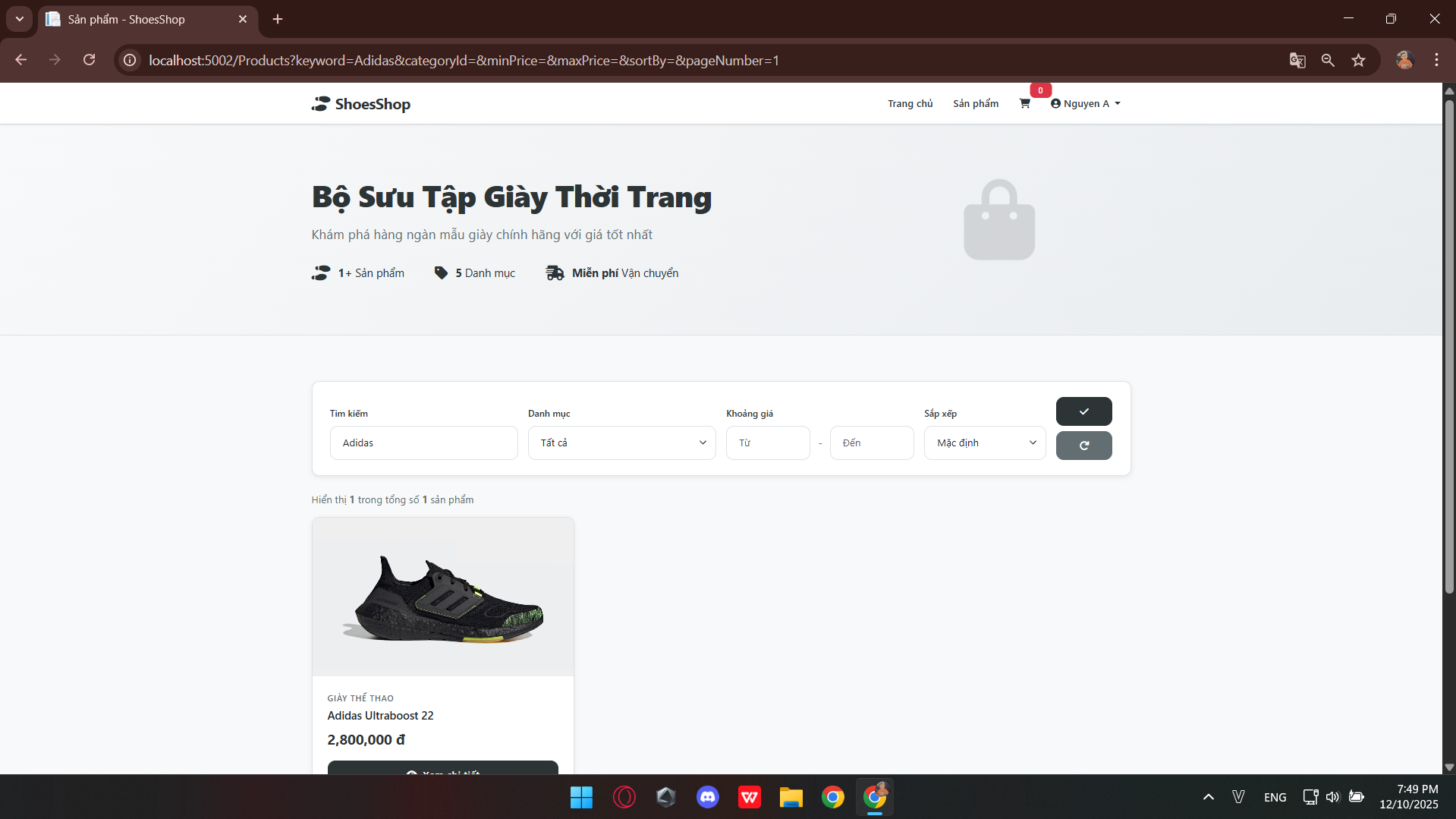
Trang danh sách sản phẩm (Pages/Products/Index.cshtml) sử dụng phương thức GetProductsWithFiltersAsync để nạp dữ liệu. Hệ thống hiển thị dưới dạng lưới (grid) với tối đa 12 sản phẩm mỗi trang. Mỗi thẻ sản phẩm (card) bao gồm hình ảnh, tên, giá cơ bản và badge (nhãn) hiển thị tên danh mục.

Thanh ngang tích hợp bộ lọc đa tiêu chí, lấy dữ liệu động từ \_categoryService, \_sizeService, và \_colorService. Người dùng có thể lọc theo Danh mục, Size hoặc Màu sắc. Khi nhấn nút lọc, IndexModel sẽ tái truy vấn dữ liệu dựa trên các tham số đã chọn.



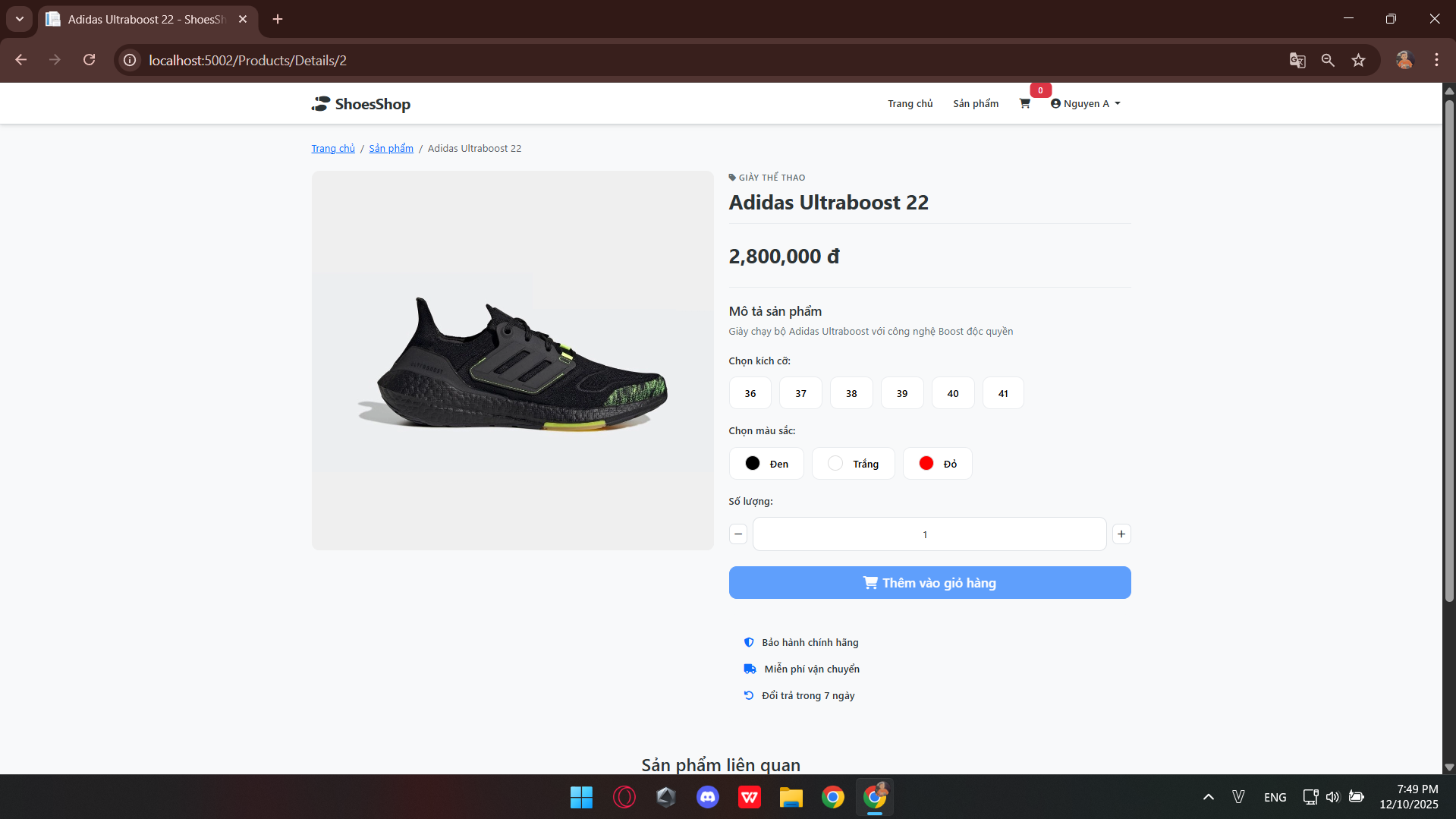
**Hình 4.7 – Kết quả filter nâng cao:** Kết quả trả về danh sách rút gọn (ví dụ: Sneaker) kèm tổng số kết quả tìm thấy.

Chức năng tìm kiếm từ khóa hoạt động bằng cách truyền tham số keyword vào IndexModel.OnGetAsync. Backend thực hiện lọc theo tên hoặc mô tả sản phẩm, kết quả trả về sẽ highlight các từ khóa trùng khớp (ví dụ: "Nike Air").



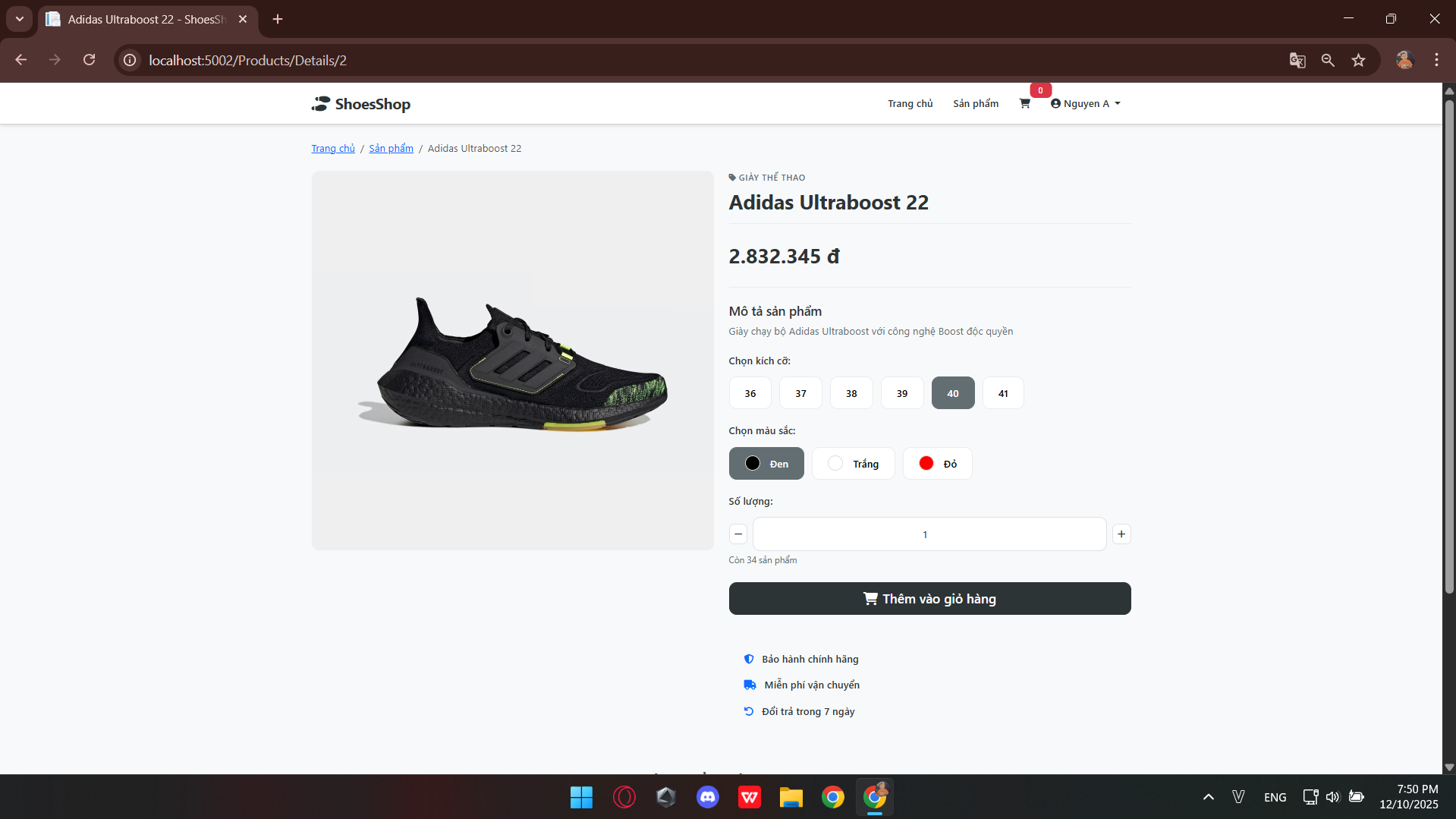
**Hình 4.8 – Tìm kiếm:** Kết quả tìm kiếm theo từ khóa.

Trang chi tiết sản phẩm (Pages/Products/Details.cshtml) được xử lý bởi DetailsModel. Hệ thống gọi \_productService.GetProductWithDetailsAsync để lấy thông tin sản phẩm gốc cùng danh sách các biến thể (Variants). Giao diện hiển thị bao gồm bộ sưu tập ảnh (gallery), bảng thông số kỹ thuật và danh sách sản phẩm liên quan.



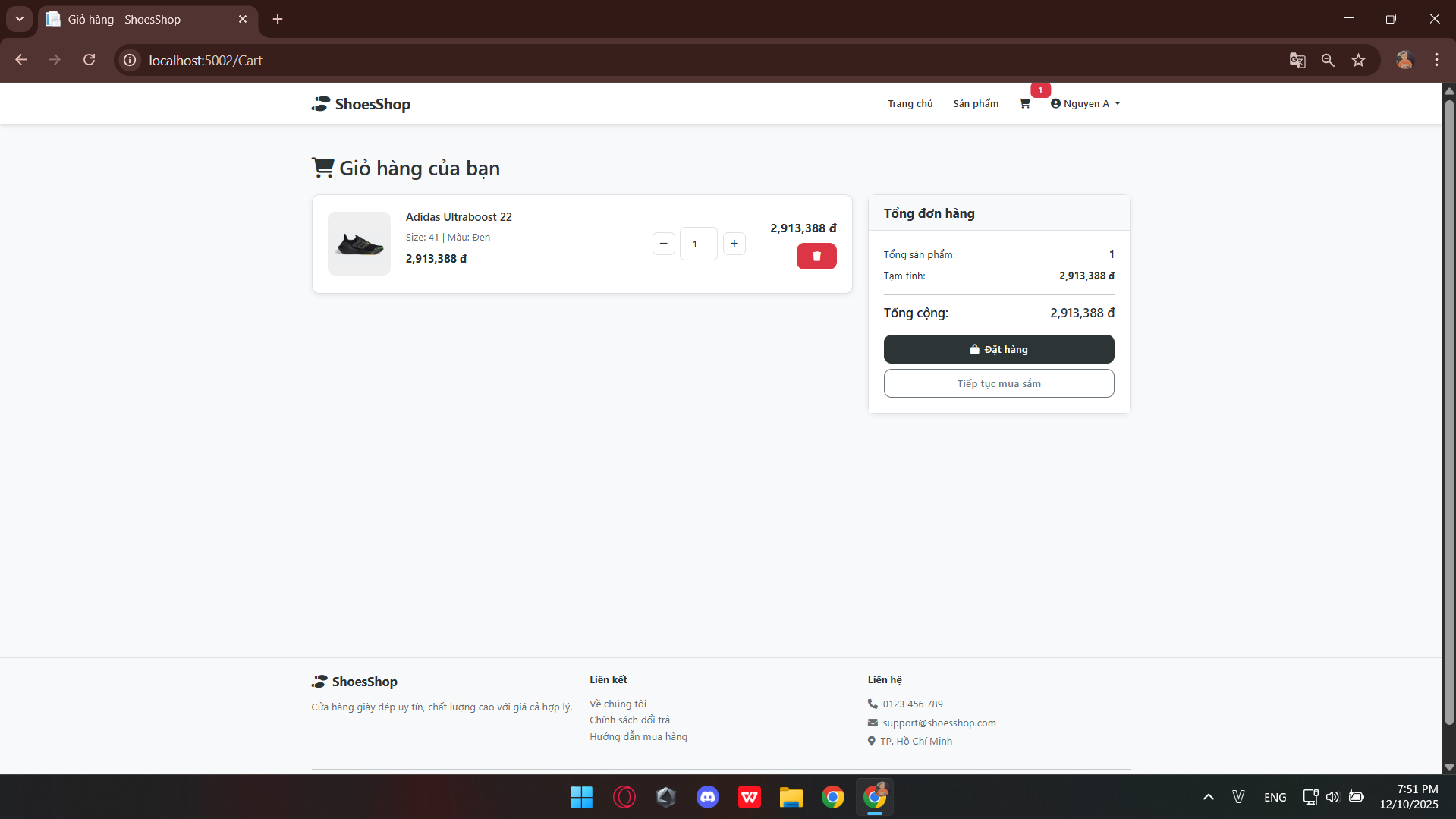
**Hình 4.9 – Trang chi tiết:** Giao diện thông tin chi tiết sản phẩm.

Khu vực chọn biến thể cho phép người dùng chọn Màu và Size. Logic phía Client-side sẽ binding với danh sách Product.Variants để cập nhật động giá bán và số lượng tồn kho (StockQuantity) tương ứng ngay khi người dùng thay đổi lựa chọn.



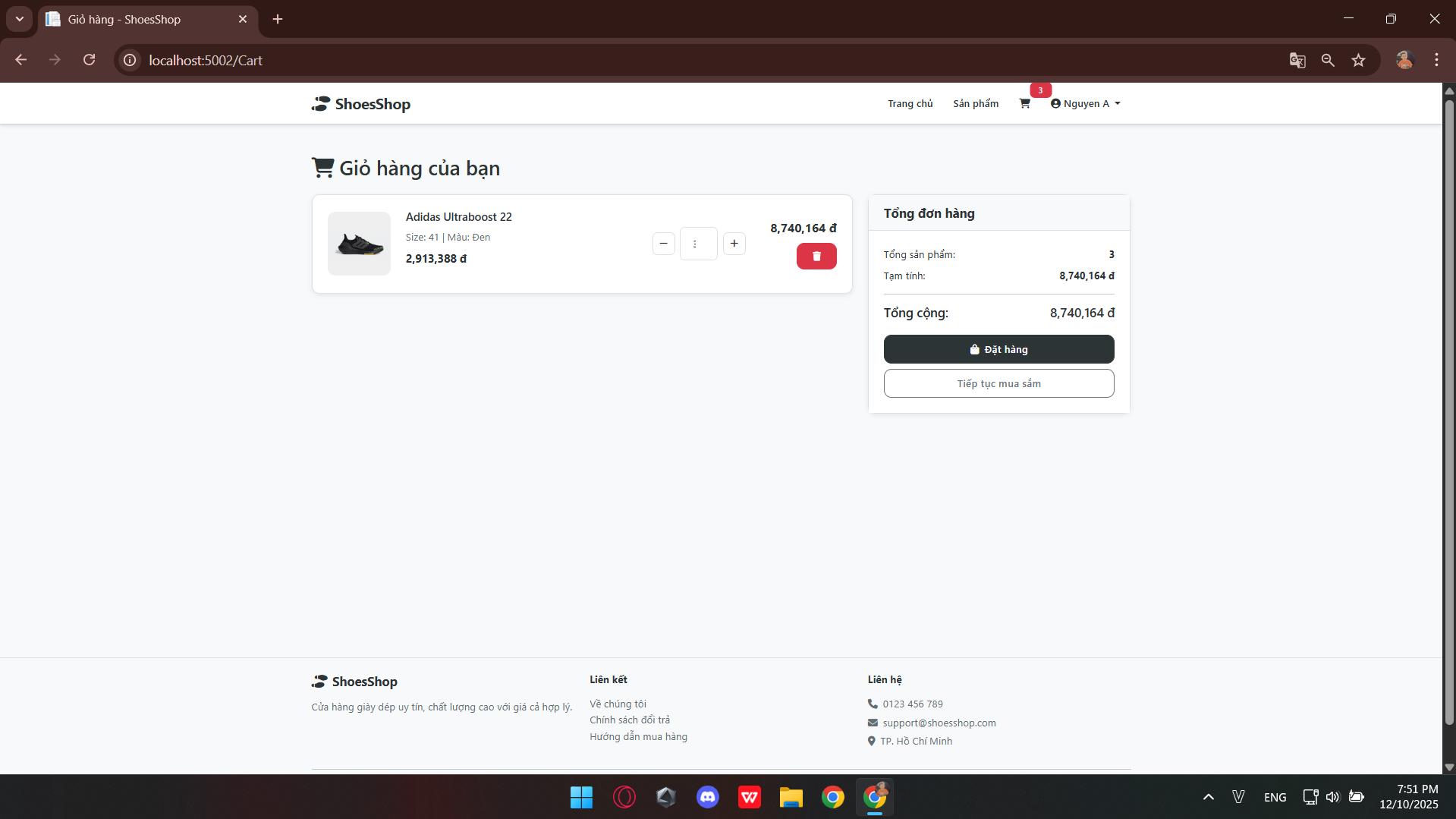
**Hình 4.10 – Chọn biến thể:** Giao diện cập nhật giá và tồn kho khi chọn màu/size.

Giỏ hàng được quản lý tại Pages/Cart/Index.cshtml (yêu cầu quyền Authorize). Khi tải trang, hệ thống truy vấn giỏ hàng theo userId, thực hiện join các bảng ProductVariants, Product, Size, Color để hiển thị đầy đủ thông tin từng dòng: hình ảnh, màu, size.



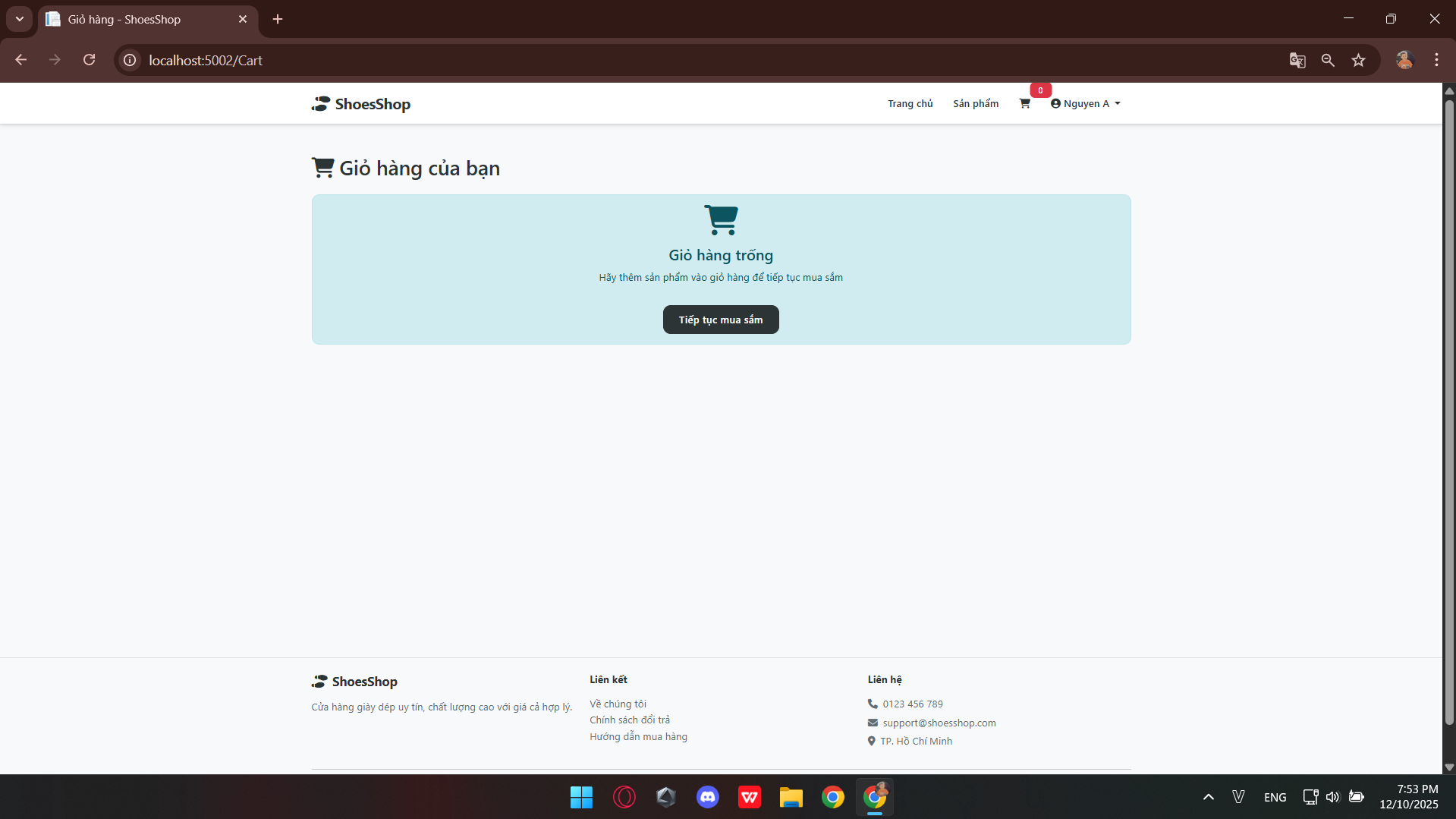
**Hình 4.11 – Giỏ hàng:** Danh sách các sản phẩm trong giỏ.

Chức năng cập nhật số lượng sử dụng AJAX gọi đến handler OnPostUpdateQuantityAsync. Khi cập nhật thành công, Client sẽ tính toán lại TotalPrice của dòng và TotalAmount của cả giỏ hàng ngay lập tức mà không cần tải lại toàn bộ trang.



**Hình 4.14 – Cập nhật số lượng AJAX:** Giao diện thay đổi giá trị tổng tiền theo thời gian thực.

Trường hợp người dùng xóa hết sản phẩm, IndexModel trả về một CartViewModel rỗng. View sẽ hiển thị trạng thái “Giỏ hàng trống” kèm nút quay lại trang mua sắm.

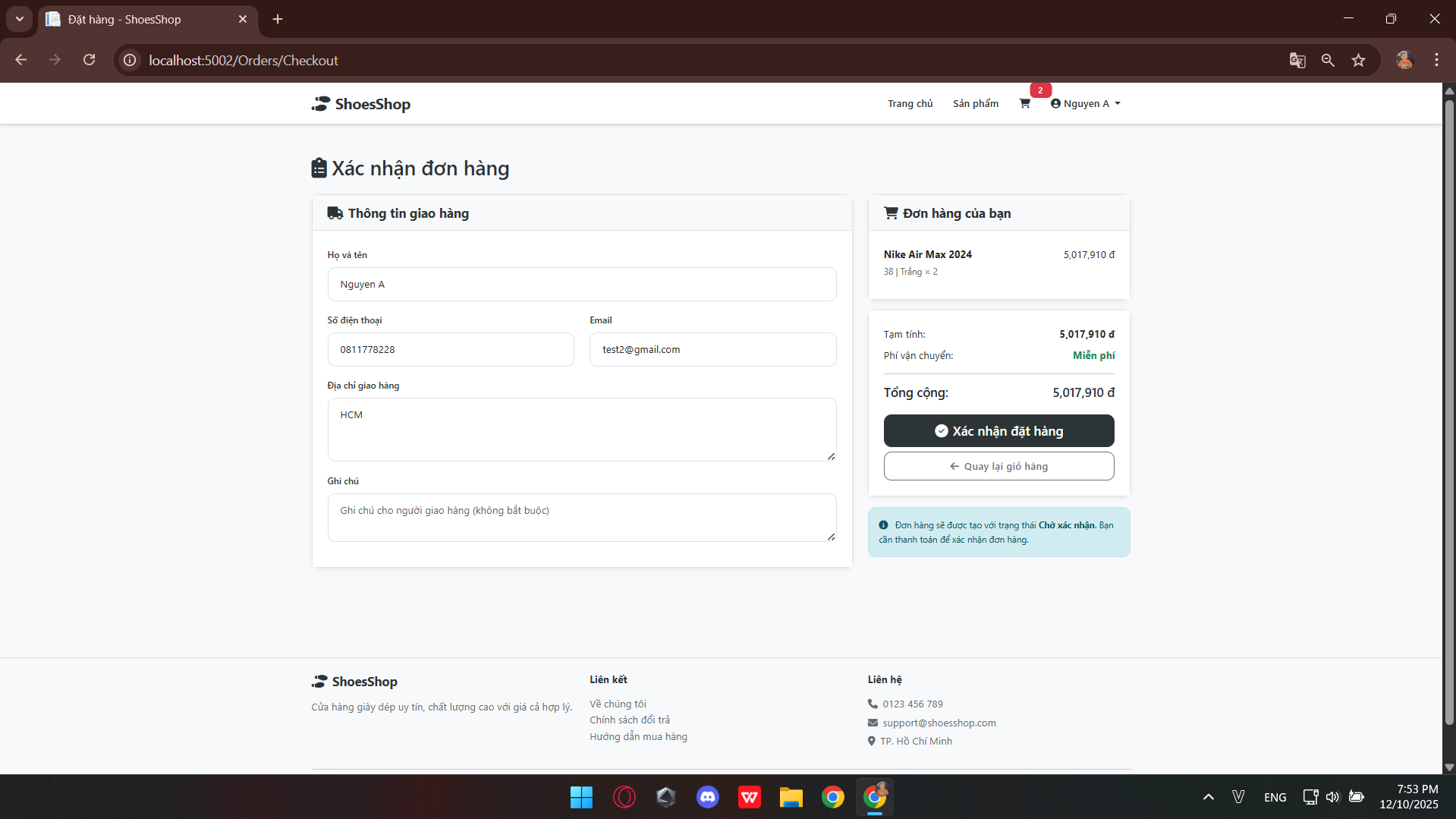


**Hình 4.15 – Giỏ trống:** Giao diện thông báo khi không có sản phẩm trong giỏ.

Quy trình thanh toán bắt đầu tại route /Orders/Checkout. Trang này được bảo vệ bởi attribute [Authorize]. Nếu người dùng chưa đăng nhập, Middleware sẽ chặn và chuyển hướng về trang Login kèm ReturnUrl.

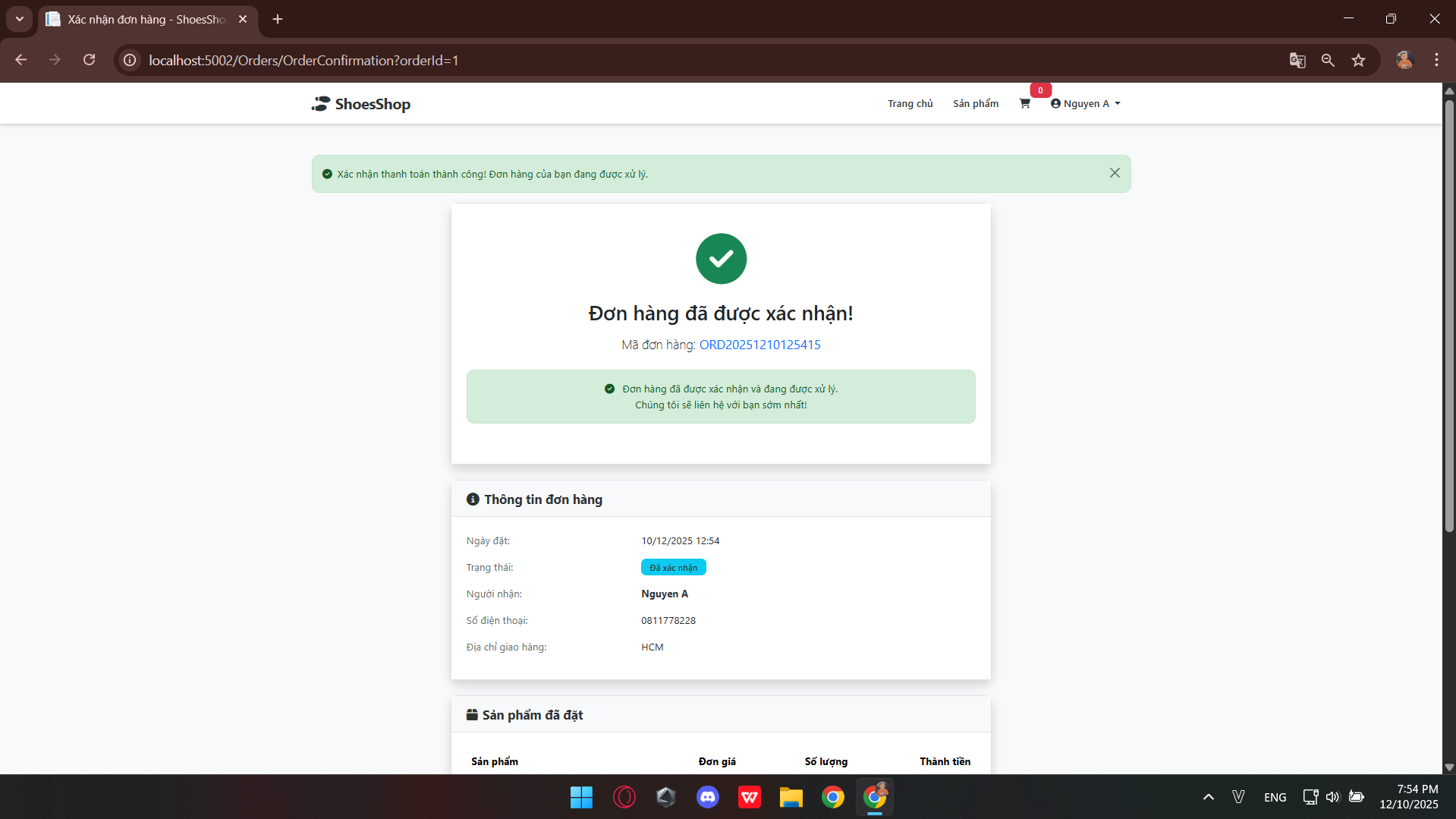
**Hình 4.17 – Yêu cầu đăng nhập trước checkout:** Chuyển hướng khi truy cập checkout mà chưa xác thực.

Tại trang Checkout, CheckoutModel.OnGetAsync tự động điền sẵn thông tin Họ tên, Email, Địa chỉ từ hồ sơ người dùng vào form. Giao diện hiển thị tóm tắt danh sách hàng hóa và form nhập ghi chú/xác nhận giao hàng.



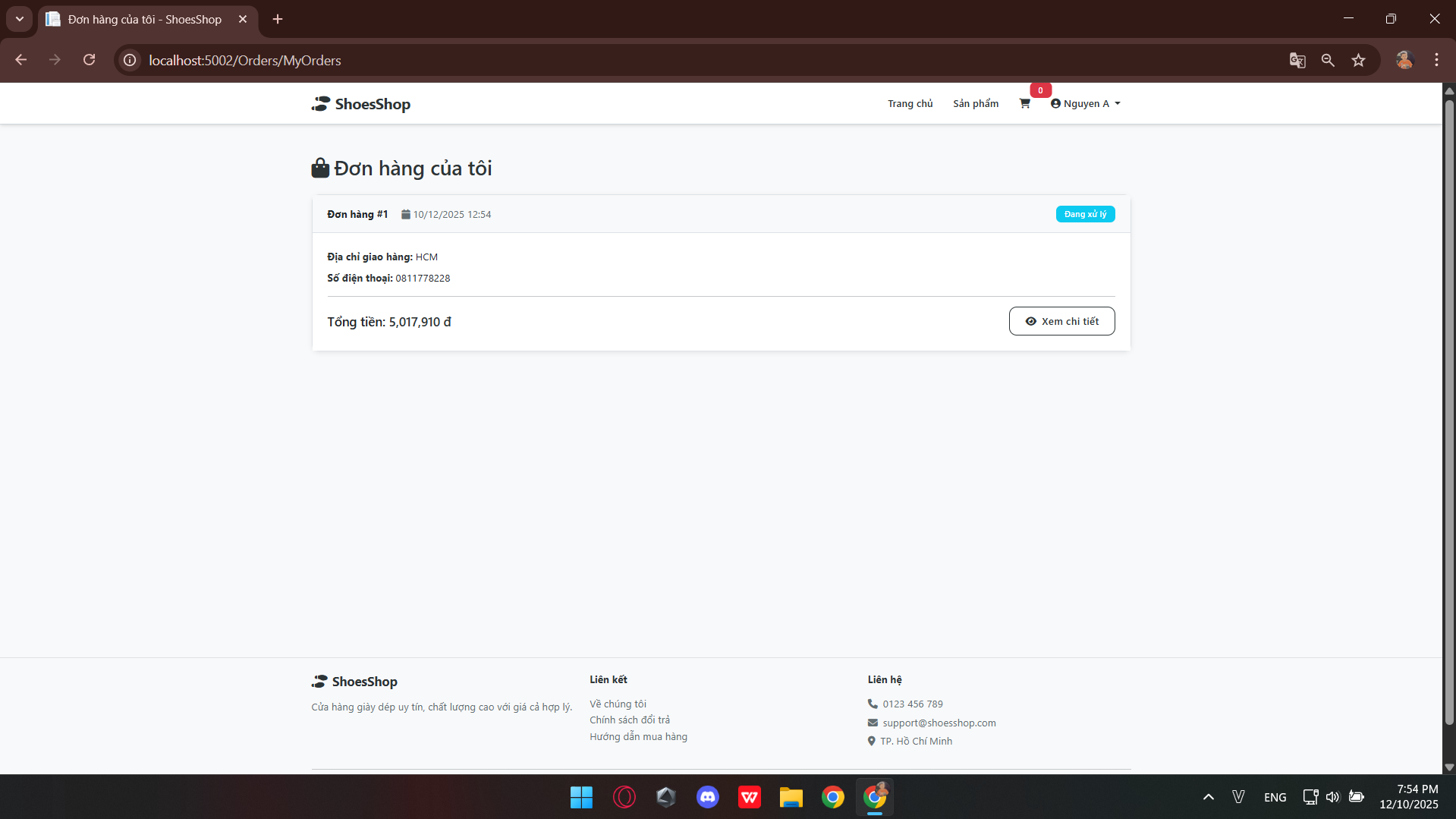
**Hình 4.18 – Form checkout:** Giao diện xác nhận thông tin đặt hàng.

Khi người dùng xác nhận (OnPostAsync), hệ thống thực hiện chuỗi tác vụ: tạo bản ghi Order (trạng thái Pending), sinh mã đơn hàng (OrderNumber), trừ số lượng tồn kho của các biến thể liên quan, xóa giỏ hàng và chuyển hướng tới trang xác nhận /Orders/OrderConfirmation.



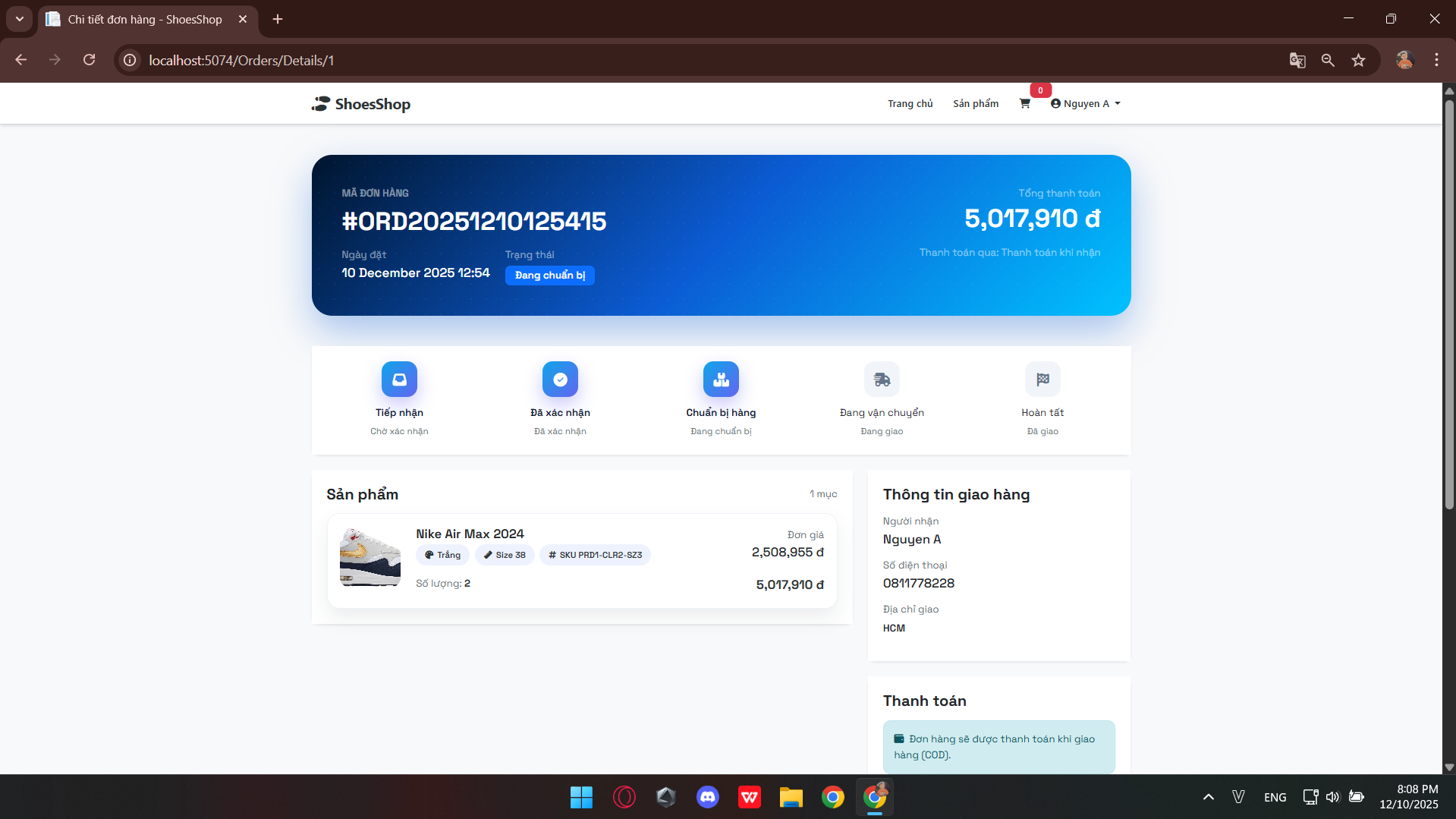
**Hình 4.19 – Xác nhận đơn:** Thông báo đặt hàng thành công.

Người dùng theo dõi đơn hàng tại trang My Orders. MyOrdersModel gọi phương thức GetOrdersByUserIdAsync để lấy toàn bộ đơn hàng, sắp xếp theo OrderDate giảm dần. Trạng thái đơn hàng (Pending, Processing, Shipping, Delivered) được hiển thị dưới dạng các badge màu sắc trực quan.



**Hình 4.21 – Lịch sử đơn hàng:** Danh sách đơn hàng cá nhân.

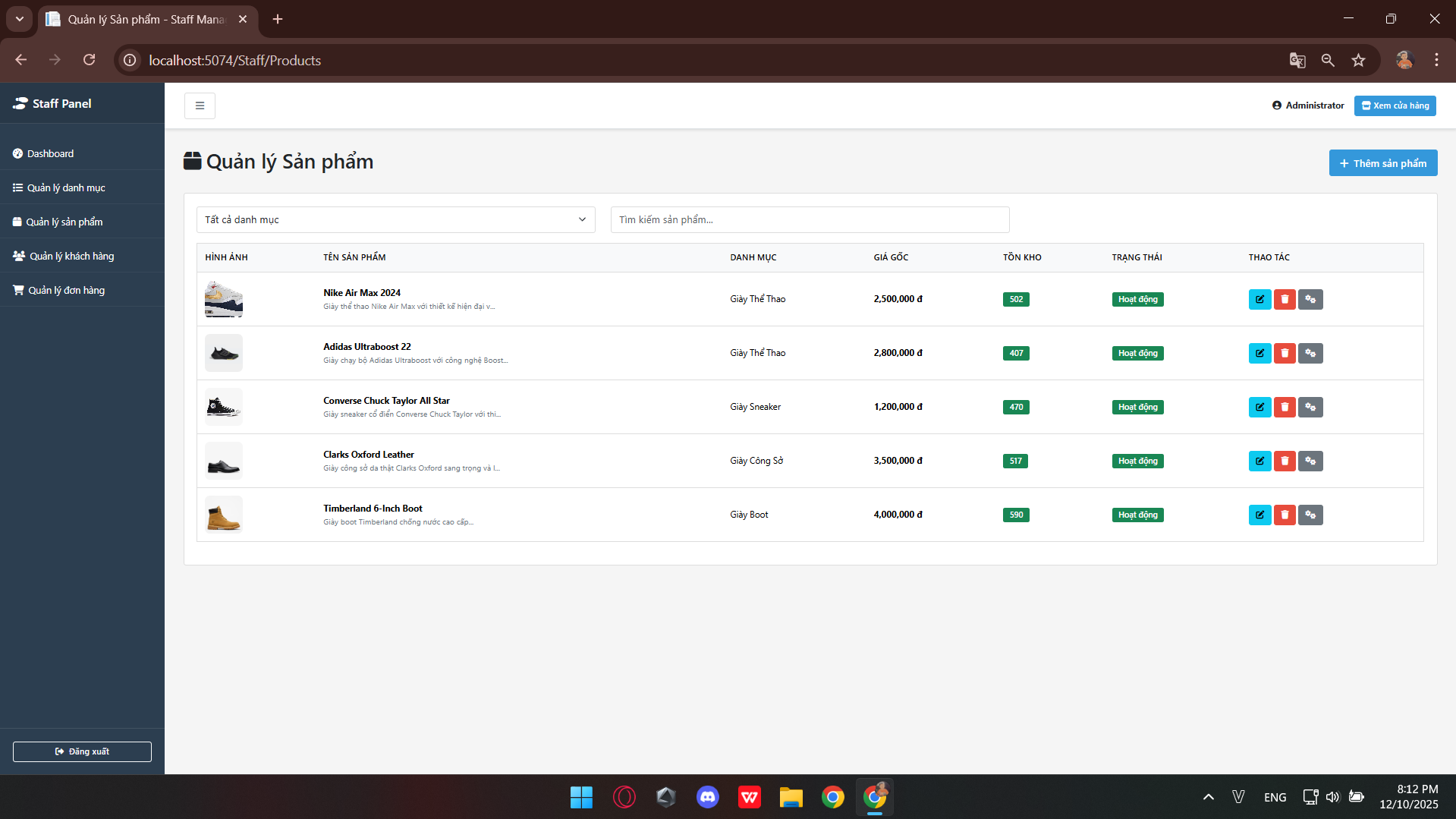
Khi xem chi tiết một đơn hàng, hệ thống gọi \_unitOfWork.Orders.GetOrderWithDetailsAsync để tải dữ liệu chi tiết từng OrderItem (tên giày, màu, size, giá) và timeline xử lý.



**Hình 4.22 – Chi tiết đơn:** Giao diện chi tiết một đơn hàng cụ thể.

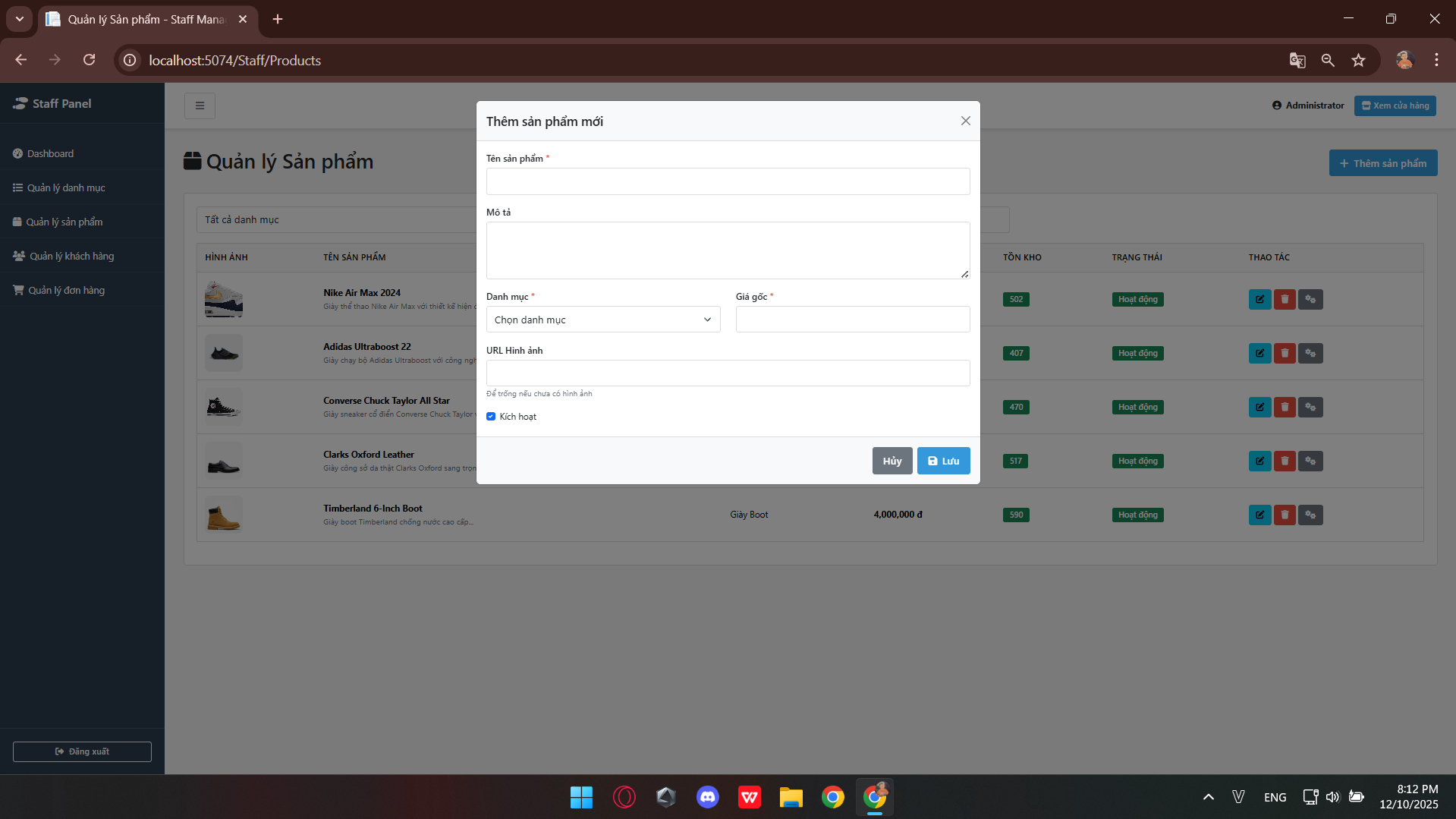
## 4.2. Kết quả theo luồng chức năng quản trị (Staff/Admin)

Trang quản lý sản phẩm (Pages/Staff/Products.cshtml) liệt kê danh sách sản phẩm kèm danh mục, số lượng biến thể và trạng thái kích hoạt (IsActive). Dữ liệu được lấy thông qua \_unitOfWork.Products.GetAllAsync.



**Hình 4.23 – Bảng sản phẩm Admin:** Danh sách quản lý sản phẩm.

Chức năng Thêm/Sửa sử dụng Modal Popup, gọi đến các handler OnPostCreateAsync và OnPostUpdateAsync. Form cho phép upload ảnh, thay đổi danh mục và bật tắt trạng thái sản phẩm.



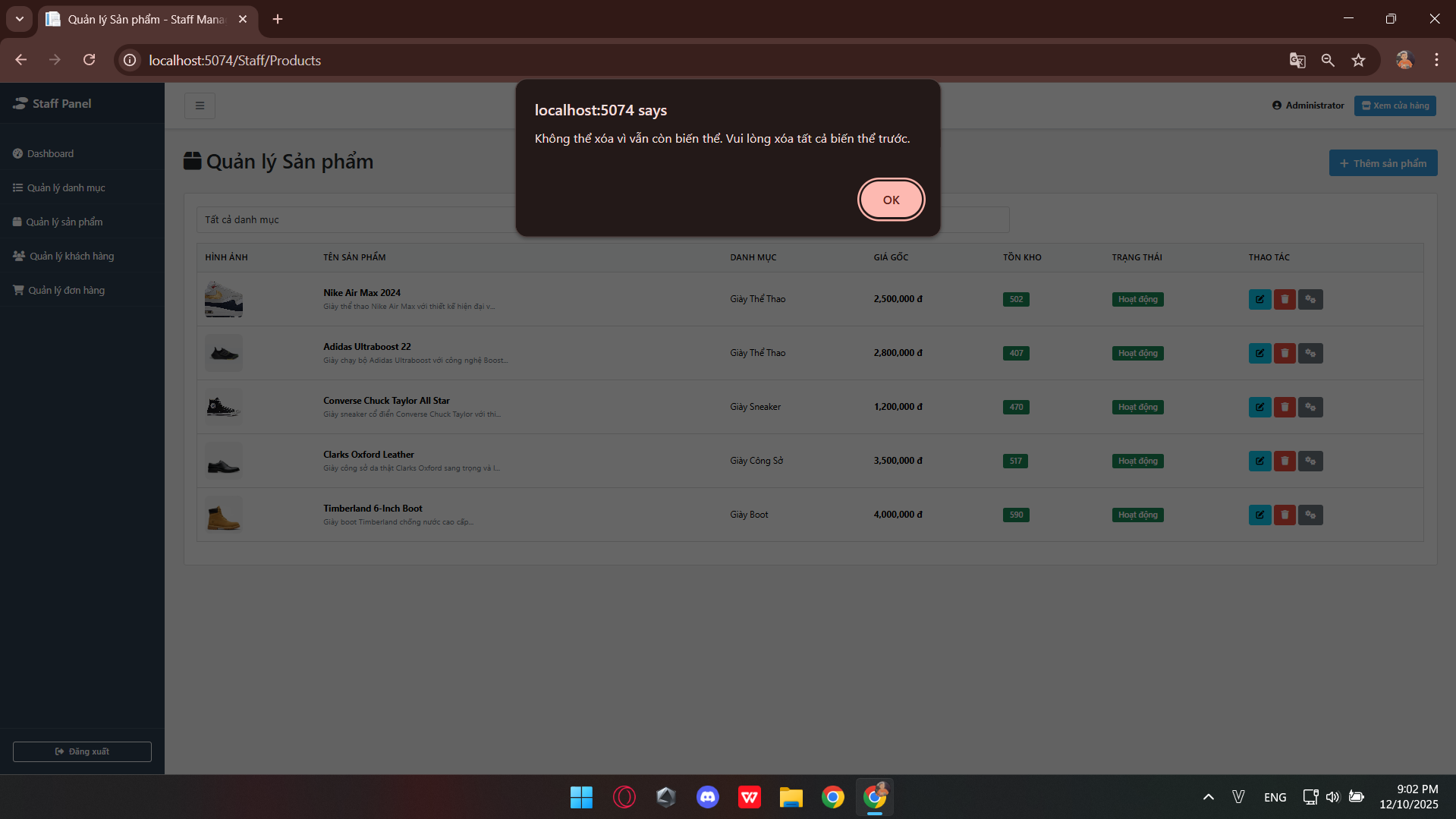
**Hình 4.24– Form sản phẩm:** Modal thêm mới/chỉnh sửa sản phẩm.

Quản lý biến thể tại Pages/Staff/ProductVariants.cshtml cho phép nạp các size/màu hiện có. Quản trị viên có thể nhập SKU, giá bán và tồn kho cho từng biến thể. Nếu bỏ trống SKU, Server sẽ tự động sinh mã định danh này.



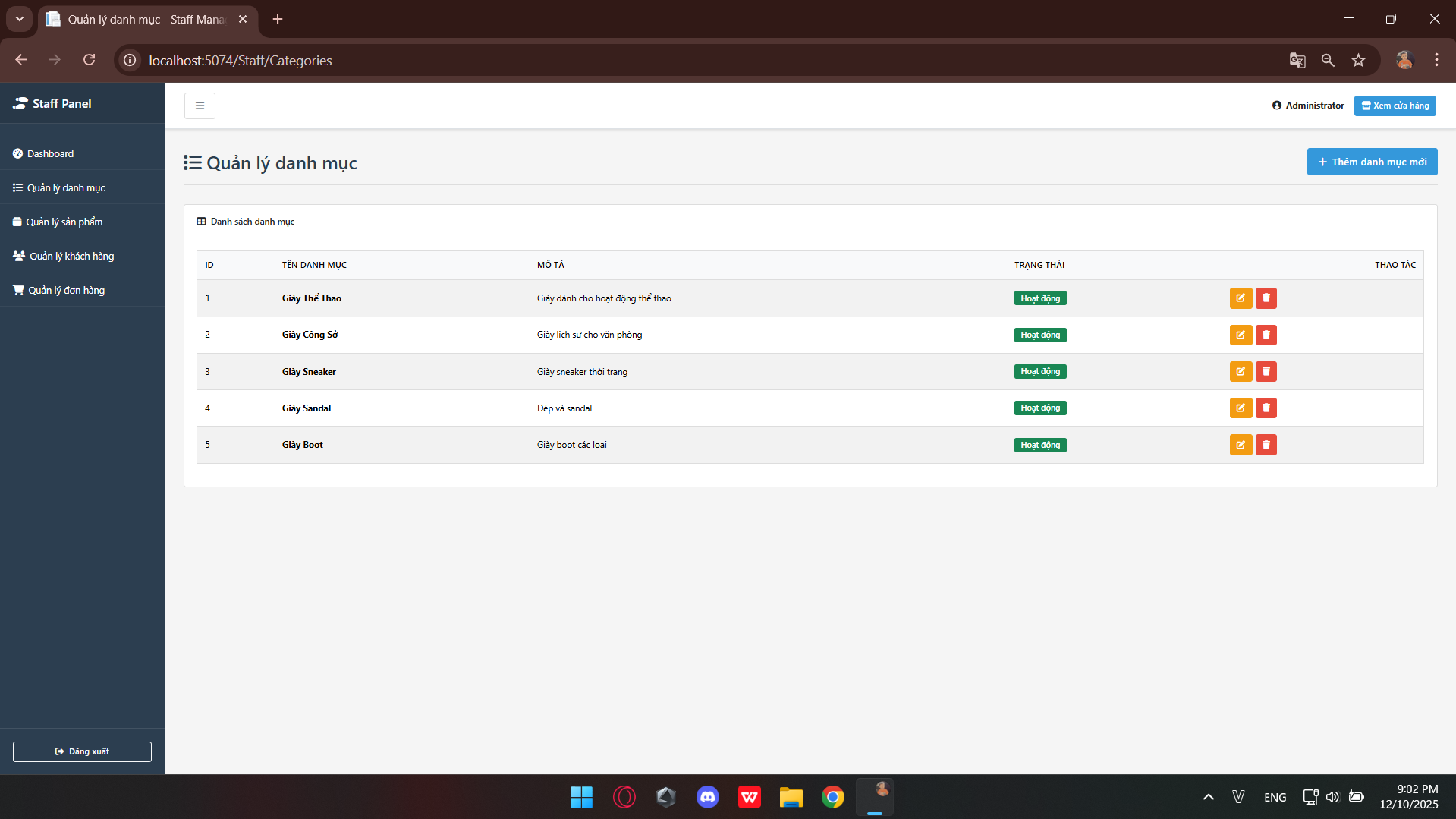
**Hình 4.25 – Quản lý biến thể:** Giao diện thiết lập biến thể cho sản phẩm.

Chức năng xóa (OnPostDeleteAsync) tích hợp kiểm tra ràng buộc toàn vẹn dữ liệu. Nếu sản phẩm vẫn còn biến thể tồn tại (\_unitOfWork.ProductVariants.FindAsync), hệ thống sẽ chặn xóa và hiển thị thông báo lỗi qua TempData.

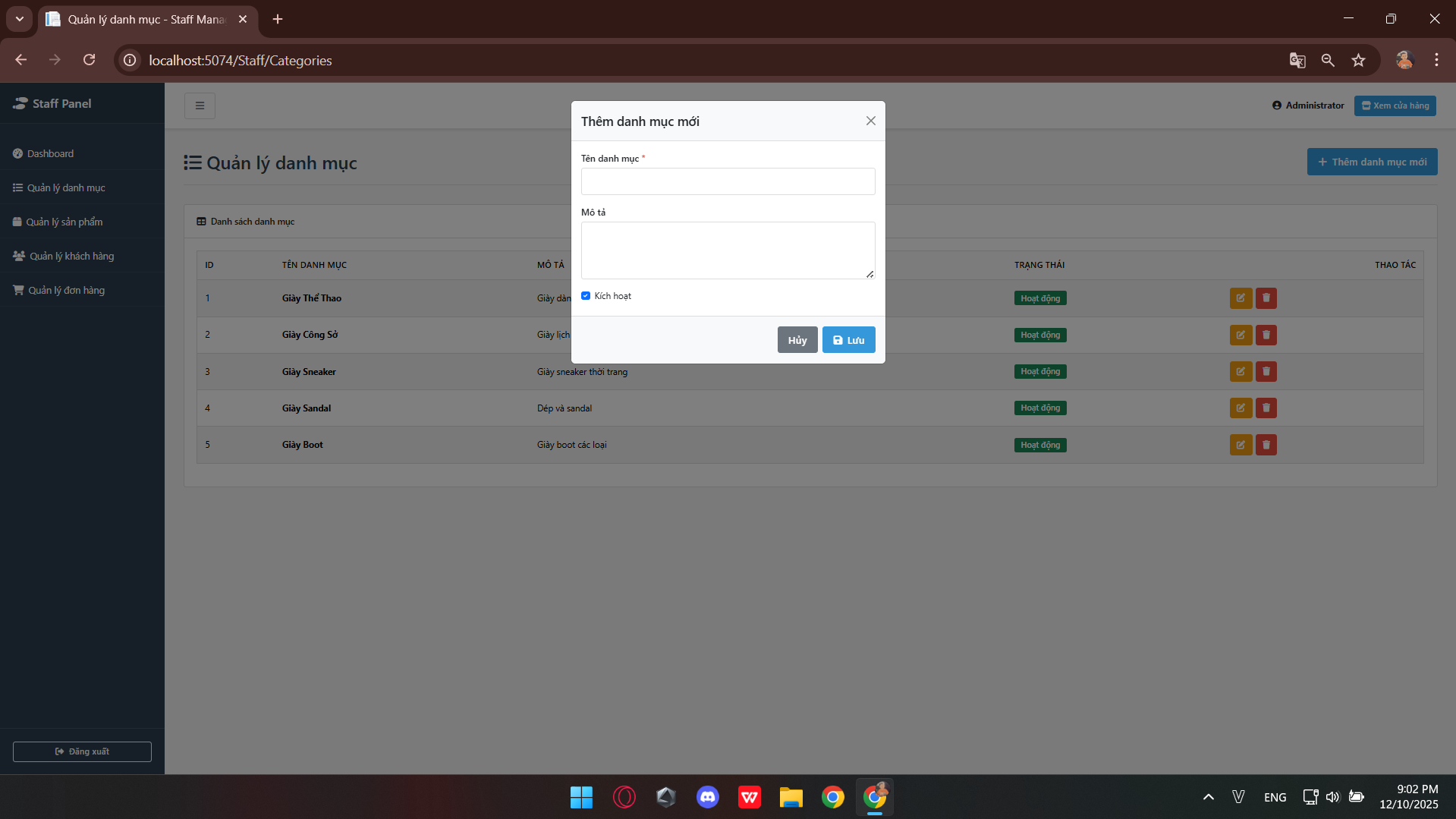


**Hình 4.26 – Xóa sản phẩm:** Thông báo xác nhận xóa sản phẩm có ràng buộc.

Tương tự, trang Pages/Staff/Categories.cshtml quản lý danh mục sản phẩm (\_unitOfWork.Categories.GetAllAsync). Form CRUD cho phép cập nhật tên, slug và mô tả. Hệ thống chặn xóa danh mục nếu đang có sản phẩm thuộc danh mục đó.

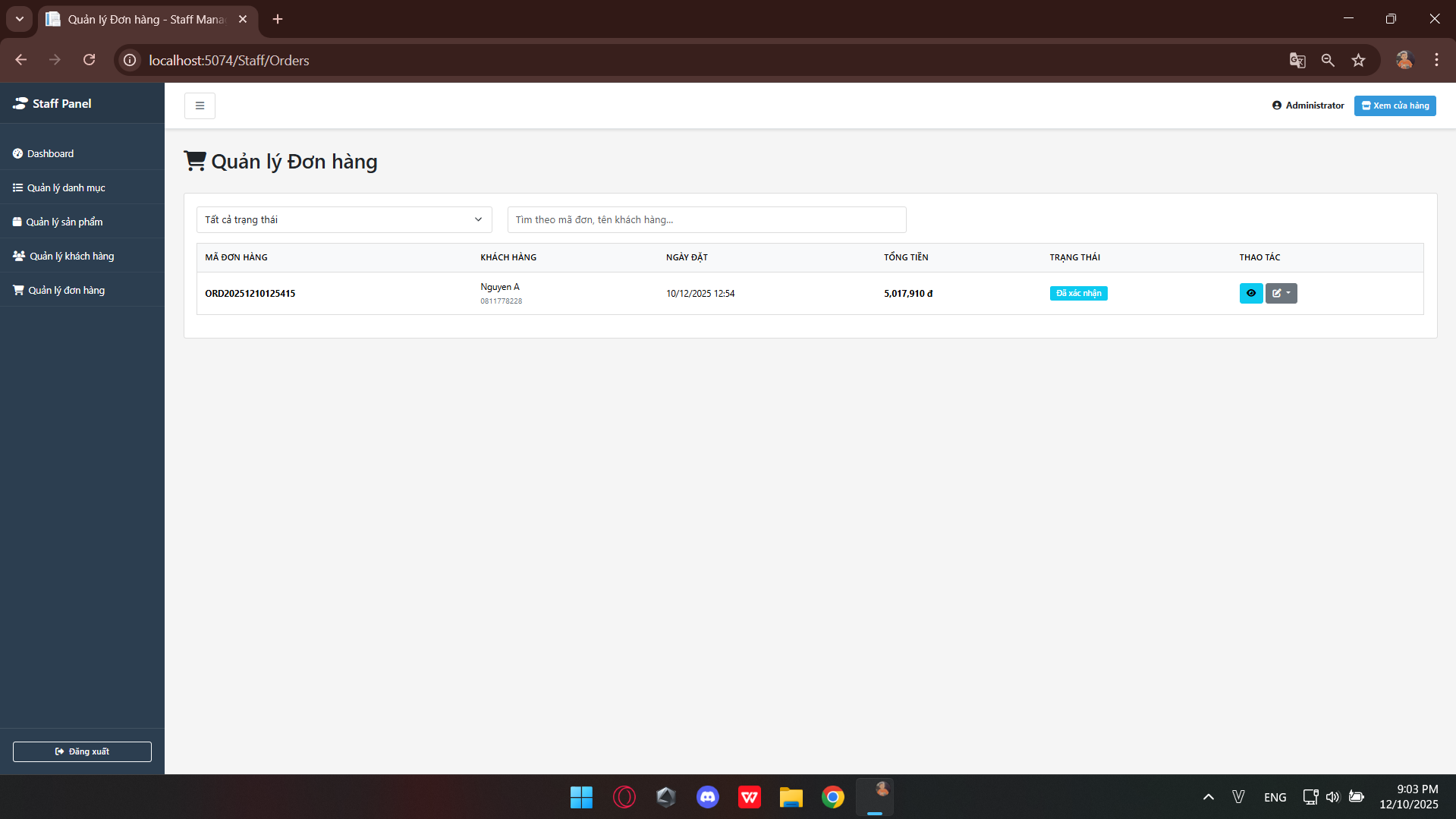


**Hình 4.27 – Bảng danh mục:** Danh sách quản lý danh mục.



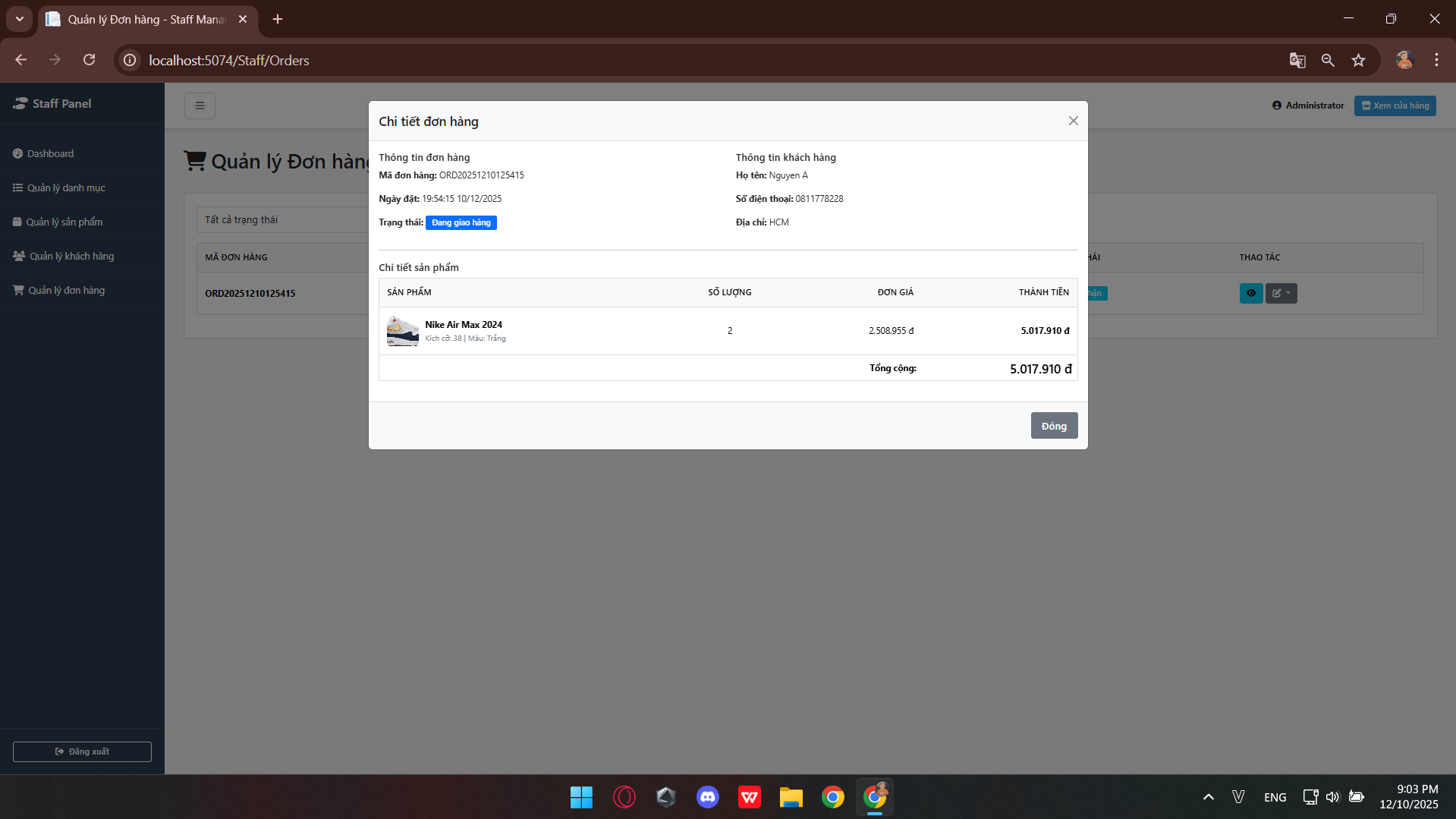
**Hình 4.28 – Form danh mục:** Modal thêm/sửa danh mục.

Trang Pages/Staff/Orders.cshtml hiển thị toàn bộ đơn hàng của hệ thống, thực hiện join với bảng Users để hiển thị tên khách hàng. Danh sách được sắp xếp mặc định theo đơn mới nhất để nhân viên dễ dàng xử lý.



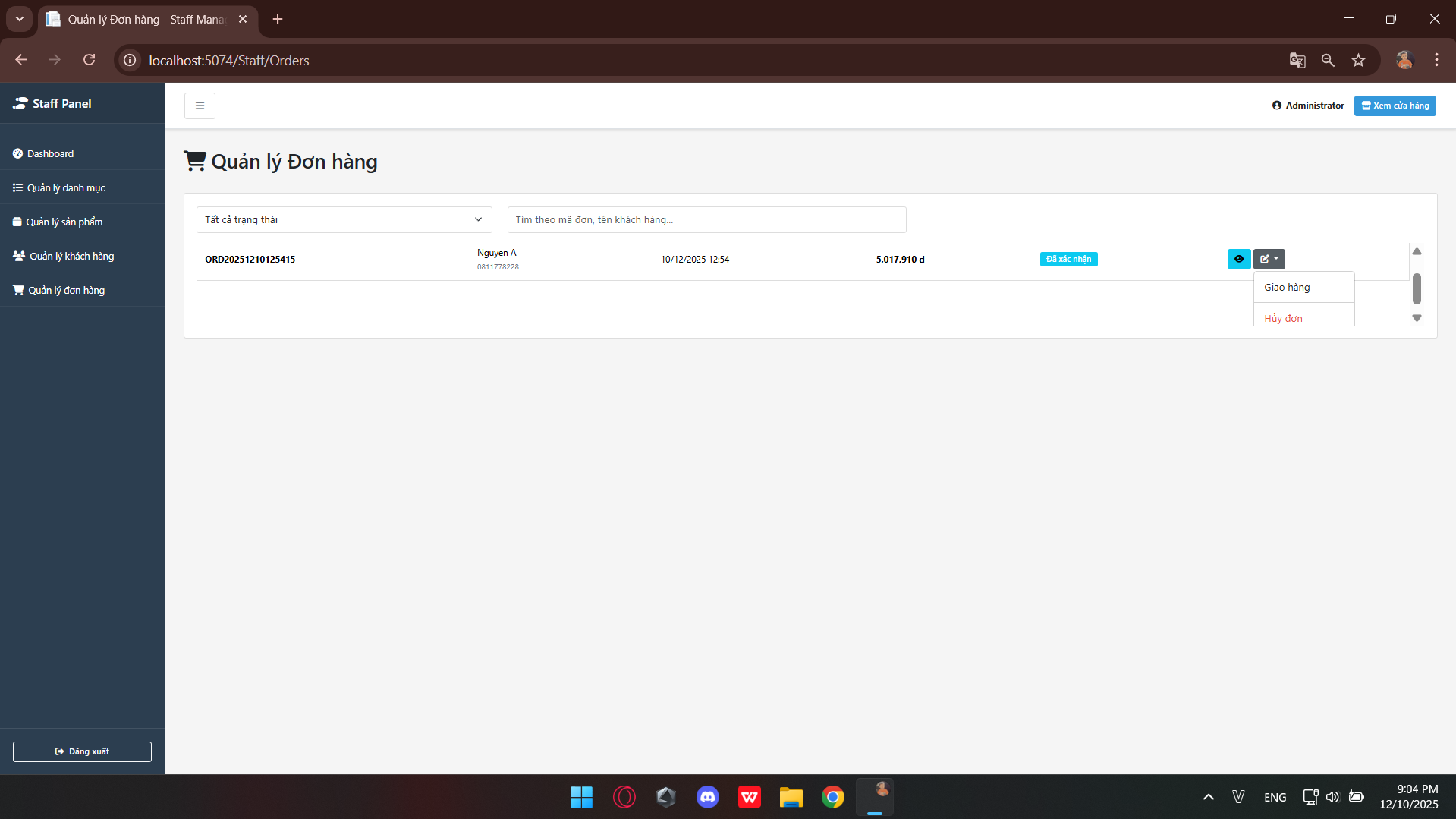
**Hình 4.29 – Danh sách đơn Staff:** Giao diện quản lý đơn hàng phía Admin.

Khi nhân viên nhấn “Xem”, handler OnGetGetOrderAsync trả về dữ liệu JSON chứa chi tiết items (tên, màu, size, giá, ghi chú) để hiển thị lên Modal mà không cần tải lại trang.



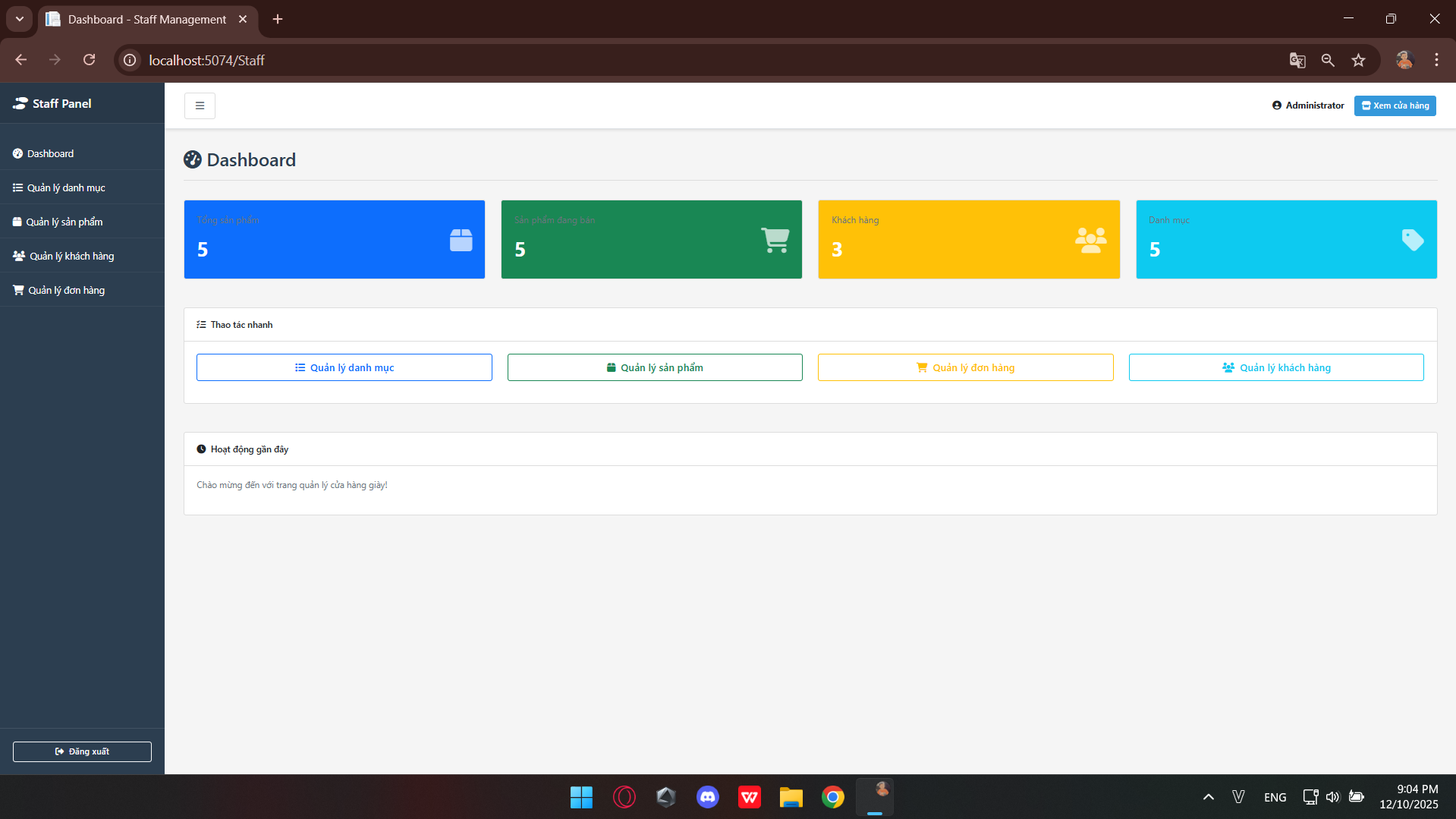
**Hình 4.30 – Chi tiết đơn Staff:** Modal xem chi tiết đơn hàng.

Tại đây, nhân viên cập nhật trạng thái đơn hàng thông qua OnPostUpdateStatusAsync. Phương thức này kiểm tra quy trình chuyển trạng thái hợp lệ (ví dụ: Pending → Processing → Shipping → Delivered) trước khi lưu xuống cơ sở dữ liệu.



**Hình 4.31 – Cập nhật trạng thái:** Dropdown cập nhật trạng thái đơn hàng.

Trang Dashboard (Pages/Staff/Index.cshtml) tổng hợp các chỉ số kinh doanh quan trọng. Các thẻ thống kê (Stats Cards) hiển thị số lượng sản phẩm, danh mục, khách hàng và số bản ghi đang kích hoạt. Dữ liệu được tổng hợp từ IProductService, ICategoryService, và IUserService.



**Hình 4.32 – Dashboard:** Giao diện tổng quan với các thẻ chỉ số.

Khu vực biểu đồ (Charts) bind dữ liệu từ IndexModel, hiển thị trực quan doanh thu hoặc tỷ lệ trạng thái đơn hàng, đảm bảo tính đồng bộ với dữ liệu thống kê.

## 4.3. Đánh giá tổng quan

Hệ thống ShoesShopWeb đã hoàn thiện các chức năng cốt lõi dựa trên công nghệ Razor Pages và kiến trúc phân lớp. Việc áp dụng AJAX trong giỏ hàng và Modal trong trang quản trị giúp cải thiện đáng kể trải nghiệm người dùng (UX). Các quy trình nghiệp vụ quan trọng như đặt hàng, kiểm soát tồn kho và phân quyền truy cập được xử lý chặt chẽ, đảm bảo tính đúng đắn của dữ liệu. Giao diện Responsive hoạt động tốt trên các thiết bị, đáp ứng yêu cầu đề ra của đồ án.

# CHƯƠNG 5: KẾT LUẬN

**5.1. Kết luận**

Đề tài "Xây dựng Website Thương mại Điện tử Bán Trà Sữa Trực Tuyến Sử Dụng ASP.NET Core Razor Pages" đã hoàn thành các mục tiêu đề ra với tỷ lệ 89.5% chức năng được triển khai thành công. Hệ thống MilkTeaWebsite đã chứng minh tính khả thi của việc áp dụng kiến trúc phân lớp trên nền tảng Razor Pages để giải quyết bài toán quản lý và bán hàng cho cửa hàng trà sữa quy mô vừa và nhỏ.

**5.1.1. Đánh giá mục tiêu đã đạt được**

Hệ thống đã triển khai đầy đủ 100% chức năng Must-have (13/13 chức năng) và Should-have (4/4 chức năng), bao gồm:

- Module khách hàng: Đăng ký/đăng nhập, duyệt sản phẩm với bộ lọc danh mục và tìm kiếm, xem chi tiết sản phẩm với cấu hình size/topping động, quản lý giỏ hàng đầy đủ, thanh toán và tạo đơn hàng, xem lịch sử đơn hàng.

- Module nhân viên: Dashboard thống kê số liệu đơn hàng theo trạng thái, quản lý đơn hàng với chức năng lọc/tìm kiếm/cập nhật trạng thái.

- Module quản trị: CRUD sản phẩm (bao gồm quản lý 3 mức giá theo size), CRUD danh mục và topping, phân quyền truy cập.

Về mặt kỹ thuật:

- Kiến trúc phân lớp: Thành công trong việc tách biệt Presentation (Razor Pages), Business Logic (BLL Services), và Data Access (DAL Repositories) với Dependency Injection, tạo nền tảng dễ bảo trì và mở rộng.

- Mô hình định giá động: Triển khai thành công cơ chế tính giá hai tầng (client-side JavaScript + server-side recalculation) đạt độ chính xác 100% (50/50 test cases passed), loại bỏ hoàn toàn sai lệch giá do nhập liệu thủ công.

- Entity Framework Core & Migrations: Quản lý schema database với 1 migration chính (InitialCreate), sử dụng Fluent API cho relationship phức tạp (many-to-many Product-Topping), áp dụng AsNoTracking tối ưu read-only queries.

- Snapshot pricing mechanism: Lưu giá tại thời điểm đặt hàng (BasePrice, ToppingPrice, UnitPrice) vào OrderDetail, đảm bảo đơn hàng cũ không bị ảnh hưởng khi giá sản phẩm/topping thay đổi sau này.

Về giải quyết bài toán thực tế:

- Giảm 100% sai lệch giá nhờ tự động hóa tính toán.

- Tập trung lưu trữ 100% đơn hàng với theo dõi trạng thái rõ ràng (Pending, Processing, Completed/Cancelled).

- Cung cấp dashboard cơ bản cho nhân viên và quản trị viên phục vụ ra quyết định.

- Tăng tốc độ xử lý đơn hàng từ ~2 phút (thủ công) xuống ~30 giây (hệ thống).

**5.1.2. Đóng góp của đề tài**

Đóng góp học thuật:

- Cung cấp một case study điển hình về áp dụng Razor Pages kết hợp Repository/Unit of Work Pattern trong domain F&B với sản phẩm đa biến thể, làm tài liệu tham khảo cho các nghiên cứu về lựa chọn kiến trúc web .NET (Razor Pages vs MVC vs SPA).

- Minh họa cụ thể cơ chế snapshot pricing để giải quyết vấn đề thay đổi giá động trong thương mại điện tử, có thể áp dụng cho các domain tương tự (đặt vé, đặt phòng khách sạn).

Đóng góp thực tiễn:

- Cung cấp giải pháp phần mềm chi phí hợp lý (open-source stack: .NET + PostgreSQL) cho nhóm SME ngành F&B, giảm phụ thuộc vào nền tảng bên thứ ba (Shopee Food, Grab Food) với phí hoa hồng cao (20-30%).

- Tạo nền tảng tích lũy dữ liệu khách hàng (lịch sử đơn, sản phẩm yêu thích) phục vụ phân tích và chiến lược marketing độc lập.

- Thiết kế kiến trúc modular cho phép dễ dàng tích hợp các module mở rộng (Payment Gateway, Recommendation Engine, Mobile App) trong tương lai.

**5.1.3. Hạn chế của đề tài**

Chức năng chưa triển khai:

- Chưa tích hợp cổng thanh toán thực tế (VNPay, MoMo) do giới hạn thời gian và yêu cầu doanh nghiệp thẩm định.

- Chưa có recommendation engine gợi ý topping/sản phẩm dựa trên lịch sử mua.

- Chưa có tính năng xuất báo cáo PDF/Excel chi tiết.

Kiểm thử chưa toàn diện:

- Chưa có unit test tự động cho BLL/DAL layer (hiện tại chỉ functional test thủ công).

- Chưa kiểm thử hiệu năng với số lượng lớn (1000+ sản phẩm, 100+ concurrent users).

- Chưa kiểm thử bảo mật chuyên sâu (penetration testing, OWASP Top 10).

Giao diện người dùng:

- UI/UX sử dụng Bootstrap mặc định, chưa được polish bởi designer chuyên nghiệp.

- Chưa có accessibility features (ARIA labels, keyboard navigation) cho người khuyết tật.

Phân quyền:

- Chưa phân tách rõ ràng giữa Staff và Admin (hiện tại Staff có thể thực hiện hầu hết thao tác quản trị).

- Chưa có audit log chi tiết cho các thao tác nhạy cảm (xóa sản phẩm, thay đổi giá).

**5.2. Hướng phát triển**

**5.2.1. Mở rộng chức năng ngắn hạn (1-3 tháng)**

Tích hợp cổng thanh toán điện tử:

- VNPay/MoMo: Triển khai IPaymentService với VNPayPaymentService và MoMoPaymentService.

- Webhook handling: Xử lý callback từ payment gateway để tự động cập nhật trạng thái đơn hàng.

- Refund mechanism: Hỗ trợ hoàn tiền cho đơn bị hủy (nếu đã thanh toán online).

Nâng cấp Dashboard với biểu đồ thực tế:

- Thay placeholder chart bằng Chart.js hoặc ApexCharts.

- Biểu đồ doanh thu theo ngày/tuần/tháng, biểu đồ tỷ lệ đơn theo trạng thái (pie chart).

- Real-time updates với SignalR (tùy chọn).

Notification system:

- Email notification cho khách hàng khi đơn hàng thay đổi trạng thái (sử dụng SendGrid/SMTP).

- In-app notification cho Staff khi có đơn mới (SignalR).

**5.2.2. Tối ưu hóa và bảo mật (2-4 tháng)**

Caching layer:

- Implement Redis distributed cache cho Product List, Category (ít thay đổi).

- Cache invalidation strategy khi Admin cập nhật sản phẩm/giá.

- Lợi ích: Giảm 40-60% load trên database cho read-heavy endpoints.

Unit & Integration tests:

- Viết unit tests cho BLL services (CartService, OrderService, ProductService) với mock repositories.

- Integration tests cho critical flows (đặt hàng end-to-end) với in-memory database.

- Target coverage: ≥ 70% cho BLL layer.

Security enhancements:

- Rate limiting cho API/Pages (sử dụng AspNetCoreRateLimit).

- Implement HTTPS strict transport security (HSTS).

- Content Security Policy (CSP) headers.

- Audit log cho Admin actions (sử dụng Serilog hoặc NLog).

Phân quyền chi tiết hơn:

- Tách rõ Staff và Admin roles.

- Permission-based authorization thay vì role-based (ví dụ: CanEditProduct, CanDeleteProduct).

**5.2.3. Mở rộng nền tảng (4-6 tháng)**

REST API layer cho Mobile App:

- Tạo Web API controllers expose BLL services (ProductController, OrderController, CartController).

- JWT authentication cho mobile clients.

- API documentation với Swagger/OpenAPI.

- \*\*Ước tính API:\*\* 1 tuần; \*\*Mobile app (Flutter/React Native):\*\* 4-6 tuần.

- \*\*Lợi ích:\*\* Tiếp cận khách hàng mobile (chiếm 70%+ traffic F&B).

Recommendation Engine:

- Collaborative filtering: "Khách mua sản phẩm A thường mua kèm topping B".

- Content-based: Gợi ý sản phẩm cùng danh mục.

- Sử dụng ML.NET hoặc Python microservice (FastAPI + scikit-learn).

Loyalty & Promotion system:

- Tích điểm cho mỗi đơn hàng, đổi điểm lấy voucher.

- Quản lý mã giảm giá (discount codes) với điều kiện áp dụng.

- Flash sale cho sản phẩm cụ thể.

**5.2.4. Tối ưu hiệu năng và mở rộng quy mô (6+ tháng)**

Database optimization:

- Partitioning cho bảng Order/OrderDetail theo tháng (nếu lượng đơn lớn).

- Read replicas cho PostgreSQL (master-slave replication).

Advanced analytics:

- Tích hợp Power BI hoặc Grafana cho dashboard nâng cao.

- Customer segmentation, RFM analysis (Recency, Frequency, Monetary).

**5.3. Khuyến nghị triển khai**

Cho đơn vị triển khai (cửa hàng trà sữa):

- Giai đoạn pilot (1-2 tháng đầu): Chạy song song hệ thống mới với quy trình cũ để huấn luyện nhân viên và thu thập phản hồi.

- Chuẩn bị dữ liệu: Chụp ảnh sản phẩm chất lượng cao, viết mô tả chi tiết, xác định giá chuẩn cho 3 size và topping.

- Đào tạo nhân viên: Tổ chức workshop 2-4 giờ về cách sử dụng Staff Dashboard, xử lý đơn, và troubleshooting cơ bản.

- Marketing: Khuyến mãi đặc biệt cho đơn đặt online trong tháng đầu (giảm 10-15%) để thúc đẩy adoption.

Cho nhà phát triển tiếp nối:

- Documentation: Bổ sung API documentation (Swagger), database schema diagram chi tiết, deployment guide.

- Code quality: Setup CI/CD pipeline (GitHub Actions/Azure DevOps) với automated testing và code coverage report.

- Monitoring: Implement Application Insights hoặc Sentry cho error tracking và performance monitoring.

- Backup strategy: Thiết lập automated daily backup cho PostgreSQL database với retention policy 30 ngày.

**5.4. Tổng kết**

Đề tài đã thành công trong việc xây dựng một nền tảng thương mại điện tử hoàn chỉnh cho cửa hàng trà sữa với kiến trúc phân lớp rõ ràng, mô hình định giá động chính xác, và trải nghiệm người dùng mượt mà. Hệ thống không chỉ giải quyết triệt để các vấn đề nghiệp vụ hiện tại (sai lệch giá, mất đơn hàng, thiếu dữ liệu) mà còn tạo nền tảng vững chắc cho các mở rộng tương lai.

Với 89.5% chức năng đã hoàn tất, 100% functional tests passed, và kiến trúc modular dễ bảo trì, MilkTeaWebsite đã chứng minh tính khả thi của việc áp dụng công nghệ .NET hiện đại cho nhóm SME ngành F&B với chi phí hợp lý. Các hướng phát triển đề xuất (Payment Gateway, Mobile App, Recommendation Engine) sẽ giúp hệ thống trở thành giải pháp toàn diện hơn, cạnh tranh được với các nền tảng bên thứ ba về cả tính năng lẫn trải nghiệm khách hàng.

Kết quả của đề tài không chỉ đóng góp về mặt học thuật (case study Razor Pages + Repository Pattern) mà còn có ý nghĩa thực tiễn cao, thúc đẩy quá trình chuyển đổi số cho cộng đồng doanh nghiệp vừa và nhỏ Việt Nam trong bối cảnh thương mại điện tử phát triển mạnh mẽ.