**BỘ GIÁO DỤC VÀ ĐÀO TẠO**

**TRƯỜNG ĐẠI HỌC MỞ THÀNH PHỐ HỒ CHÍ MINH**

**NGUYỄN THỊ THU UYÊN**

**XÂY DỰNG WEB BÁN QUẦN ÁO NỮ ONLINE TRACYSHOP**

**ĐỒ ÁN NGÀNH**

**NGÀNH CÔNG NGHỆ THÔNG TIN**

**TP. HỒ CHÍ MINH, 2021BỘ GIÁO DỤC VÀ ĐÀO TẠO**

**TRƯỜNG ĐẠI HỌC MỞ THÀNH PHỐ HỒ CHÍ MINH**

**NGUYỄN THỊ THU UYÊN**

**XÂY DỰNG WEB BÁN QUẦN ÁO NỮ ONLINE TRACYSHOP**

**Mã số sinh viên: 1851050182**

**ĐỒ ÁN NGÀNH**

**NGÀNH CÔNG NGHỆ THÔNG TIN**

**Giảng viên hướng dẫn: ThS. NGUYỄN THỊ PHƯƠNG TRANG**

**TP. HỒ CHÍ MINH, 2021**

**Ý KIẾN CHO PHÉP BẢO VỆ ĐỒ ÁN/ KHÓA LUẬN TỐT NGHIỆP**

**CỦA GIẢNG VIÊN HƯỚNG DẪN**

**Giảng viên hướng dẫn:**

**Sinh viên thực hiện: Lớp:**

**Ngày sinh: Nơi sinh:**

**Tên đề tài:**

**Ý kiến của giảng viên hướng dẫn về việc cho phép sinh viên được bảo vệ đồ án/ khóa luận trước Hội đồng:**

*Thành phố Hồ Chí Minh, ngày … tháng … năm ……*

Người nhận xét

LỜI CẢM ƠN

Trong quá trình làm đồ án ngành, em đã nhận được nhiều sự quan tâm, đóng góp ý kiến, chỉ dạy nhiệt tình từ quý thầy cô cũng như sự động viên từ gia đình, bạn bè.

Em xin chân thành cảm ơn quý thầy cô của trường Đại học Mở thành phố Hồ Chí Minh nói chung và quý thầy cô của khoa Công nghệ thông tin nói riêng đã tận tình giảng dạy, chia sẻ nhiều kinh nghiệm quý báu, giúp em có được những kiến thức và nền tảng vững chắc để làm được đồ án ngành này.

Đặc biệt, em xin gửi lời cảm ơn chân thành đến **ThS.** **Nguyễn Thị Phương Trang**, giảng viên khoa Công nghệ thông tin, trường Đại học Mở thành phố Hồ Chí Minh đã tận tình hướng dẫn, giúp đỡ và chỉ dạy cho em trong suốt quá trình làm đồ án ngành, cô luôn đưa ra những ý kiến giúp em có thể hoàn thành đồ án ngành một cách tốt nhất.

Cuối cùng, em xin gửi lời cảm ơn đến gia đình, bạn bè, những người đã luôn động viên, quan tâm, giúp đỡ em trong quá trình làm đồ án ngành này.

NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN

TÓM TẮT KHÓA LUẬN

Kinh doanh online hay bán hàng online không phải là một cái gì đó xa lạ với nhiều người, nhất là ở thời đại mạng Internet đang ngày càng phát triển. Nhiều doanh nghiệp, cửa hàng lớn và nhỏ luôn chọn hình thức kinh doanh này song song với việc bán hàng trực tiếp. Bên cạnh đó, người mua cũng rất ưa chuộng hình thức mua sắm online này vì sự tiện lợi của nó mang lại. Bắt kịp với xu hướng và nhu cầu của khách hàng hiện nay, trong đề tài này, em xây dựng một trang web bán quần áo nữ online cho phép người mua có thể đặt hàng, mua hàng và có thể thanh toán trực tiếp thông qua cổng thanh toán điện tử. Tuy nhiên, cùng với việc tiện lợi của mua bán online thì nó cũng vướng phải một số vấn đề như sau: bảo mật thông tin người dùng, hiệu suất ứng dụng, chi phí bộ nhớ, vấn đề đa nền tảng. Với ASP.NET Core thì những vấn đề trên đều được giải quyết. Do đó, em có ý tưởng sử dụng công nghệ ASP.NET Core MVC để xây dựng web bán quần áo nữ online**.** Ngoài ra, trong đề tài, em có tích hợp thêm chức năng gửi mail để xác thực tài khoản email của khách hàng nhằm tăng tính bảo mật thông tin người dùng.

**MỤC LỤC**

[LỜI CẢM ƠN 2](#_Toc85213394)

[NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN 3](#_Toc85213395)

[TÓM TẮT KHÓA LUẬN 4](#_Toc85213396)

[DANH MỤC TỪ VIẾT TẮT 7](#_Toc85213397)

[DANH MỤC HÌNH VẼ 8](#_Toc85213398)

[DANH MỤC BẢNG 9](#_Toc85213399)

[MỞ ĐẦU 10](#_Toc85213400)

[Chương 1. TỔNG QUAN ĐỀ TÀI 12](#_Toc85213401)

[1.1. Lý do chọn đề tài 12](#_Toc85213402)

[1.2. Bố cục đề tài 12](#_Toc85213403)

[Chương 2. CƠ SỞ LÝ THUYẾT 13](#_Toc85213404)

[2.1. Giới thiệu về .NET Core 13](#_Toc85213405)

[2.1.1. .NET Core là gì? 13](#_Toc85213406)

[2.1.2. Môi trường thực thi 13](#_Toc85213407)

[2.2. Giới thiệu về ASP.NET Core 13](#_Toc85213408)

[2.2.1. ASP.NET Core là gì? 13](#_Toc85213409)

[2.2.2. Ưu điểm 14](#_Toc85213410)

[2.3. Tổng quan về ASP.NET Core MVC 14](#_Toc85213411)

[2.3.1. Mô hình MVC là gì? 14](#_Toc85213412)

[2.3.2. ASP.NET Core MVC là gì? 15](#_Toc85213413)

[2.3.3. Routing 15](#_Toc85213414)

[2.3.4. Model binding 17](#_Toc85213415)

[2.3.5. Model validation 17](#_Toc85213416)

[2.3.6. Dependency Injection (DI) 20](#_Toc85213417)

[2.3.7. Tag Helpers 24](#_Toc85213418)

[2.3.8. Migration 26](#_Toc85213419)

[2.4. Tìm hiểu về ASP.NET Core Identity 28](#_Toc85213420)

[2.4.1. ASP.NET Core Identity là gì? 28](#_Toc85213421)

[2.4.2. Xác thực người dùng 31](#_Toc85213422)

[2.4.3. Phân quyền người dùng 32](#_Toc85213423)

[2.4.4. Xác thực từ bên thứ ba 33](#_Toc85213424)

[2.4.5. Các cấu hình dịch vụ trong Identity 40](#_Toc85213425)

[2.5. Kiểm tra gửi mail bằng Mailtrap 43](#_Toc85213426)

[2.5.1. Giới thiệu về Mailtrap 43](#_Toc85213427)

[2.5.2. Mailtrap là gì? 43](#_Toc85213428)

[2.5.3. Cấu hình dịch vụ gửi mail qua Mailtrap 45](#_Toc85213429)

[2.5.4. Thiết lập gửi mail qua Mailtrap 47](#_Toc85213430)

[Chương 3. WEBSITE BÁN QUẦN ÁO NỮ ONLINE TRACYSHOP 51](#_Toc85213431)

[3.1. Giới thiệu đề tài 51](#_Toc85213432)

[3.2. Phân tích và thiết kế hệ thống 52](#_Toc85213433)

[3.2.1. Đặc tả usecase 52](#_Toc85213434)

[3.2.2. Sơ đồ tuần tự 71](#_Toc85213435)

[3.3. Thiết kế cơ sở dữ liệu 71](#_Toc85213436)

[3.4. Thực nghiệm hệ thống 71](#_Toc85213437)

[Chương 4. KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN ĐỀ TÀI 72](#_Toc85213438)

[4.1. Kết luận 72](#_Toc85213439)

[4.2. Hướng phát triển đề tài 72](#_Toc85213440)

[TÀI LIỆU THAM KHẢO 73](#_Toc85213441)

[PHỤ LỤC 74](#_Toc85213442)

DANH MỤC TỪ VIẾT TẮT

DI Dependency Injection

DANH MỤC HÌNH VẼ

[Hình 2.1: Các thành phần trong mô hình MVC và sự tương tác giữa chúng (nguồn: [4]) 14](#_Toc84986786)

[Hình 2.2: Mô hình MVC trong dự án ASP.NET Core MVC 15](#_Toc84986787)

[Hình 2.3: Hiển thị lỗi cho người dùng khi sử dụng asp-validation-summary 19](#_Toc84986788)

[Hình 2.4: Hiển thị lỗi cho người dùng khi sử dụng asp-validation-for 20](#_Toc84986789)

[Hình 2.5: Thư mục Migrations trong dự án 26](#_Toc84986790)

[Hình 2.6: Giao diện của Manage NuGet Packages trong Visual Studio 2019 28](#_Toc84986791)

[Hình 2.7: Các Packages trong Dependencies 28](#_Toc84986792)

[Hình 2.8: Các bảng trong IdentityDbContext 30](#_Toc84986793)

[Hình 2.9: Trang trả về 404 khi Roles của người dùng không hợp lệ 32](#_Toc84986794)

[Hình 2.10: Giao diện của trang Google Cloud Platform 33](#_Toc84986795)

[Hình 2.11: Giao diện sau khi tạo xong ứng dụng trong Google Cloud Platform 33](#_Toc84986796)

[Hình 2.12: Thiết lập URIs 34](#_Toc84986797)

[Hình 2.13: ID Client và Secret Client của Google 34](#_Toc84986798)

[Hình 2.14: Màn hình đăng nhập qua Google 35](#_Toc84986799)

[Hình 2.15: Giao diện của trang Facebook for Developers 36](#_Toc84986800)

[Hình 2.16: Giao diện chọn loại ứng dụng trên Facebook for Developers 36](#_Toc84986801)

[Hình 2.17: Giao diện thêm chi tiết ứng dụng trong Facebook for Developers 37](#_Toc84986802)

[Hình 2.18: Nhập URL chuyển hướng trong Facebook for Developers 37](#_Toc84986803)

[Hình 2.19: Các thông tin cơ bản của ứng dụng trên Facebook for Developers 38](#_Toc84986804)

[Hình 2.20: Nhập mật khẩu Facebook để xem khóa bí mật của ứng dụng 38](#_Toc84986805)

[Hình 2.21: ID ứng dụng và khóa ứng dụng của Facebook 38](#_Toc84986806)

[Hình 2.22: Màn hình đăng nhập qua Facebook 39](#_Toc84986807)

[Hình 2.23: Trang chủ của Mailtrap 42](#_Toc84986808)

[Hình 2.24: Giao diện đăng ký của Mailtrap 43](#_Toc84986809)

[Hình 2.25: Giao diện hộp thư đến trên Mailtrap 43](#_Toc84986810)

[Hình 2.26: Các bản trả phí trên Mailtrap 44](#_Toc84986811)

[Hình 2.27: Chọn Integrations 44](#_Toc84986812)

[Hình 2.28: Cấu hình SMTP trong Mailtrap 45](#_Toc84986813)

[Hình 2.29: Giao diện nhận mail trong Mailtrap 49](#_Toc84986814)

DANH MỤC BẢNG

[Bảng 3.1: Đặc tả usecase đăng nhập 53](#_Toc85218798)

[Bảng 3.2: Đặc tả usecase đăng xuất 53](#_Toc85218799)

[Bảng 3.3: Đặc tả usecase đăng ký 54](#_Toc85218800)

[Bảng 3.4: Đặc tả usecase đặt hàng 55](#_Toc85218801)

[Bảng 3.5: Đặc tả usecase tra cứu 55](#_Toc85218802)

[Bảng 3.6: Đặc tả usecase phản hồi đánh giá 56](#_Toc85218803)

[Bảng 3.7: Đặc tả usecase quản lý tài khoản 57](#_Toc85218804)

[Bảng 3.8: Đặc tả usecase thanh toán 58](#_Toc85218805)

[Bảng 3.9: Đặc tả usecase quản lý giỏ hàng 60](#_Toc85218806)

[Bảng 3.10: Đặc tả usecase quên mật khẩu 61](#_Toc85218807)

[Bảng 3.11: Đặc tả usecase xem thông tin sản phẩm 61](#_Toc85218808)

[Bảng 3.12: Đặc tả usecase lịch sử mua hàng 62](#_Toc85218809)

[Bảng 3.13: Đặc tả usecase quản lý nhân viên 63](#_Toc85218810)

[Bảng 3.14: Đặc tả usecase quản lý sản phẩm 64](#_Toc85218811)

[Bảng 3.15: Đặc tả usecase quản lý khuyến mãi 65](#_Toc85218812)

[Bảng 3.16: Đặc tả usecase thống kê 66](#_Toc85218813)

[Bảng 3.17: Đặc tả usecase tìm kiếm của admin và nhân viên 67](#_Toc85218814)

[Bảng 3.18: Đặc tả usecase quản lý kho 68](#_Toc85218815)

[Bảng 3.19: Đặc tả usecase xem thông tin phản hồi 68](#_Toc85218816)

[Bảng 3.20: Đặc tả usecase quản lý khách hàng 69](#_Toc85218817)

[Bảng 3.21: Đặc tả usecase xem thông tin đặt hàng 70](#_Toc85218818)

MỞ ĐẦU

Trong điều kiện nền kinh tế ngày càng phát triển, đời sống của con người ngày một nâng cao thì các nhu cầu, dịch vụ của con người cũng ngày một tăng. Cùng với đó, mạng Internet cũng đang ngày một phát triển và phổ biến với người người nhà nhà, dẫn đến các hoạt động giải trí, sinh hoạt, học tập ít nhiều cũng có sự ảnh hưởng và thay đổi. Đi đôi với sự thay đổi đó, việc kinh doanh, mua bán của các doanh nghiệp, cửa hàng cũng có những thay đổi rõ rệt nhất định. Kể đến là sự đa dạng hóa về hình thức kinh doanh, mua bán khi có sự góp mặt của mạng Internet. Từ kinh doanh, mua bán trực tiếp theo kiểu truyền thống đến kinh doanh, mua bán online. Việc kinh doanh online giúp cho những doanh nghiệp, cửa hàng tiết kiệm được nhiều chi phí, phải kể đến là chi phí thuê mặt bằng. Đối với những cửa hàng kinh doanh vừa và nhỏ, kinh phí thấp thì đây là một lợi thế vô cùng lớn khi họ có thể tiết kiệm được một nguồn chi phí khá lớn cho việc thuê mặt bằng. Bên cạnh đó, việc kinh doanh online sẽ giúp nhiều khách hàng biết đến các doanh nghiệp, cửa hàng và việc tiếp cận với khách hàng ở nhiều nơi khác nhau sẽ trở nên dễ dàng hơn. Ngoài ra, kinh doanh online cũng không bó buộc về thời gian như kinh doanh trực tiếp theo kiểu truyền thống như trước đây. Thông thường, một cửa hàng sẽ mở cửa vào lúc 7 hoặc 8 giờ sáng và đóng cửa lúc 10 giờ tối. Nếu mua hàng trực tiếp tại cửa hàng thì khách hàng chỉ có thể đi đến cửa hàng trong khoảng thời gian này. Điều này khá bất tiện cho những ai bận rộn, không có thời gian mua sắm hoặc với những khách hàng ở xa. Nhưng với việc mua hàng online, khách hàng có thể mua hàng mọi lúc, mọi nơi mà không cần quan ngại về vấn đề thời gian hay khoảng cách địa lý, chỉ cần những thao tác đơn giản là bạn có thể đặt một hay nhiều món hàng mà bạn cần mua, đặc biệt với những ai có công việc bận rộn hoặc ở xa thì đây là một cách mua sắm vô cùng tiện lợi. Nhờ đó, các cửa hàng hoặc doanh nghiệp không những có thêm nguồn lợi nhuận ngoài giờ bán trực tiếp mà còn tăng thêm lượng khách hàng từ nhiều nơi khác nhau, giúp cho việc kinh doanh phát triển, doanh thu và lợi nhuận của các cửa hàng và doanh nghiệp được tăng cao. Đồng thời, việc kinh doanh online cũng bắt nhịp được tâm lý mua hàng của các khách hàng hiện nay khi nhiều khách hàng không muốn tốn quá nhiều thời gian cho việc mua sắm cũng như đi lại mà chỉ cần ngồi tại nhà lựa chọn những sản phẩm cần mua và với vài cú nhấp chuột hay nhấp chọn là có thể lên đơn hàng.

Ngày nay hình thức kinh doanh online đã không còn quá xa lạ, chúng ta có thể bắt gặp trên các nền tảng mạng xã hội như Facebook, Instagram, Zalo, TikTok,... Dù vậy, việc kinh doanh online qua website vẫn luôn được các cửa hàng và doanh nghiệp ưu ái hơn cả. Song song với sự tiện lợi cũng có một vài bất cập, mà đáng quan tâm nhất là tính bảo mật người dùng. Trước những nhu cầu đó, em đã thực hiện đề tài Xây dụng web bán quần áo nữ online của thương hiệu TracyShop với ngôn ngữ C#, công nghệ sử dụng là ASP.NET Core MVC, sử dụng hệ thống chứng thực phân quyền ASP.NET Core Identity, xác thực email người dùng đăng ký qua Mailtrap và sử dụng cơ sở dữ liệu SQL Server để lưu trữ.

# TỔNG QUAN ĐỀ TÀI

Chương này nhằm để trình bày tóm tắt lý do chọn đề tài và bố cục của đề tài, giúp mọi người có cái nhìn tổng quan hơn về đề tài.

## Lý do chọn đề tài

Ngày nay, chất lượng cuộc sống được nâng cao kéo theo đó là nhu cầu về cái đẹp của con người cũng ngày một thay đổi. Vẻ đẹp ấy không chỉ dừng lại ở nội tâm bên trong mà còn là vẻ đẹp về hình thức bên ngoài. Do đó, con người cũng dần quan tâm đến phong cách ăn mặc và vấn đề làm đẹp hơn. Đặc biệt đối với phái nữ, việc sắm cho mình những bộ cánh phù hợp với túi tiền nhưng lại không kém phần thời thượng luôn được hội chị em săn đón và lựa chọn. Việc khoác lên mình những bộ trang phục đẹp cũng phần nào giúp cho các bạn nữ thêm phần tự tin và tỏa sáng. Hiểu được các nhu cầu đó, các thương hiệu thời trang nữ đã dần mọc lên để đáp ứng nhu cầu của các khách hàng nữ. Với tiêu chí rẻ - đẹp - chất lượng, các thương hiệu thời trang đã cho ra đời những mẫu mã đa dạng, giá thành phải chăng nhưng chất lượng vẫn được đảm bảo. Vì vậy, em chọn đề tài Xây dựng web bán quần áo nữ online TracyShop để đưa các sản phẩm của các thương hiệu thời trang đến gần hơn với khách hàng và giúp cho việc mua sắm trở nên một cách dễ dàng mà không cần lo ngại về thời gian và khoảng cách địa lý.

## Bố cục đề tài

Nội dung đề tài được chia làm 4 chương:

* Chương 1: Tổng quan về đề tài: nêu lý do chọn đề tài và bố cục của đề tài.
* Chương 2: Cơ sở lý thuyết: trình bày các khái niệm cũng như các lý thuyết liên quan đến công nghệ được sử dụng trong đề tài.
* Chương 3: Website bán quần áo nữ online TracyShop: trình bày phân tích thiết kế của hệ thống và thực nghiệm website.
* Chương 4: Kết luận và hướng phát triển của đề tài.

# CƠ SỞ LÝ THUYẾT

Chương này nhằm trình bày các lý thuyết liên quan đến các công nghệ được sử dụng trong đề tài, giúp mọi người hiểu rõ hơn các kiến thức và nền tảng cơ bản để xây dựng nên đề tài.

## Giới thiệu về .NET Core

### .NET Core là gì?

.NET Core là là một phiên bản mới của .NET Framework. Đây là một nền tảng phát triển miễn phí, mã nguồn mở, được duy trì và phát triển bởi Microsoft và cộng đồng .NET ở trên Github. Nó là một khung nền tảng chéo, chạy trên hệ điều hành Windows, macOS và cả Linux [1]. Về cấu trúc, .NET Core được tổ chức theo hướng module hóa nên các ứng dụng chạy bằng .NET Core thường nhẹ, nhanh, dung lượng bộ nhớ thấp, hiệu suất ứng dụng được tối ưu hóa.

### Môi trường thực thi

CoreCLR được gọi là môi trường thực thi của .NET Core có thể chạy được trên nhiều hệ điều hành như Windows, macOS và Linux.

CoreCLR sử dụng trình biên dịch JIT compiler. Trình biên dịch này của CoreCLR có thể dịch được mã IL sang mã máy của ba nền tảng mà nó hỗ trợ là Windows, macOS và Linux [2].

## Giới thiệu về ASP.NET Core

### ASP.NET Core là gì?

ASP.NET Core là một framework miễn phí, mã nguồn mở và chạy đa nền tảng. Giống như .NET Core, nó có thể chạy được trên nhiều hệ điều hành như Windows, Linux, macOS, có cấu trúc theo module hóa với chi phí thấp. Nếu người dùng muốn tích hợp thêm các tính năng nâng cao khác thì chỉ cần thêm vào dưới dạng gói NuGet [3].

ASP.NET Core cũng được hỗ trợ và phát triển bởi Microsoft và cộng đồng trên Github.

### Ưu điểm

Vì ASP.NET Core có cấu trúc module hóa và chỉ cần thêm các gói NuGet khi cần thiết nên các ứng dụng sử dụng ASP.NET Core thường có hiệu suất cao, yêu cầu ít bộ nhớ hơn, kích thước triển khai cũng ít hơn và dễ bảo trì. Bên cạnh đó, với kích thước triển khai của ứng dụng ít hơn thì ứng dụng sẽ có tính bảo mật cao và chặt chẽ hơn.

## Tổng quan về ASP.NET Core MVC

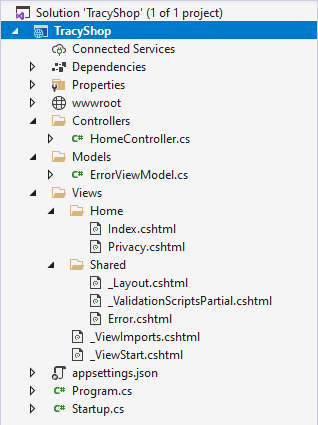
### Mô hình MVC là gì?

Mô hình MVC (Model – View – Controller) là mô hình kiến trúc gồm 3 phần: Model, View, Controller. Mỗi phần thực hiện một chức năng cụ thể. Với mô hình này, các yêu cầu của người dùng sẽ được chuyển đến Controller. Controller sẽ tiếp nhận yêu cầu của người dùng và tiến hành làm việc với Model để thực hiện những yêu cầu của người dùng và có thể lấy kết quả dữ liệu của các câu truy vấn. Sau đó, Controller sẽ chọn một View tương ứng trong các View để hiển thị ra cho người dùng và cung cấp cho nó những dữ liệu của Model mà nó yêu cầu [4].



Hình 2.1: Các thành phần trong mô hình MVC và sự tương tác giữa chúng (nguồn: [4])

Việc phân chia chức năng từng thành phần trong mô hình MVC giúp cho việc viết code, kiểm tra và sửa lỗi trở nên trực quan và dễ dàng hơn . Đồng thời, nó cũng giúp cho chương trình có tính mở rộng và dễ bảo trì.



Hình 2.2: Mô hình MVC trong dự án ASP.NET Core MVC

### ASP.NET Core MVC là gì?

ASP.NET Core MVC là một framework mã nguồn mở, nhẹ, có khả năng tối ưu hóa hiệu năng ứng dụng với ASP.NET Core. Nó cung cấp những tính năng dựa vào mô hình xây dựng các trang web động cho phép tách biệt các khối lệnh một cách rõ ràng [4].

### Routing

#### Routing là gì?

Routing là quá trình mà khi các URL request được gửi đến, ASP.NET Core sẽ tiến hành xem xét và “chỉ dẫn” cho các URL request đó đường đi đến Controller Action được chọn [5]. Ngoài ra, nó cũng được dùng để tạo một URL output.

#### Nhiệm vụ

Nhiệm vụ của Routing là ánh xạ request được gửi đến vào Controller Action và tạo một URL output tương ứng với Controller Action [5].

Một cách dễ hiểu, khi một request đến thông qua cơ chế Routing, nó sẽ chọn Controller và Action tương ứng để xử lý request đó. Nếu cần thiết, nó sẽ tạo ra một routing output. Nếu trường hợp trả về là một RedirectToAction, output sẽ là một redirect URL tương ứng để chuyển hướng đến URL đó.

#### Convention – based routing

Convention – based routing [4] là các routing cấu hình toàn cục các định dạng URL trong Configure của file cấu hình dự án Startup.cs. Mỗi URL được ánh xạ đến một Action cụ thể của một Controller nhất định. Khi request đến được nhận, routing engine sẽ phân tích URL và xem nó khớp với định dạng URL nào đã được xác định, sau đó gọi Action của Controller tương ứng.

app.UseEndpoints(endpoints =>

{

endpoints.MapControllerRoute(

name: "Admin",

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

);

endpoints.MapControllerRoute(

name: "default",

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

});

#### Attribute routing

Attribute routing [4] là các routing được tạo riêng cho từng Action và các routing này được đặt trước các Action tương ứng thông qua các attribute [Route].

[Route("/about", Name = "about")]

public IActionResult About()

{

return View();

}

### Model binding

#### Model binding là gì?

Model binding là quá trình truyền dữ liệu từ View sang Controller thông qua Http Request và đi vào các tham số của Action Method ở trong Controller. Các tham số này có thể có kiểu dữ liệu objects hoặc các kiểu dữ liệu phức tạp [6]. Đây là một ví dụ về việc chuyển dữ liệu của thuộc tính Id, Name từ View sang Controller bằng phương thức Post sử dụng [HttpPost] và [Bind(“Id,Name”)]:

[HttpPost]

public async Task<IActionResult> Create([Bind("Id,Name")] Category category)

{

if (ModelState.IsValid)

{

\_context.Add(category);

await \_context.SaveChangesAsync();

return RedirectToAction(nameof(Index));

}

return View(category);

}

Http Request có thể chứa các dữ liệu ở rất nhiều các định dạng khác nhau như route value, query string. Các dữ liệu này có thể được chứa trong Html Form.

#### Lấy dữ liệu từ Form Data trong Controller

Để lấy dữ liệu từ Form Data trong Controller, trước hết, ta cần có một class chứa các giá trị của dữ liệu được lấy từ form sau khi đã tạo hoặc chỉnh sửa. Khi người dùng nhấn vào button để gửi dữ liệu này đi thì dữ liệu này sẽ tự động được truyền đến lớp chứa các giá trị của dữ liệu trong form đó. Form sẽ gửi dữ liệu này đến Action xử lý tương ứng trong Controller.

### Model validation

#### Giới thiệu về Model validation

Đa số các dữ liệu được nhập từ người dùng đều tiềm ẩn nguy cơ không hợp lệ, do đó không thể thiết lập trực tiếp vào cơ sở dữ liệu. Các thông tin này có thể chứa những lỗi vô ý hoặc cố ý như sai lỗi chính tả hoặc lỗi cố ý làm sai lệch dữ liệu hay bất cứ lỗi gì của người dùng làm cho dữ liệu không được chuẩn mực. Để ngăn chặn việc này, ASP.NET Core cung cấp cơ chế Model validation và thành phần thực hiện cơ chế này được gọi là Model validator [7], nó giúp chúng ta kiểm tra sự hợp lệ của dữ liệu trước khi thiết lập xuống cơ sở dữ liệu.

Có hai mức kiểm tra trong Model validation:

* Client – side validation (mức client): Khi request được gửi lên server, Client – side validation sẽ chặn request ở mức trình duyệt, giúp dữ liệu được hợp lệ ngay trên trình duyệt mà không cần gửi đến server. Do đó, nó giúp tiết kiệm được băng thông và tăng sự trải nghiệm cho người dùng. Đồng thời, nó cũng giúp được server giảm được một phần công việc và giúp chúng ta tiết kiệm được thời gian.
* Server – side validation (mức server): Tuy Client – side validation giúp trải nghiệm người dùng tốt hơn nhưng nó không tin cậy. Trong một số trường hợp, có những lúc trình duyệt vô tình hay cố ý vô hiệu hóa các javascript validation, làm các dữ liệu đẩy lên server có thể không đúng. Để ngăn chặn việc dữ liệu đưa vào hệ thống không hợp lệ, ta cần tiến hành kiểm tra tính hợp lệ của dữ liệu ở mức server. Server – side validation đảm bảo được dữ liệu không hợp lệ sẽ không được đưa vào server. Do đó, nó có độ tin cậy cao và an toàn hơn. Tuy nhiên, việc kiểm tra dữ liệu ở Server – side validation sẽ làm giảm trải nghiệm của người dùng, tiêu tốn băng thông và tốn thời gian vì nó phải gửi request lên server.

Trong thực tế, người ta thường kết hợp Client – side validation và Server – side validation, tức là những dữ liệu không hợp lệ ở phía client bỏ qua thì đến server sẽ được kiểm tra tính hợp lệ một lần nữa. Qua đó ta có thể tận dụng được các ưu điểm và khắc phục các nhược điểm của đôi bên, làm cho hệ thống tối ưu hơn, vừa đảm bảo tính trải nghiệm của người dùng vừa đảm bảo dữ liệu đưa lên hệ thống được an toàn.

#### Validation Attributes

Trong ASP.NET Core có các Validation Attributes viết sẵn, ta chỉ cần đặt nó trước các thuộc tính của Model hoặc ViewModel để thiết lập các ràng buộc cho chúng. Các quy tắc này cũng được gọi là **DataAnnotations**, nó nằm trong namespace **System.ComponentModel.DataAnnotations**.

[Required(ErrorMessage = "Vui lòng điền số điện thoại")]

[Display(Name = "Số điện thoại")]

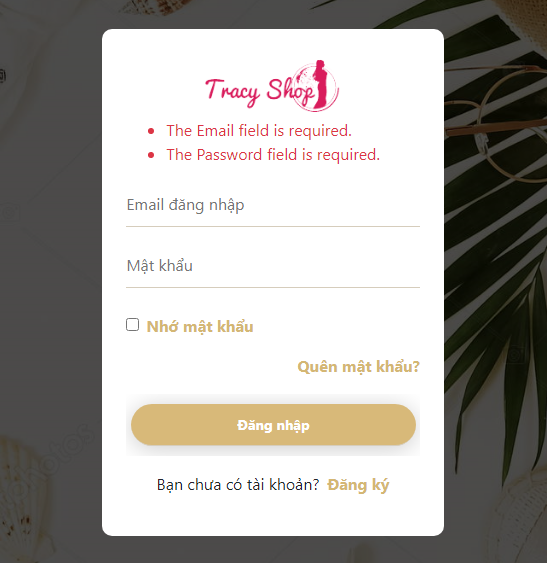
[Phone(ErrorMessage = "Vui lòng nhập số điện thoại hợp lệ")]

public string PhoneNumber { get; set; }

#### Validation Tag Helpers

Dùng Validation Tag Helper [7] để thông báo các lỗi nhập liệu cho người dùng trên View. Với asp-validation-summary sẽ hiển thị các thông báo lỗi của riêng Model validation. Các lỗi này sẽ được đặt trong thẻ div và hiển thị ở phía trên cùng của form.

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



Hình 2.3: Hiển thị lỗi cho người dùng khi sử dụng asp-validation-summary

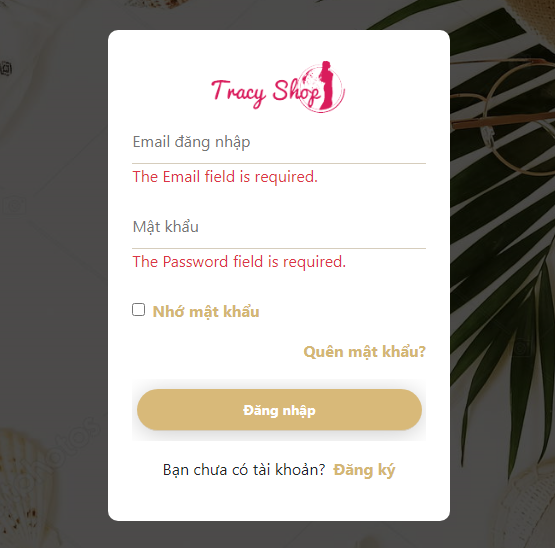
Với asp-validation-for sẽ hiển thị các lỗi của thuộc tính ở trên view. Các lỗi này được đặt trong thẻ span và hiển thị bên cạnh thuộc tính.

<div class="form-group form-input">

<input **asp-for**="Email" **type**="text" placeholder="Email đăng nhập"/>

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

</div>



Hình 2.4: Hiển thị lỗi cho người dùng khi sử dụng asp-validation-for

### Dependency Injection (DI)

#### Giới thiệu về Dependency Injection (DI)

Trong thực tế, một hệ thống không chỉ dừng lại với những chức năng đã được thiết lập lúc ban đầu khi hệ thống được tạo ra. Các chức năng của hệ thống luôn phải được cập nhật và thay đổi tùy theo nhu cầu của con người. Điển hình là các ứng dụng như Facebook, Google,... luôn đưa ra những bản cập nhật mới để cập nhật các tính năng và khắc phục sự cố nhằm tăng tính trải nghiệm và đáp ứng nhu cầu của người dùng trên thế giới. Do đó, việc bảo trì và mở rộng hệ thống luôn được các nhà lập trình quan tâm và họ luôn tìm mọi cách để giúp việc bảo trì và mở rộng code được hiệu quả.

Với một hệ thống, tùy vào tư duy lập trình của các lập trình viên lúc viết code nên các hàm trong hệ thống luôn tiềm ẩn rủi ro làm giảm tính bảo trì và mở rộng. Ví dụ một chức năng Duyệt đơn hàng của khách hàng được viết như sau:

public class Cart

{

public void CheckOrder(int orderId, int userId)

{

AppData db = new AppData();

db.Save(orderId);

Logger log = new Logger();

log.LogInfo("Order has been checked!");

EmailService es = new EmailService();

es.SendEmail(userId);

}

}

Nhìn chung, đoạn code trên có vẻ không có gì sai và việc viết code rất nhanh. Tuy nhiên, với cách làm này thì có thể sẽ dẫn đến một số vấn đề trong tương lai [8]:

* Hàm CheckOrder này rất khó trong việc kiểm tra vì nó liên quan đến hai module là AppData và EmailService.
* Nếu thay đổi AppData hoặc EmailService hoặc cả hai thì có thể phải sửa lại tất cả các code khởi tạo và gọi hai module này trong CheckOrder. Điều đó gây mất thời gian và rất dễ xảy ra lỗi.
* Về thời gian lâu dài, code sẽ có tính kết dính. Những module có tính kết dính cao, khi thay đổi một module bất kì sẽ dẫn đến hàng loạt các thay đổi khác. Điều này rất bất lợi cho việc bảo trì và mở rộng.

Do đó, việc Dependency Injection (DI) ra đời đã khắc phục được các vấn đề trên. Nó giúp linh hoạt hơn trong việc tổ chức code, dễ dàng kiểm tra và viết Unit Test và giúp cho việc bảo trì và mở rộng code dễ dàng hơn.

Hiện nay, Dependency Injection (DI) đã trở thành một phần trong ASP.NET Core.

#### Dependency Injection là gì?

Dependency Injection (viết tắt là DI) là một kĩ thuật trong đó một đối tượng có thể nhận được các đối tượng khác truyền vào nó khi nó cần đến. Đặc điểm chính của DI là làm giảm sự phụ thuộc giữa các đối tượng, giữa các module với nhau.

@inject TracyShop.Data.AppDbContext \_context

<!DOCTYPE html>

<html lang= "en">

<head>

<title>@\_context.User.Name</title>

<head>

<body>

<h1>@\_context.User.Name</h1>

<body>

<html>

#### DI Container

DI Container là một đối tượng có trách nhiệm tạo mới các dependencies (các phụ thuộc) và gán nó cho các đối tượng yêu cầu nó [9].

Để DI Container biết được các đối tượng nào được tạo, ta cần đăng ký các đối tượng đó trong phương thức **ConfigureServices** của lớp **Startup**.

services.AddScoped<ILoginRepository, LoginRepository>();

services.AddScoped<IUserService, UserService>();

services.AddScoped<IEmailService, EmailService>();

services.AddScoped<IUserClaimsPrincipalFactory<AppUser>, AppUserClaimsPrincipalFactory>();

services.AddScoped<IAvatarSevice, AvatarService>();

#### Vòng đời trong DI (DI lifetime)

Việc hiểu rõ vòng đời của những services trong DI là rất cần thiết trước khi tiến hành sử dụng chúng. Nếu chúng ta tạo services mà không hiểu được sự khác biệt giữa các mức vòng đời của nó thì có thể dẫn đến việc hệ thống của chúng ta hoạt động thất thường. Ngoài ra, khi một service gửi yêu cầu đến một service khác thông qua DI, việc biết liệu nó nhận được một đối tượng mới hay là một đối tượng đã được tạo là rất quan trọng [9]. Do đó, việc biết được vòng đời của các services trong việc đăng ký chúng là vô cùng cần thiết.

Ta quản lý vòng đời của services trong DI bằng 3 mức độ:

* Transient: một đối tượng mới luôn luôn được tạo khi có yêu cầu.
* Scoped: với Scoped sẽ bị giới hạn so với Transient, một đối tượng mới được tạo sẽ được dùng chung cho tất cả các scope với mỗi scope là một yêu cầu. Trong scope, service sẽ được tái sử dụng lại.
* Singleton: service chỉ được tạo duy nhất một lần.

Vậy vấn đề đặt ra ở đây là nên dùng Transient, Scoped hay Singleton?

Transient service là một cách an toàn nhất dùng để tạo service vì nó luôn tạo một đối tượng mới khi có yêu cầu. Do đó, việc sử dụng Transient service sẽ tốn nhiều tài nguyên và bộ nhớ. Nếu có quá nhiều đối tượng được tạo (hay có quá nhiều yêu cầu) thì sẽ làm giảm hiệu năng của hệ thống. Transient service phù hợp với những service nhỏ và nhẹ, không chứa các trạng thái.

Scoped service là một tùy chọn tốt hơn nếu ta muốn duy trì các trạng thái trong một yêu cầu.

Singleton service chỉ được tạo một lần và nó không hủy cho đến khi hệ thống tắt. Do đó, nó sẽ chiếm một phần bộ nhớ của hệ thống và sau mỗi yêu cầu, bộ nhớ của nó sẽ tăng dần lên cho đến khi bị đầy và hệ thống không hoạt động. Tuy nhiên, Singleton cũng giúp tiết kiệm bộ nhớ nếu trong trường hợp xử lý tốt vì nó có cơ chế sử dụng lại và dùng được ở nhiều nơi.

Tùy vào mục đích sử dụng và kích thước của của từng service đưa vào hệ thống mà ta có những sự lựa chọn khác nhau để hệ thống được tối ưu, tăng trải nghiệm của người dùng và tiết kiệm tài nguyên bộ nhớ.

#### Ưu điểm

* Giảm sự phụ thuộc giữa các đối tượng với nhau
* Code sạch và giúp dễ đọc hơn
* Hệ thống dễ bảo trì và mở rộng
* Dễ dàng cập nhật các triển khai mà không cần có nhiều sự thay đổi về code

### Tag Helpers

#### Tag Helper là gì?

Tag Helper nằm trong thư viện **Microsoft.AspNetCore.Mvc.TagHelppers** của ASP.NET Core, là một tính năng giúp thêm các đoạn code xử lý trên server dưới dạng giống với HTML, giúp nhìn thân thiện với code HTML hơn. Các code này được xử lý bởi Razor engine, tạo ra các đoạn code HTML và gen lên trên trình duyệt. Ví dụ, ta dùng Form Tag Helper với asp-action:

<form **asp-action**="ForgotPassword">

Khi gen lên trình duyệt sẽ chuyển Form Tag Helper thành thẻ <form> và asp-action thành action của <form> trên HTML:

<form **action**="ForgotPassword" menthod="post">

Nó cũng thao tác với các HTML elements để tạo ra các HTML elements mới. Chúng ta cũng có thể sử dụng thêm, sửa, xóa để tạo ra.

Để thêm Tag Helper vào một dự án, ta sử dụng @addTagHelper. Mặc định, khi tạo một dự án ASP.NET Core MVC thì nó sẽ tự động thêm các Tag Helper vào trong layout \_ViewImport.cshtml có sẵn của dự án.

@addTagHelper \*, Microsoft.AspNetCore.Mvc.TagHelpers

#### Form Tag Helper

Form Tag Helper sử dụng thẻ <form> để bao bọc, và sử dụng một số thuộc tính như asp-controller, asp-action, asp-area giúp tạo ra các HTML trên trình duyệt.

<form **asp-action**="CheckedOrder" **asp-controller**="Order" **asp-area**="Admin" method="post" **asp-route-id**="@item.Id">

Đoạn code trên sau khi đưa lên trình duyệt:

<form method="post" action="/admin/order/checkedorder/2">

<input name="\_\_RequestVerificationToken" type="hidden" value="CfDJ8BfCWMX6ePhCgmcIn5vf\_6pBUkL8utMuXX5QEfFpy58GtePHPJOI \_4bz0aeQlrHf4ejYr58ivUtaBy- W5tXjVMea\_8JGUy4sNtxwCi2M\_ronFv6TDS2gNz8L1rcOc8aTyzL1RxTFeKILcC MbSGzaSgZ93MxJV3YjY6fX4M9pbAStwvVJVnqH5S4kI2xw0ncR9w">

</form>

#### Label Tag Helper

Label Tag Helper sử dụng thẻ <label> để bao bọc và nó sử dụng thuộc tính asp-for giúp tạo các HTML trên trình duyệt.

<label **asp-for**="Name" class="control-label">Họ và tên</label>

Đoạn code trên sau khi đưa lên trình duyệt:

<label class="control-label" for="Name">Họ và tên</label>

#### Input Tag Helper

Input Tag Helper sử dụng thẻ <input> và nó tạo ra một thẻ <input> tương ứng trong HTML, thường nó dùng để chuyển dữ liệu của một thuộc tính trong Model. Tương tự label, nó cũng sử dụng asp-for để tạo các HTML trên trình duyệt.

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

Đoạn code trên sau khi đưa lên trình duyệt:

<input class="form-control" type="text" id="Name" name="Name" value="Uyên Nguyễn">

#### Lợi ích của Tag Helper

* Thân thiện với cú pháp HTML: Tag Helper có cú pháp giống với HTML giúp dễ dàng tương tác và sử dụng và cách sử dụng tương tự như HTML. Đồng thời, ta cũng có thể sử dụng thêm CSS để gán cho các Tag Helper như trong HTML để giao diện được đẹp hơn.
* Code sạch hơn: code dễ nhìn và sạch sẽ, không cần phải dùng kí tự @ để chuyển đổi code C# nên giống như ta đang viết HTML.
* Dễ dàng tạo View hiển thị trên trình duyệt với những Tag Helper có sẵn. Ngoài ra ta có thể tự tạo một Tag Helper nếu nó không có sẵn trong thư viện.

### Migration

Giống như những nền tảng khác như Django, Laravel, Ruby,... trong ASP.NET Core cũng hỗ trợ Migration trong mô hình Code First.

Migration là một kĩ thuật tương tác với cơ sở dữ liệu trong mô hình Code First. Nó giúp thay đổi cấu trúc cơ sở dữ liệu dưới cơ sở dữ liệu khi ta thay đổi cấu trúc cơ sở dữ liệu trong code ở Models, đảm bảo dữ liệu không bị mất và mỗi lần migrate nó sẽ lưu lại lịch sử cập nhật xuống cơ sở dữ liệu. Việc tạo và cập nhật các migrations được thực hiện trong Terminal.

Để migrate xuống cơ sở dữ liệu, ta thực hiện lệnh sau trong Terminal:

dotnet ef migrations Init

với Init là tên migrations, tên này ta tự đặt.

Sau khi migrations, trong cấu hình dự án sẽ xuất hiện một thư mục Migrations. Trong thư mục này sẽ sinh ra các file .cs.



Hình 2.5: Thư mục Migrations trong dự án

File AppDbContextModelSnapshot.cs sẽ chứa các cấu trúc dữ liệu của tại thời điểm ta tạo migrations đầu tiên. Mỗi lần migrations sẽ tự động tạo ra các file tương tự file 21211002071747\_Init.cs. Dãy số đầu tiên trong tên của mỗi file được sinh ra bằng cách random ngẫu nhiên khi ta tạo migrations và nó sẽ gán vào đầu của mỗi tên file. Những file này sẽ lưu các cập nhật tại các thời điểm ta thay đổi code trong Models.

Để biết được trong dự án có những migrations nào, ta sử dụng lệnh:

dotnet ef migrations list

Khi chạy lệnh này nó sẽ kiểm tra và hiển thị các danh sách tất cả migrations đã được tạo của dự án trong Terminal.

* Một số lệnh trong Migrations
* Lệnh xóa một migration cuối cùng được tạo:

dotnet ef migrations remove

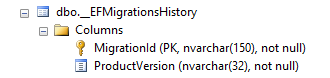
* Lệnh cập nhật tất cả các migrations xuống cơ sở dữ liệu:

dotnet ef database update

Nếu chỉ cập nhật một migration hoặc muốn quay về một migrations nào đó, ta chỉ cần chỉ tên của migration đó trong câu lệnh cập nhật, ví dụ với migration có tên Init:

dotnet ef database update Init

Sau khi cập nhật, nó sẽ lưu lại lịch sử cập nhật trong một bảng có tên là \_EFMigrationsHistory dưới cơ sở dữ liệu.



Hình 2.6: Các thuộc tính trong bảng \_EFMigrationsHistory

Bảng này gồm hai thuộc tính là MigrationId và ProductVersion với thuộc tính MigrationId lưu tên của các migrations và ProductVersion lưu thông tin version sử dụng của các migrations đó.



Hình 2.7: Dữ liệu trong bảng \_EFMigrationsHistory

* Lệnh hiển thị tất cả các câu truy vấn SQL mà nó thực hiện dưới cơ sở dữ liệu từ migration đầu đến migration cuối:

dotnet ef migrations script

* Lệnh lưu tất cả các truy vấn trong migrations ra file script SQL với tên file là MigrationScript:

dotnet ef migrations script –o MigrationScript.sql

* Lệnh xóa database:

dotnet ef database drop –f

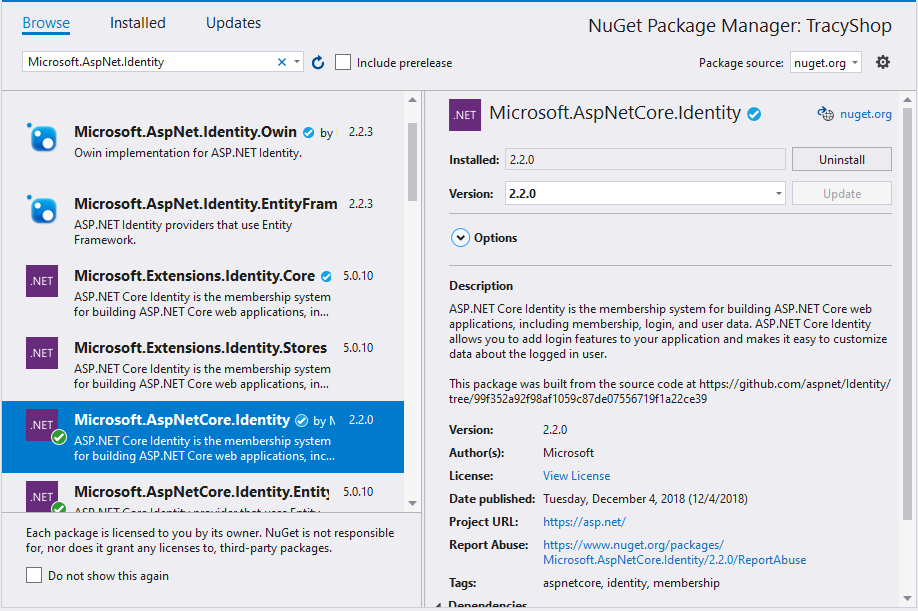
## Tìm hiểu về ASP.NET Core Identity

### ASP.NET Core Identity là gì?

ASP.NET Core Identity là một hệ thống chứng thực phân quyền mạnh mẽ trong ASP.NET Core. Nó giúp chứng thực và phân quyền người dùng và hỗ trợ các chức năng như Đăng ký, Đăng nhập, Đăng xuất trong Windows. Đồng thời, nó cũng hỗ trợ chức năng Đăng nhập từ bên thứ ba như Facebook, Google,...

Để sử dụng ASP.NET Core Identity, ta cần tích hợp các thư viện sau vào dự án bằng cách cài đặt nó trong Manage NuGet Packages:

* Microsoft.AspNetCore.Identity
* Microsoft.AspNetCore.Identity.UI
* Microsoft.AspNetCore.Identity.EntityFrameworkCore
* Microsoft.AspNetCore.Identity.Authentication



Hình 2.8: Giao diện của Manage NuGet Packages trong Visual Studio 2019

Hoặc ta có thể cài đặt các thư viện này trên Terminal bằng các lệnh sau:

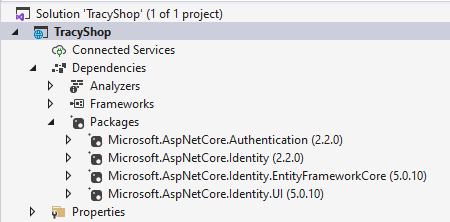
dotnet add package Microsoft.AspNetCore.Identity

dotnet add package Microsoft.AspNetCore.Identity.EntityFrameworkCore

dotnet add package Microsoft.AspNetCore.Identity.UI

dotnet add package Microsoft.AspNetCore.Authentication

Sau khi cài đặt hai thư viện này, nó sẽ tích hợp vào dự án. Ta kiểm tra trong Dependencies Packages của dự án:



Hình 2.9: Các Packages trong Dependencies

Tiếp đến, ta tạo một lớp AppUser trong Models kế thừa lớp IdentityUser của IdentityDbContext trong ASP.NET Core Identity. Lớp này có sẵn trong thư viện **Microsoft.AspNetCore.Identity** mà ta đã cài đặt lúc nãy và nó đã có sẵn có các thuộc tính của User như Id, Username, Password, Email, PhoneNumber,... Ta chỉ cần thêm một số thuộc tính nếu cần như Is\_active, Joined\_date,...

public class AppUser : IdentityUser

{

[MaxLength(100)]

public string Name { set; get; }

[MaxLength(255)]

public string Avatar { set; get; }

public DateTime? Birthday { set; get; }

[MaxLength(10)]

public string Gender { set; get; }

public bool Is\_active { set; get; } = true;

public DateTime Joined\_date { set; get; } = DateTime.Now;

}

Ngoài lớp IdentityUser, trong IdentityDbContext [10] còn sử dụng các lớp như IdentityRole, IdentityUserRole, IdentityUserClaim, IdentityUserLogin, IdentityUserToken, IdentityRoleClaim. Các lớp này sẽ có trong DbContext của dự án.

**DbContext** là nơi trung gian, giúp kết nối và tương tác giữa các lớp Models và các bảng dưới cơ sở dữ liệu. Do đó, nó là một lớp vô cùng quan trọng. Khi dự án sử dụng ASP.NET Core Identity, mặc định **IdentityDbContext** là DbContext của dự án và nó đã định nghĩa sẵn các bảng trong IdentityDbContext. Để tạo một DbContext riêng, ta chỉ cần cho nó kế thừa IdentityDbContext.

Khi tạo các bảng dưới cơ sở dữ liệu, mặc định các bảng trong IdentityDbContext sẽ có tên với tiền tố là AspNet như AspNetUser, AspNetUserRoles,... Nếu muốn loại bỏ tiền tố đó khi tạo cơ sở dữ liệu, ta chỉ cần thêm đoạn code sau vào DbContext của dự án:

foreach (var entityType in builder.Model.GetEntityTypes())

{

var tableName = entityType.GetTableName();

if (tableName.StartsWith("AspNet"))

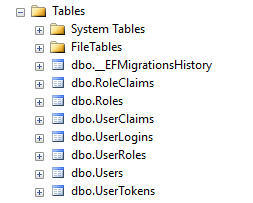
{

entityType.SetTableName(tableName.Substring(6));

}

}

Các bảng sau khi được tạo dưới cơ sở dữ liệu:



Hình 2.10: Các bảng trong IdentityDbContext

Để sử dụng chức năng chứng thực và phân quyền trong ASP.NET Core Identity, ta thêm vào phương thức Configure trong lớp Startup hai dòng sau để sử dụng dịch vụ:

app.UseAuthentication();

app.UseAuthorization();

Với **UseAuthentication** dùng để xác thực người dùng và **UseAuthorization** dùng để phân quyền người dùng.

### Xác thực người dùng

Chức năng xác thực người dùng được Identity xử lý rất tốt. Để xác thực thông tin người dùng trước khi truy cập một số chức năng, ta gán trước Action của các chức năng đó một thuộc tính [Authorize], thuộc tính này nằm trong thư viện **Microsoft.AspNetCore.Authorization**. Trước khi truy cập các chức năng đó thì hệ thống sẽ kiểm tra thông tin người dùng đã được xác thực hay chưa. Nếu chưa, người dùng phải đăng nhập mới có thể truy cập được những chức năng này.

[Authorize]

[HttpGet]

[Route("profile/change-address", Name = "change-address")]

public IActionResult ChangeAddress()

{

var qr = \_context.Address.ToList();

var address = qr.Where(d => d.UserId == \_userManager.GetUserId(HttpContext.User)).ToList();

if (address.Count() == 0)

{

return View();

}

else

{

var model = address.First();

return View(model);

}

}

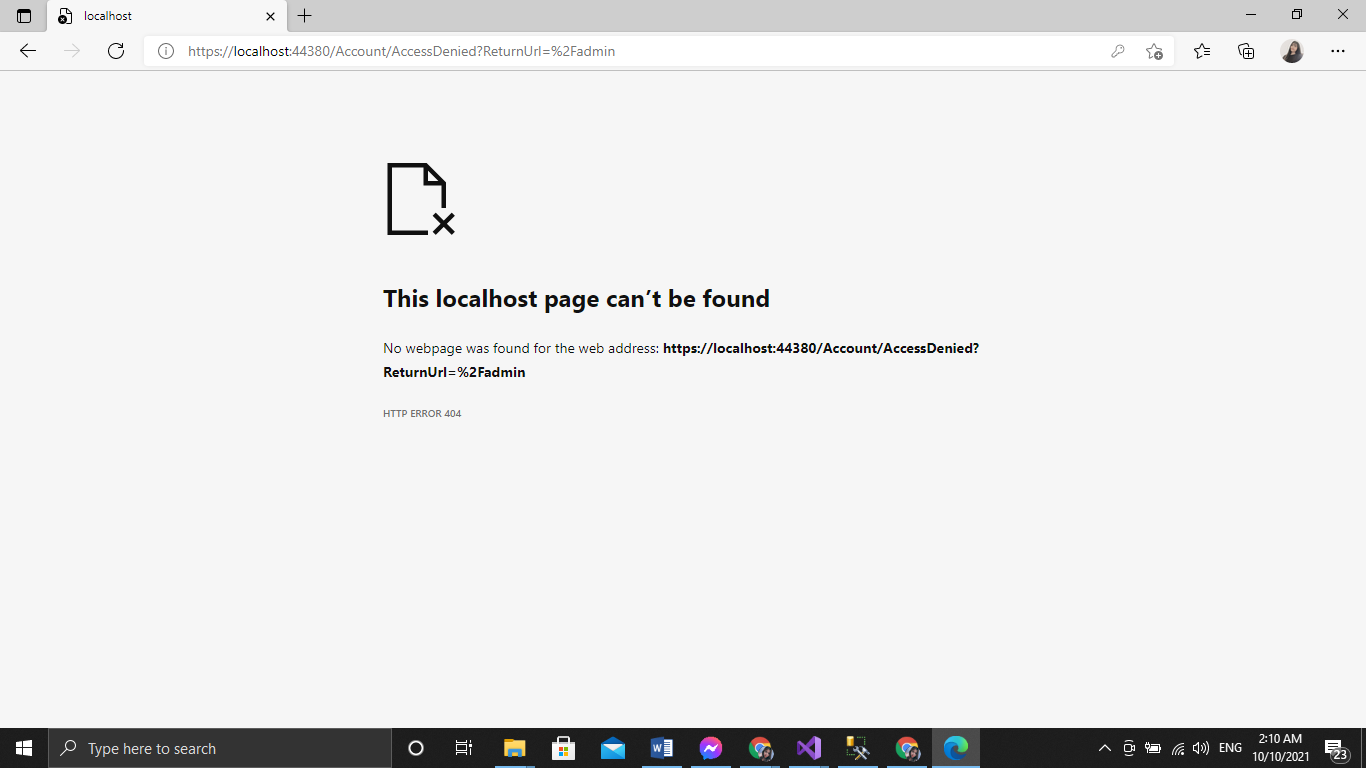
Nếu người dùng chưa có tài khoản thì phải tiến hành đăng ký tài khoản mới có thể truy cập. Với chức năng Đăng ký, khi người dùng gửi các thông tin cần đăng ký đến hệ thống, hệ thống sẽ tiến hành kiểm tra người dùng này đã có trong danh sách user chưa. Nếu tồn tại, nó sẽ trả về lỗi. Ngược lại, nó sẽ tiến hành băm mật khẩu của người dùng đăng ký và lưu xuống cơ sở dữ liệu cùng các thông tin khác mà người dùng cung cấp. Đồng thời nó cũng tạo cho người dùng một “tem an ninh” (Security Stamp) và “tem hợp lệ” (Concurrency Stamp) và hai giá trị này được lưu lần lượt trong thuộc tính SecurityStamp và ConcurrencyStamp của bảng User dưới cơ sở dữ liệu. Thuộc tính SecurityStamp được sử dụng cho mục đích bảo mật khi các thông tin quan trọng của người dùng thay đổi, chẳng hạn như thay đổi mật khẩu. Còn thuộc tính ConcurrencyStamp giúp ngăn chặn việc xảy ra xung đột trong việc cập nhật đồng thời thông tin của một người dùng. Qua đó, giúp cho thông tin người dùng được bảo mật hơn.

### Phân quyền người dùng

Hệ thống phân quyền người dùng của Identity vô cùng mạnh mẽ và cách thiết lập nó cũng vô cùng đơn giản. Để phân quyền người dùng trên một Action của một Controller, ta chỉ cần gán trước Action đó một thuộc tính [Authorize] và gắn thêm Roles cho nó.

[Authorize(Roles = "Admin, Employee")]

Nếu một người dùng có Roles khác Admin và Employee thì sẽ không cho truy cập và trả về trang HTTP ERROR 404.



Hình 2.11: Trang trả về 404 khi Roles của người dùng không hợp lệ

### Xác thực từ bên thứ ba

Ngoài chức năng xác thực và phân quyền trên Windows, ASP.NET Core Identity còn hỗ trợ xác thực người dùng thông qua bên thứ ba như Facebook, Google,... Để sử dụng chức năng này, đầu tiên ta cần cài đặt gói hỗ trợ chứng thực từ bên thứ ba. Tùy vào nhu cầu sử dụng mà ta chọn một hay nhiều gói trong những gói sau và chạy các lệnh sau trong Terminal để tiến hành cài đặt:

dotnet add package Microsoft.AspNetCore.Authentication.Facebook

dotnet add package Microsoft.AspNetCore.Authentication.Google

dotnet add package Microsoft.AspNetCore.Authentication.JwtBearer

dotnet add package Microsoft.AspNetCore.Authentication.MicrosoftAccount

dotnet add package Microsoft.AspNetCore.Authentication.Twitter

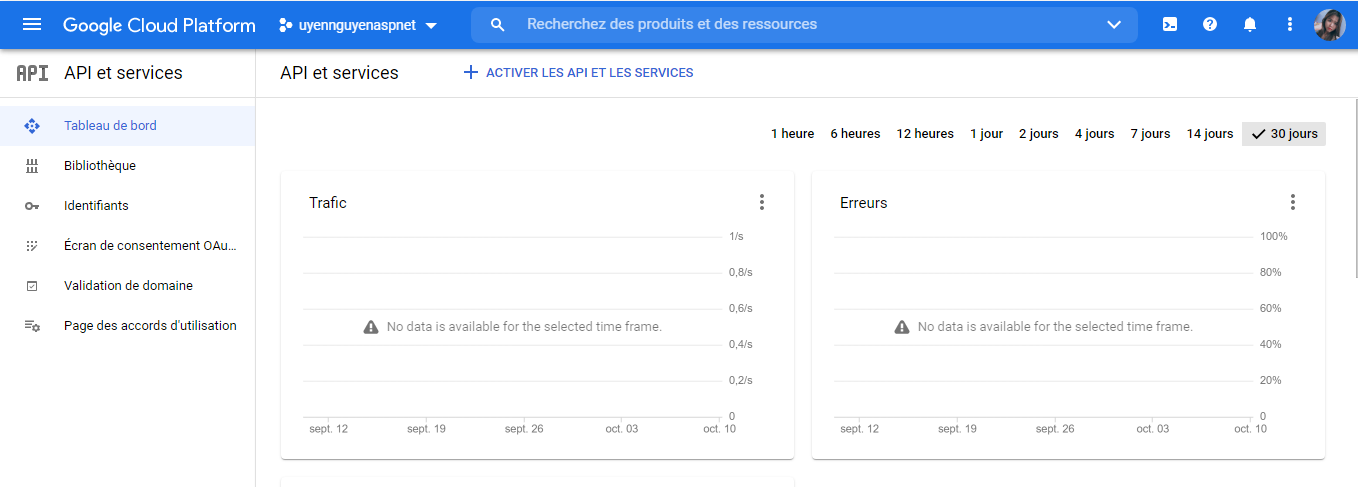
Ngoài ra, ta cũng có thể cài đặt các gói này trong Manage NuGet Packages.

Tiếp theo, ta cần phải đăng ký dịch vụ xác thực của các bên thứ ba mà ta đã cài đặt trước đó ở trên. Giả sử, ta chọn xác thực bên thứ ba là Facebook và Google.

#### Xác thực qua Google

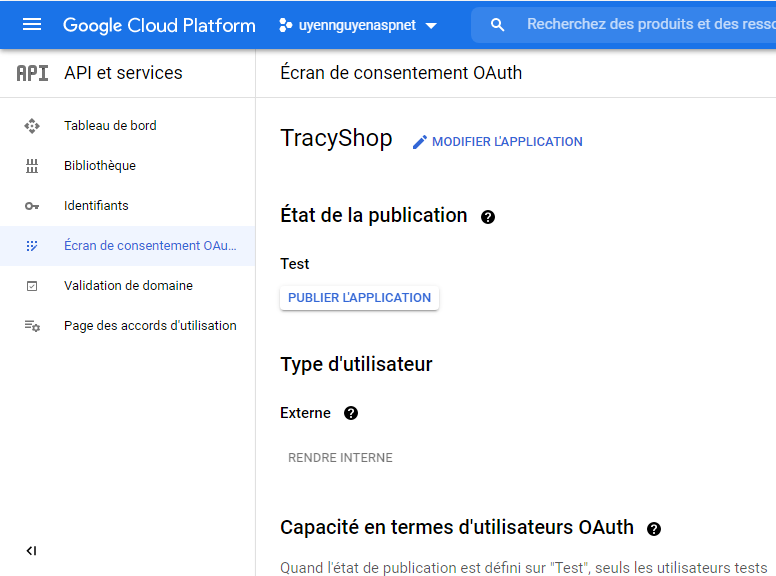
Xác thực qua Google hiện là hình thức xác thực được nhiều trang web hỗ trợ. Để xác thực qua Google, đầu tiên ta cần đăng ký dịch vụ xác thực từ Google:

* Truy cập vào trang <https://console.developers.google.com/> và bấm vào Create Project để tạo một dự án mới nếu bạn chưa có, giả sử ta tạo một dự án uyennguyenaspnet.



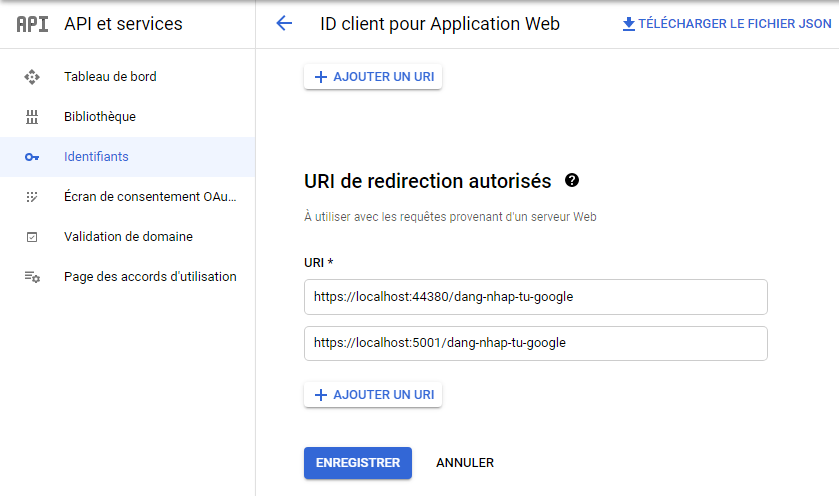
Hình 2.12: Giao diện của trang Google Cloud Platform

* Sau đó, ta tạo một ứng dụng mới, giả sử có tên là TracyShop



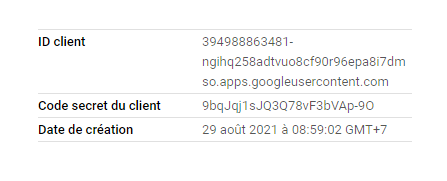
Hình 2.13: Giao diện sau khi tạo xong ứng dụng trong Google Cloud Platform

* Qua tab Identifiants, ta tạo một Oauth Client ID với loại ứng dụng là Web application và tên là TracyShop. Tên này ta có thể tự đặt tùy ý. Tại đây ta điền một URI để nó chuyển đến khi chứng thực.



Hình 2.14: Thiết lập URIs

Sau khi hoàn thành xong, Google Cloud Platform sẽ trả về một ID Client và một Secret Client.



Hình 2.15: ID Client và Secret Client của Google

Ta đặt hai giá trị này trong appsettings.json:

"Authentication": {

"Google": {

"ClientId":"394988863481- ngihq258adtvuo8cf90r96epa8i7dmso.apps.googleusercontent.com",

"ClientSecret": "9bqJqj1sJQ3Q78vF3bVAp-9O"

}

},

Tiếp đến, ta đăng ký dịch vụ xác thực bằng Google trong phương thức ConfigureServices của lớp Startup:

services.AddAuthentication()

.AddGoogle(googleOptions =>

{

IConfigurationSection googleAuthNSection = Configuration.GetSection("Authentication:Google");

googleOptions.ClientId = googleAuthNSection["ClientId"];

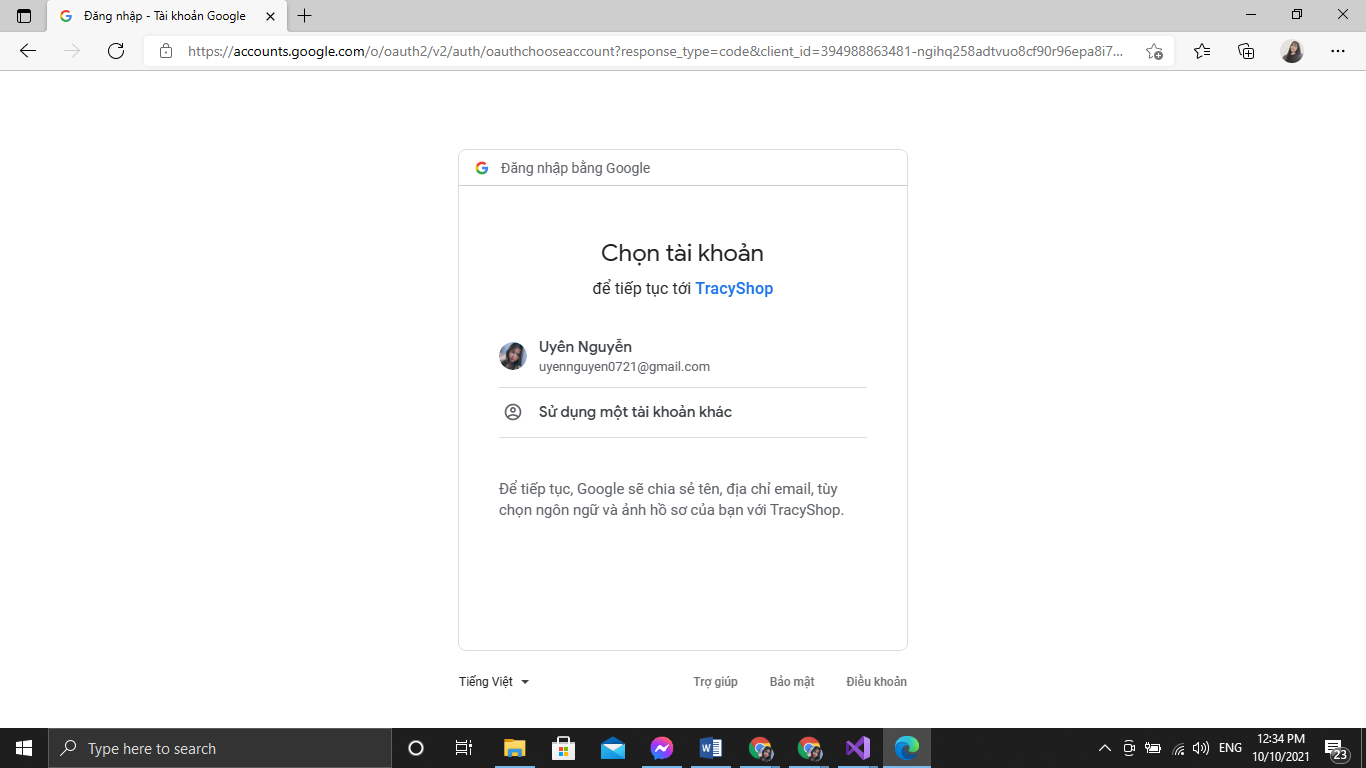
googleOptions.ClientSecret = googleAuthNSection["ClientSecret"];

googleOptions.ClaimActions.MapJsonKey("image", "picture");

googleOptions.CallbackPath = "/dang-nhap-tu-google";

});

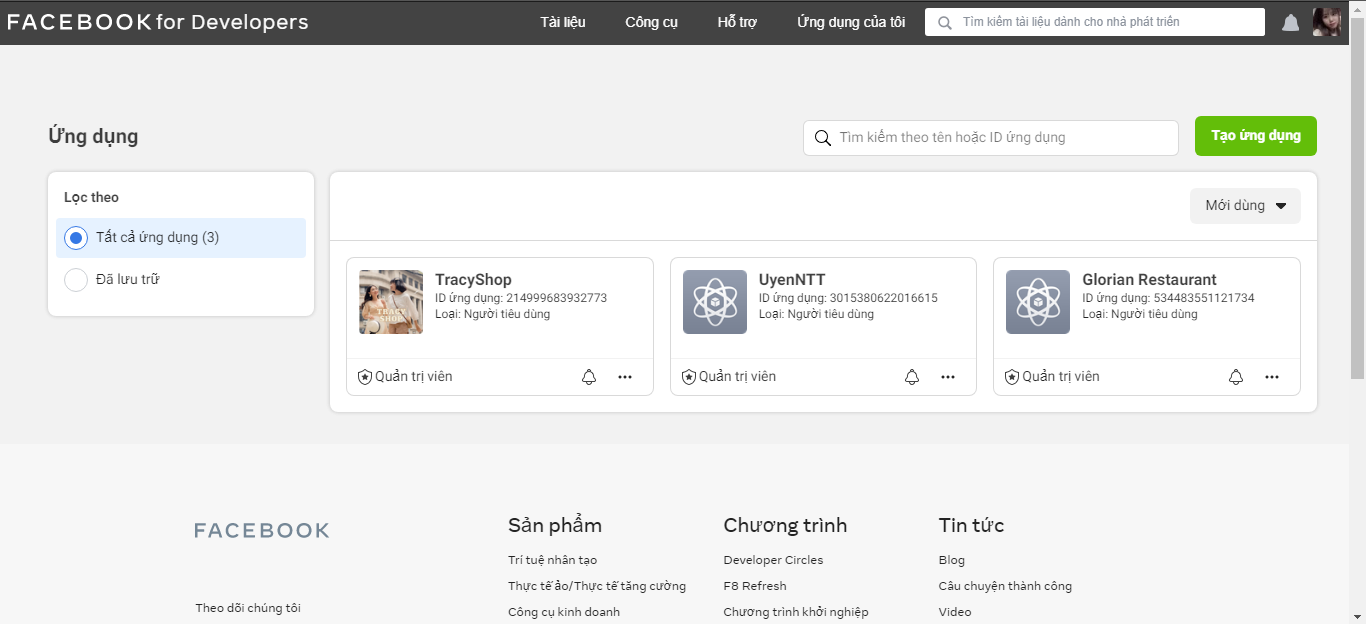
Nếu các cấu hình cài đặt không gặp vấn đề sai sót gì thì Identity trong dự án đã có thể cho phép người dùng đăng nhập qua Google.



Hình 2.16: Màn hình đăng nhập qua Google

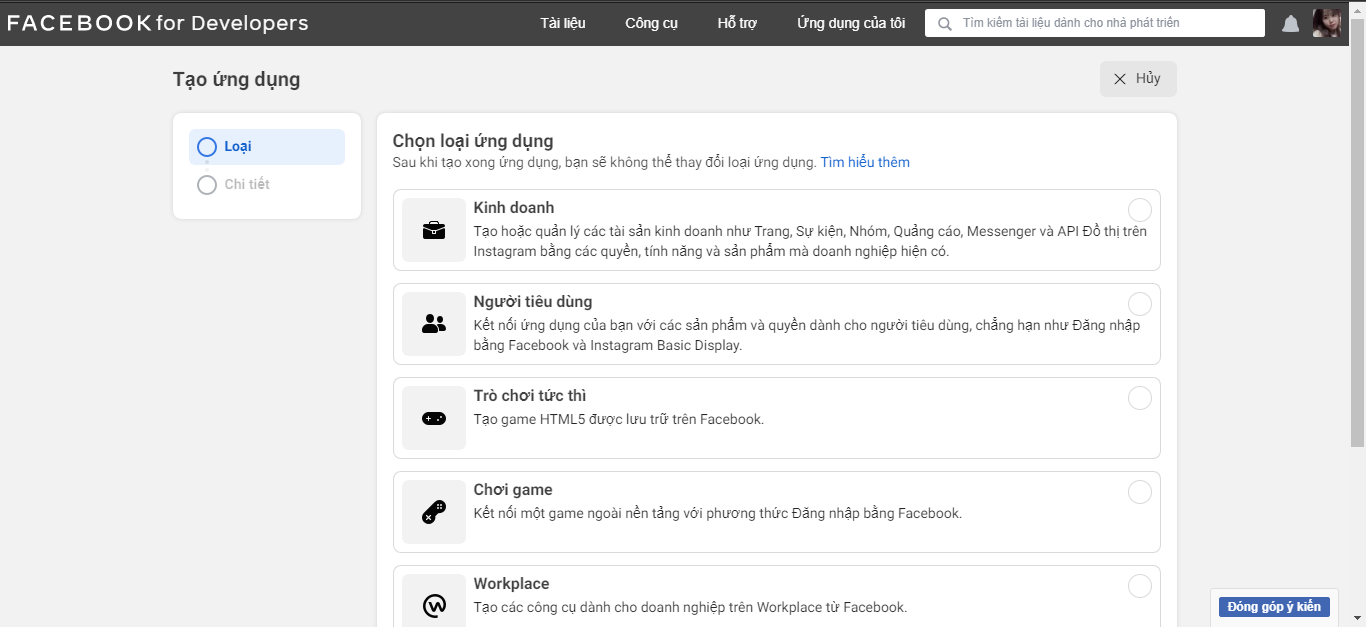
#### Xác thực qua Facebook

Tương tự như Google, việc xác thực qua Facebook cũng được nhiều trang web hỗ trợ. Để xác thực được qua Facebook, ta phải tạo tài khoản phát triển trên Facebook bằng cách truy cập vào địa chỉ <https://developers.facebook.com/apps/> và tạo một ứng dụng, ví dụ một ứng dụng có tên là TracyShop. Tên ứng dụng ta tự đặt tùy ý.



Hình 2.17: Giao diện của trang Facebook for Developers

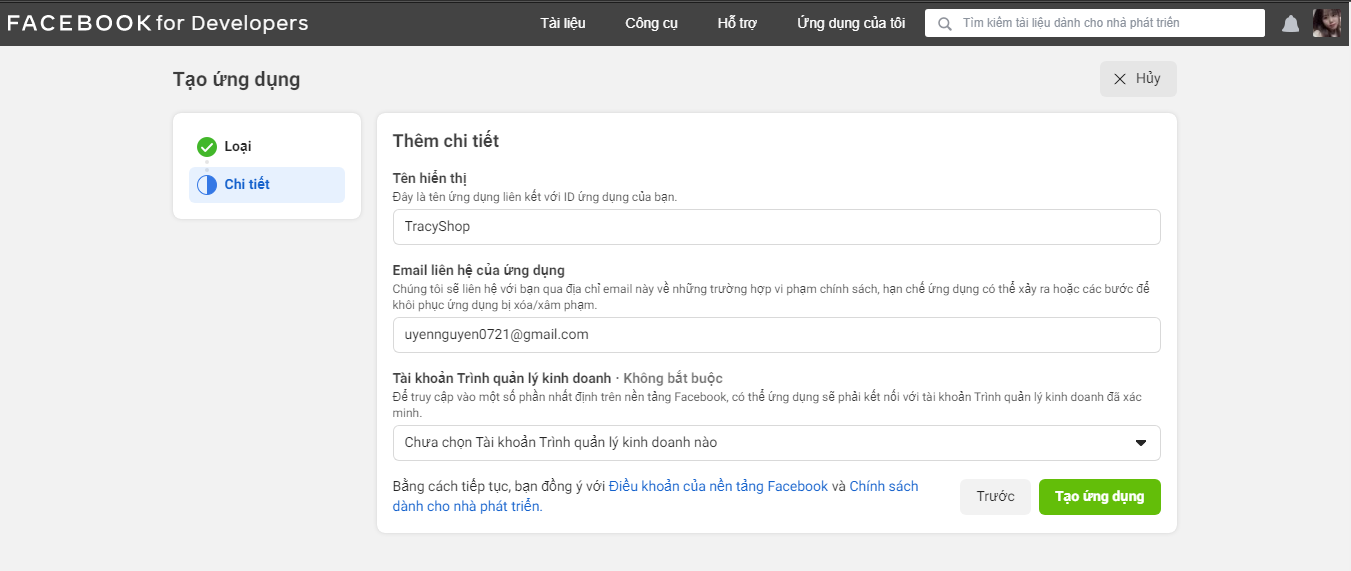
Khi tạo ứng dụng, đầu tiên nó sẽ hiển thị trang để người dùng chọn loại ứng dụng.



Hình 2.18: Giao diện chọn loại ứng dụng trên Facebook for Developers

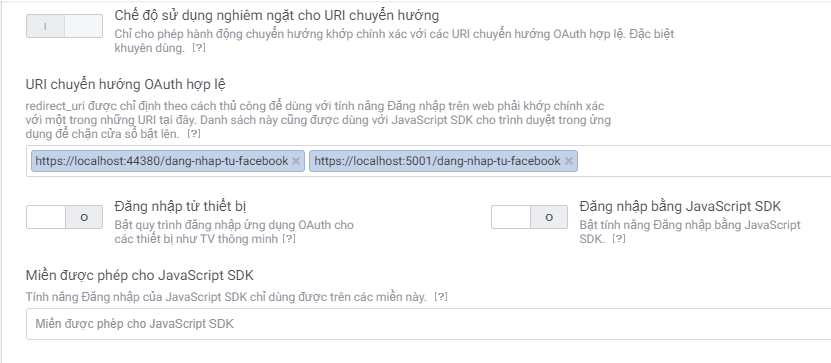
Tùy vào nhu cầu của người dùng mà ta chọn loại ứng dụng phù hợp. Với nhu cầu chỉ dùng để đăng nhập qua Facebook nên ta chọn loại ứng dụng Người tiêu dùng.

Sau khi chọn loại ứng dụng, nó sẽ yêu cầu người dùng điền thông tin của ứng dụng như tên ứng dụng, email liên hệ.



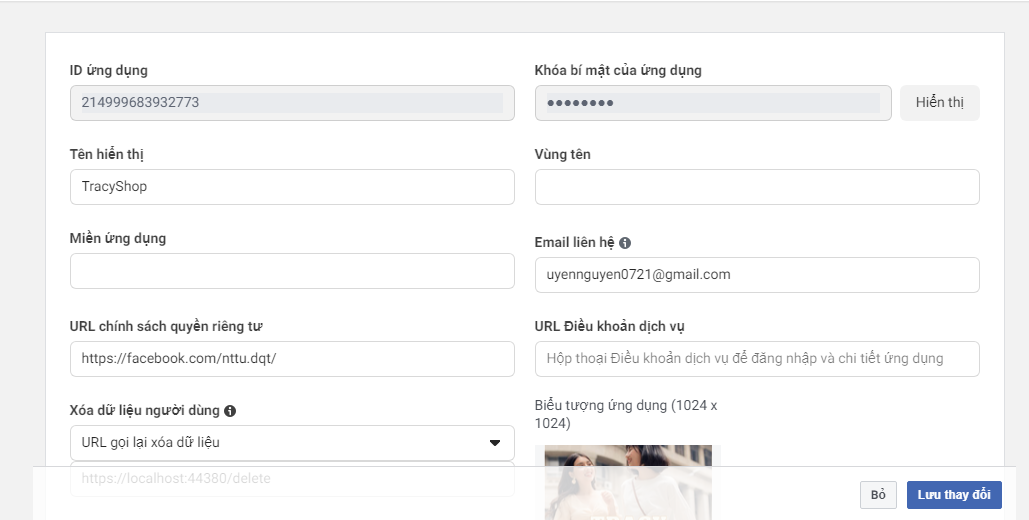
Hình 2.19: Giao diện thêm chi tiết ứng dụng trong Facebook for Developers

Tạo ứng dụng xong, ta thiết lập tính năng đăng nhập bằng Facebook và chọn nền tảng ứng dụng là web. Tiếp theo, ta chọn Cài đặt trong mục Đăng nhập bằng Facebook và nhập địa chỉ URL chuyển hướng OAuth hợp lệ với tên miền là tên miền địa chỉ web của dự án đang chạy trên trình duyệt.



Hình 2.20: Nhập URL chuyển hướng trong Facebook for Developers

Cuối cùng, ta vào Cài đặt và chọn mục Thông tin cơ bản để xem giá trị ID của ứng dụng và khóa bí mật.



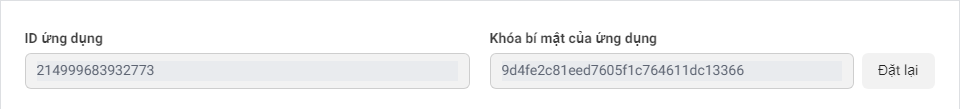
Hình 2.21: Các thông tin cơ bản của ứng dụng trên Facebook for Developers

Mặc định, khóa bí mật sẽ không hiển thị. Để xem ta cần nhấn vào hiển thị bên phải khóa bí mật, nó sẽ yêu cầu bạn nhập mật khẩu Facebook của bạn để xác nhận:



Hình 2.22: Nhập mật khẩu Facebook để xem khóa bí mật của ứng dụng

Sau khi xác nhận xong, nó sẽ hiển thị khóa bí mật lên màn hình và khóa bí mật này ta có thể đặt lại nếu cảm thấy không an toàn.



Hình 2.23: ID ứng dụng và khóa ứng dụng của Facebook

Ta đặt hai giá trị này trong appsettings.json của dự án:

"Authentication": {

"Facebook": {

"AppId": "214999683932773",

"AppSecret": "9d4fe2c81eed7605f1c764611dc13366"

}

},

Tiếp đến, ta đăng ký dịch vụ xác thực bằng Facebook trong phương thức ConfigureServices của lớp Startup:

services.AddAuthentication()

.AddFacebook(facebookOptions =>

{

IConfigurationSection facebookAuthNSection = Configuration.GetSection("Authentication:Facebook");

facebookOptions.AppId = facebookAuthNSection["AppId"];

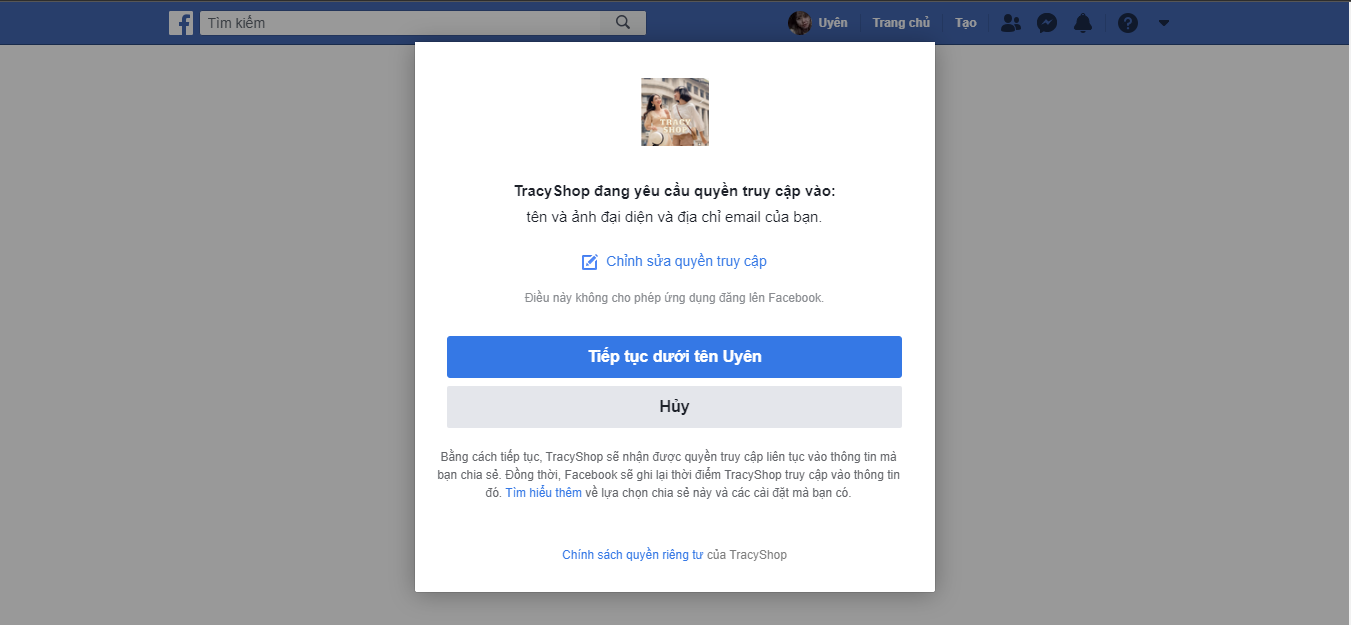
facebookOptions.AppSecret = facebookAuthNSection["AppSecret"];

facebookOptions.ClaimActions.MapJsonKey("image", "picture");

facebookOptions.CallbackPath = "/dang-nhap-tu-facebook";

});

Nếu các cấu hình cài đặt không gặp vấn đề sai sót gì thì Identity trong dự án đã có thể cho phép người dùng đăng nhập qua Facebook:



Hình 2.24: Màn hình đăng nhập qua Facebook

### Các cấu hình dịch vụ trong Identity

Identity được biết đến với những tính năng mạnh mẽ trong việc xác thực và phân quyền người dùng. Bên cạnh đó, Identity cũng hỗ trợ những dịch vụ nhằm đảm bảo việc bảo mật thông tin người dùng được chặt chẽ. Những dịch vụ này được cung cấp bởi **IdentityOptions**. Nó nằm trong thư viện **Microsoft.AspNetCore.Identity**.

Identity cung cấp các cấu hình dịch vụ như các thiết lập về mật khẩu, lockout user (khóa người dùng), đăng nhập người dùng,... Những cấu hình dịch vụ này được cấu hình trong phương thức **ConfigureServices** trong lớp **Startup** của dự án.

Về mật khẩu, **IdentityOptions** có những thiết lập cụ thể giúp cho mật khẩu của người dùng mạnh mẽ hơn và có tính bảo mật cao, cụ thể là mật khẩu bắt buộc phải có số, chữ thường, kí tự đặc biệt hoặc chữ in hoa,... hay quy định số kí tự tối thiểu của mật khẩu, số kí tự đặc biệt,...

services.Configure<IdentityOptions>(options => {

// Bắt buộc phải có số

options.Password.RequireDigit = true;

// Bắt buộc phải có chữ thường

options.Password.RequireLowercase = true;

// Bắt buộc có ký tự đặc biệt

options.Password.RequireNonAlphanumeric = true;

// Không bắt buộc chữ in

options.Password.RequireUppercase = false;

// Số ký tự tối thiểu của password

options.Password.RequiredLength = 6;

// Số ký tự riêng biệt

options.Password.RequiredUniqueChars = 1;

});

Trong quá trình đăng nhập sẽ không thể tránh khỏi việc người dùng đăng nhập thất bại nhiều lần do quên mật khẩu hay một người lạ cố tình tấn công tài khoản của một người dùng bằng cách đăng nhập thất bại nhiều lần để dò tìm tên đăng nhập hoặc mật khẩu. Dù nguyên nhân là gì thì nó luôn đe dọa đến việc thông tin người dùng bị đánh cắp. Do đó, Identity đã cung cấp tính năng lockout user (khóa người dùng), cho phép người dùng đăng nhập thất bại trong số lần cho phép.

services.Configure<IdentityOptions>(options => {

// Khóa 5 phút

options.Lockout.DefaultLockoutTimeSpan = TimeSpan.FromMinutes(5);

// Khóa khi đăng nhập thất bại 5 lần

options.Lockout.MaxFailedAccessAttempts = 5;

options.Lockout.AllowedForNewUsers = true;

});

Identity cũng cung cấp những thiết lập cấu hình cho người dùng như quy định các kí tự dùng để đặt tên người dùng, email của người dùng phải duy nhất,...

services.Configure<IdentityOptions>(options => {

// quy định các ký tự đặt tên user

options.User.AllowedUserNameCharacters = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789 -.\_@+";

// Email là duy nhất

options.User.RequireUniqueEmail = true;

});

Ngoài ra, để tăng tính bảo mật, Identity cũng cung cấp tính năng xác thực email và số điện thoại để kiểm tra xem thông tin email và số điện thoại người dùng cung cấp có tồn tại hay không.

services.Configure<IdentityOptions>(options => {

// Cấu hình xác thực địa chỉ email (email phải tồn tại)

options.SignIn.RequireConfirmedEmail = true;

// Xác thực số điện thoại

options.SignIn.RequireConfirmedPhoneNumber = false;

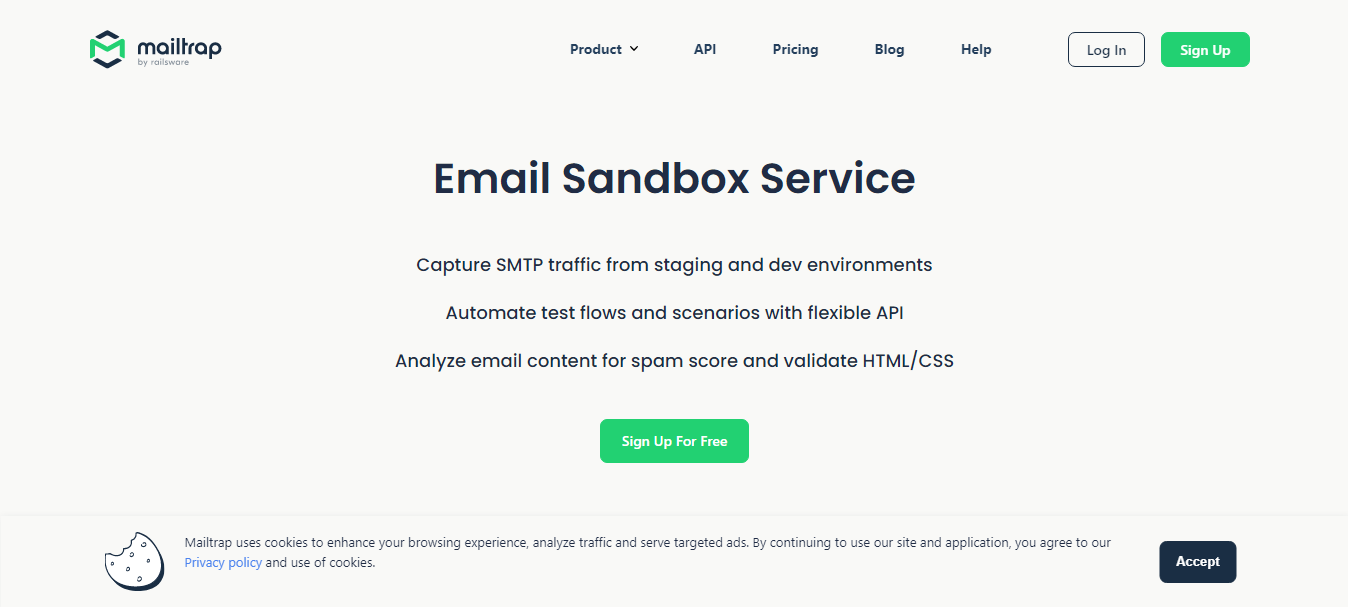
});

Với những dịch vụ bảo mật người dùng mà Identity cung cấp không những giúp cho việc tăng tính bảo mật thông tin người dùng mà còn giúp cho các lập trình viên không cần phải code quá nhiều, từ đó giúp tiết kiệm thời gian lập trình và kiểm tra code. Việc sử dụng các dịch vụ của Identity cũng vô cùng dễ dàng, chỉ cần đăng kí là có thể sử dụng các dịch vụ của nó. Do đó, nó giúp cho code được sạch sẽ và dễ nhìn hơn, đồng thời cũng giúp cho quá trình kiểm tra và chỉnh sửa được dễ dàng.

## Kiểm tra gửi mail bằng Mailtrap

### Giới thiệu về Mailtrap

Với một dự án công nghệ thông tin thì việc kiểm thử các chức năng luôn được tiến hành xuyên suốt từ giai đoạn viết code đến giai đoạn hoàn thành nhằm nhanh chóng phát hiện những lỗi sai và khắc phục chúng. Trong thực tế, để kiểm thử các chức năng của ứng dụng khi một dự án hoàn thành, các nhà phát triển web hay phần mềm thường đưa ra các phiên bản beta đến người dùng để người dùng trải nghiệm thử các tính năng và đưa ra các phản hồi. Từ các phản hồi của người dùng, các nhà lập trình sẽ tiến hành sửa lỗi và cập nhật lại các chức năng của ứng dụng. Nhưng với một số chức năng, việc kiểm thử trên người dùng sẽ gặp một số khó khăn. Điển hình là chức năng gửi mail đến người dùng đã đăng ký, việc nhận hàng loạt các mail rác sẽ làm cho người dùng cảm thấy khó chịu và bị làm phiền. Đồng thời, nó cũng vi phạm tính bảo mật người dùng. Hoặc khi gửi mail đến danh sách người dùng đã đăng ký, ta không thể xác nhận được mail này đã gửi đến tất cả người dùng hay chưa. Và rồi, nhiều công cụ kiểm tra gửi mail đã ra đời để giải quyết các vấn đề trên, trong đó Mailtrap là một công cụ kiểm tra gửi mail được nhiều người sử dụng.

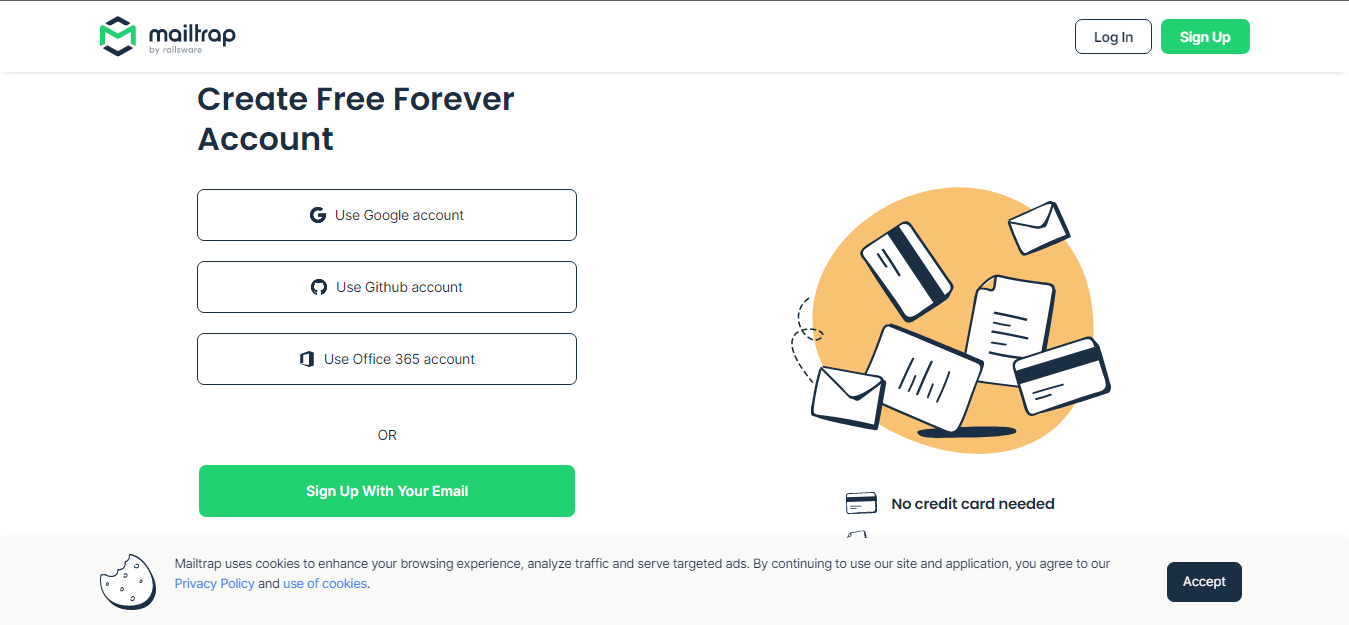


Hình 2.25: Trang chủ của Mailtrap

### Mailtrap là gì?

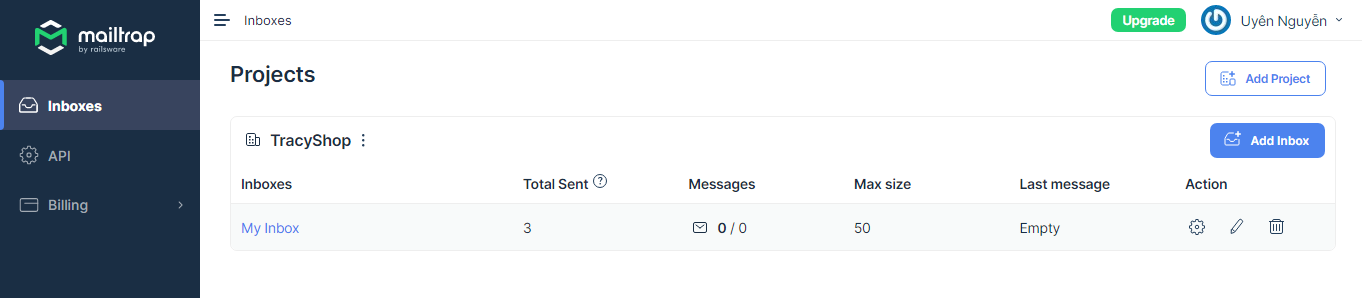
Mailtrap là một công cụ cung cấp máy chủ SMTP giả dùng để kiểm tra các mail được gửi từ môi trường pre-production và môi trường kiểm thử thực tế, giúp tránh được tình trạng gửi thư rác đến người dùng. Đồng thời, nó tự động đưa các mail quan trọng đến hộp thư thường xuyên và có hỗ trợ chia sẻ với nhóm.

Mailtrap có giao diện thân thiện với người dùng. Để sử dụng, ta cần đăng ký một tài khoản. Ta có thể đăng đăng ký bằng email hay sử dụng tài khoản Google, Github hay Office 365.



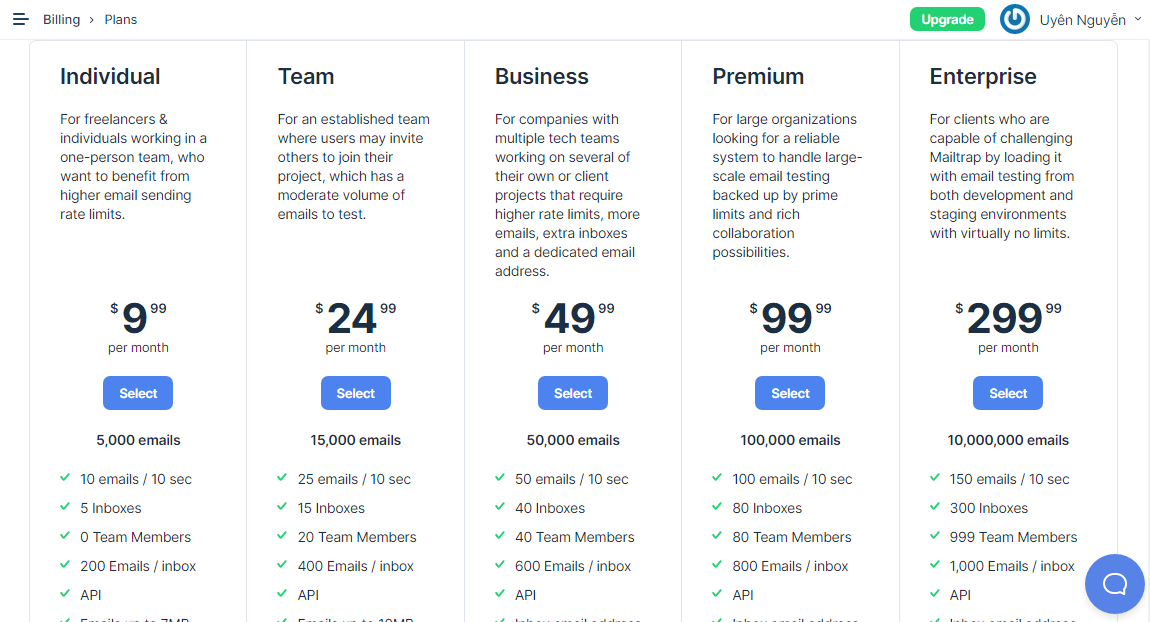
Hình 2.26: Giao diện đăng ký của Mailtrap

Sau khi tạo tài khoản và đăng nhập, ta đã có thể tiến hành tích hợp Mailtrap vào dự án để tiến hành kiểm thử gửi mail.



Hình 2.27: Giao diện hộp thư đến trên Mailtrap

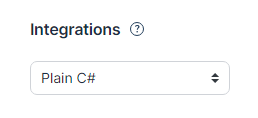
Với dự án nhỏ chỉ kiểm tra gửi mail trên số lượng ít không quá 50 mail thì ta có thể sử dụng bản miễn phí của Mailtrap. Ngược lại nếu nhu cầu của hệ thống cần kiểm tra gửi mail với số lượng lớn thì ta chọn bản trả phí để tăng giới hạn số lượng email. Dưới đây là các bản trả phí trên Mailtrap với các số lượng email giới hạn tương ứng:



Hình 2.28: Các bản trả phí trên Mailtrap

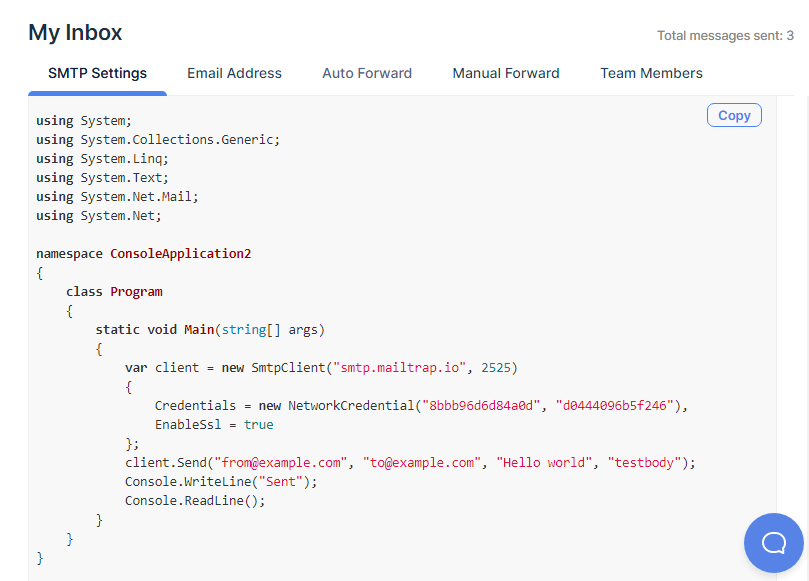
### Cấu hình dịch vụ gửi mail qua Mailtrap

Mailtrap sử dụng phương thức SMTP để gửi mail nên nó có thể hoạt động trên các port 25, 2525, 465, 587 và một số port khác của SMTP. Để xem cấu hình của Mailtrap, ta vào My Inbox chọn mục SMTP Settings, ở phần Integrations chọn ngôn ngữ được sử dụng trong dự án. Trong dự án này sử dụng ngôn ngữ C# nên ta chọn Plain C#.



Hình 2.29: Chọn Integrations

Sau khi chọn Plain C#, nó hiện ra một đoạn code C#, trong đó có chứa các giá trị cấu hình của SMTP trong Mailtrap như host, port, username, password và EnableSSL.



Hình 2.30: Cấu hình SMTP trong Mailtrap

Ta đưa cấu hình này vào tập tin appsettings.json của dự án:

"SMTPConfig": {

"SenderAddress": "no-reply@tracyshop.com",

"SenderDisplayName": "TracyShop Team",

"UserName": "8bbb96d6d84a0d",

"Password": "d0444096b5f246",

"Host": "smtp.mailtrap.io",

"Port": 587,

"EnableSSL": true,

"UseDefaultCredentials": true,

"IsBodyHTML": true

},

Giá trị SenderAddress trong cấu hình là địa chỉ email của người gửi. Giá trị này ta tự đặt sao cho đúng với cấu hình của một địa chỉ email. Tương tự với giá trị SenderDisplayName. Giá trị này là tên hiển thị của người gửi và giá trị này cũng được đặt tùy ý.

### Thiết lập gửi mail qua Mailtrap

Trong ASP.NET Core, chức năng gửi mail được hỗ trợ bởi thư viện **System.Net**. Thư viện này cung cấp lớp **SmtpClient** hỗ trợ gửi mail thông qua giao thức SMTP [11]. Các thông tin của mail như tiêu đề, nội dung, địa chỉ email và tên người gửi được tạo trong đối tượng lớp **MailMessage**. Lớp này nằm trong thư viện **System.Net.Mail**, là một thư viện con của thư viện **System.Net**. Sau khi có các nội dung của mail, ta gọi phương thức **SendMailAsync** trong SmtpClient để tiến hành gửi mail cho người dùng.

Để thiết lập gửi mail bằng Mailtrap, đầu tiên ta xây dựng một giao diện có tên IEmailService. Giao diện này dùng để triển khai các dịch vụ đến người dùng, giả sử dịch vụ trong giao diện cần triển khai là kiểm tra gửi mail đến các người dùng đăng ký:

public interface IEmailService

{

Task SendEmailForTestMail(UserEmailOptions userEmailOptions);

}

Tiếp theo, ta xây dựng một lớp EmailService. Lớp này kế thừa và triển khai các dịch vụ trong giao diện IEmailService. Để lấy các giá trị cấu hình SMTP trong appsettings.json, ta tạo một lớp có tên là SMTPConfigModel chứa các thuộc tính cấu hình của SMTP trong appsettings.json như SenderAddress (địa chỉ người gửi), SenderDisplayName (tên hiển thị của người gửi), Username (tên đăng nhập mail), Password (mật khẩu đăng nhập mail), Host (địa chỉ tên miền của host), Port,...

Trong EmailService, ta tạo một đối tượng lớp này và khởi tạo nó để sử dụng:

private readonly SMTPConfigModel \_smtpConfig;

public EmailService(IOptions<SMTPConfigModel> smtpConfig)

{

\_smtpConfig = smtpConfig.Value;

}

Sau khi được khởi tạo, các giá trị cấu hình SMTPConfig trong appsettings.json sẽ được lưu vào đối tượng \_smtpConfig.

Ta tiếp tục tạo phương thức SendEmail trong EmailService để xử lý gửi mail với tham số truyền vào là một đối tượng lớp UserEmailOptions. Lớp này chứa các thuộc tính sau:

* ToEmail: danh sách địa chỉ email của người nhận
* Subject: chủ đề mail
* Body: nội dung mail
* PlaceHolders: đây là một thuộc tính có kiểu danh sách (List), mỗi phần tử trong danh sách là một KeyValuePair<string, string> chứa một thông tin của người dùng để gắn các key vào template của mail được tạo bằng HTML và nó sẽ trả về value của từng key để hiển thị lên giao diện mail.

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<title></title>

</head>

<body>

<p>

Hello **{{**UserName**}}**, <br />

This email is coming from TracyShop web app

</p>

</body>

</html>

Đoạn HTML trên là một template mail dùng để kiểm tra gửi mail đến người dùng đăng ký với {{Username}} là key của thuộc tính PlaceHolders được gán vào HTML và nó sẽ trả về value của key để hiển thị tên người dùng trong mail.

Trong phương thức SendEmail, ta tạo một đối tượng lớp MailMessage chứa thông tin của một mail như chủ đề, nội dung, thông tin người gửi với các thuộc tính như Subject, Body, From, IsBodyHtml,... Thuộc tính From là một đối tượng lớp MailAddress lưu thông tin của người gửi như địa chỉ email, tên hiển thị.

MailMessage mail = new MailMessage

{

Subject = userEmailOptions.Subject,

Body = userEmailOptions.Body,

From = new MailAddress(\_smtpConfig.SenderAddress, \_smtpConfig.SenderDisplayName),

IsBodyHtml = \_smtpConfig.IsBodyHTML

};

Để biết được mail này sẽ gửi đến những người nhận nào, ta cần thêm những địa chỉ email của người nhận chứa trong thuộc tính ToEmail của đối tượng UserEmailOptions vào thuộc tính To trong đối tượng mail.

foreach (var toEmail in userEmailOptions.ToEmails)

{

mail.To.Add(toEmail);

}

Cũng trong phương thức này, ta tạo một đối tượng lớp SmtpClient chứa thông tin cấu hình gửi mail thông qua giao thức SMTP. Trong thông tin cấu hình này có chứa thông tin xác thực để khi lên server, nó sẽ tiến hành xác thực trước khi gửi mail. Ta tạo một đối tượng lớp NetworkCredential chứa thông tin xác thực SMTP gồm tên đăng nhập và mật khẩu được lấy từ SMTPConfig trong appsettings.json:

NetworkCredential networkCredential = new NetworkCredential(\_smtpConfig.UserName, \_smtpConfig.Password);

Sau đó gắn nó vào trong thuộc tính Credentials của đối tượng lớp SmtpClient. Thuộc tính này chứa thông tin xác thực SMTP:

SmtpClient smtpClient = new SmtpClient

{

UseDefaultCredentials = false,

Host = \_smtpConfig.Host,

Port = \_smtpConfig.Port,

EnableSsl = \_smtpConfig.EnableSSL,

Credentials = networkCredential

};

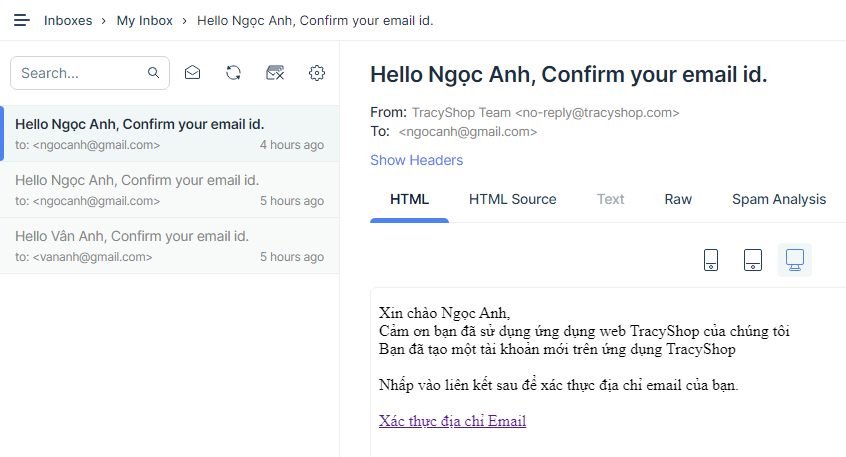
Để quy định hiển thị nội dung mail trong Mailtrap, ta sử dụng đối tượng lớp **Encoding** trong thư viện **System.Text** và gán giá trị cho thuộc tính BodyEncoding trong đối tượng mail:

mail.BodyEncoding = Encoding.Default;

Sau khi thiết lập cấu hình cũng như có các thông tin, nội dung của mail, ta gọi phương thức SendMailAsync trong lớp SmtpClient để gửi mail này đến địa chỉ email của người nhận:

await smtpClient.SendMailAsync(mail);

Trong lớp EmailService, ta tiếp tục tạo các phương thức gửi mail riêng cho từng chức năng như phương thức gửi mail cho chức năng xác nhận địa chỉ email, phương thức gửi mail cho chức năng quên mật khẩu,... Các phương thức này đều sử dụng phương thức SendEmail để gửi mail và với mỗi chức năng sẽ có những chủ đề và nội dung mail được tạo khác nhau. Nội dung mail của từng chức năng được tạo bằng HTML.



Hình 2.31: Giao diện nhận mail trong Mailtrap

# WEBSITE BÁN QUẦN ÁO NỮ ONLINE TRACYSHOP

Chương này nhằm giới thiệu về đề tài “Xây dựng web bán quần áo nữ online TracyShop”, trình bày về phân tích thiết kế của hệ thống gồm mô tả các chức năng và các sơ đồ tuần tự của từng chức năng, thiết kế các bảng trong cơ sở dữ liệu và thực nghiệm hệ thống.

## Giới thiệu đề tài

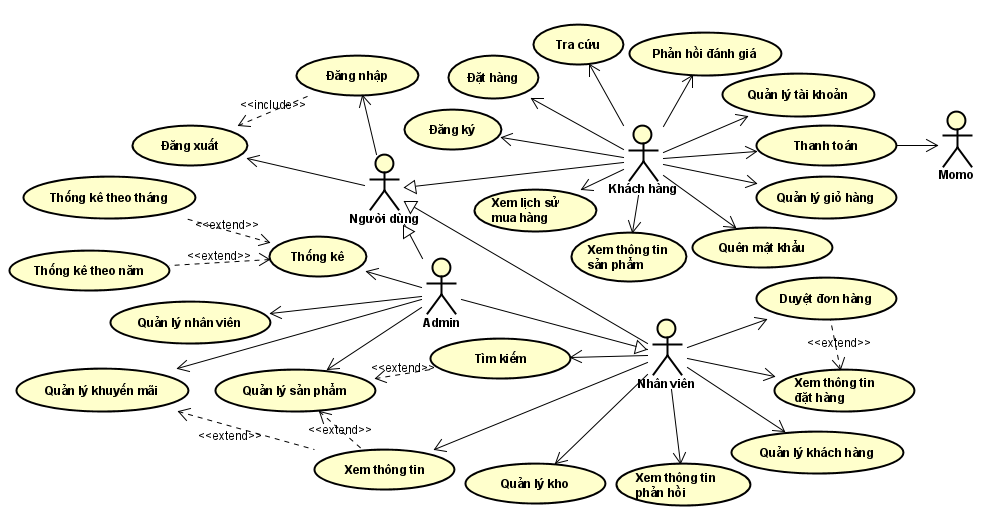
Hiện nay, tình hình dịch Covid-19 đang diễn biến phức tạp, việc đi lại của mọi người cũng trở nên khó khăn. Nhiều người đã lựa chọn việc mua hàng qua các trang web, ứng dụng thay cho việc mua hàng tại các cửa hàng, siêu thị. Do đó, mua hàng online đã trở nên phổ biến và là lựa chọn hàng đầu của nhiều người cũng một phần bởi tính tiện lợi mà nó mang lại. Cùng với đó, các cửa hàng thời trang cũng đẩy mạnh việc mua bán online để bắt kịp xu hướng và đáp ứng nhu cầu của người dùng.

Website bán quần áo nữ online TracyShop cung cấp nhiều tính năng cho người dùng mua sắm tại nhà. Người dùng có thể truy cập để xem danh sách và chi tiết các các sản phẩm của shop, thực hiện các chức năng quản lý tài khoản. Đồng thời, hệ thống cũng cho phép người dùng đánh giá sản phẩm, bình luận và xem các đánh giá, bình luận của các người dùng khác. Ngoài ra, người dùng có thể đặt hàng, thêm sản phẩm vào giỏ hàng và được hỗ trợ thanh toán đơn hàng với nhiều hình thức như thanh toán khi nhận hàng và thanh toán qua cổng thanh toán điện tử Momo. Bên cạnh đó, hệ thống cũng có các chức năng quản lý dành cho admin như quản lý sản phẩm, quản lý đơn hàng, quản lý khách hàng,...

## Phân tích và thiết kế hệ thống

### Đặc tả usecase

#### Sơ đồ usecase



Hình 3.1: Sơ đồ usecase

#### Usecase đăng nhập

|  |  |
| --- | --- |
| Mô tả | Use case cho phép người dùng đăng nhập vào hệ thống để thực hiện các chức năng |
| Actor chính | Người dùng |
| Actor phụ | Không có |
| Tiền điều kiện | Người dùng phải có tài khoản trên hệ thống |
| Hậu điều kiện | Nếu người dùng đăng nhập thành công thì được thực hiện chức năng của hệ thống, ngược lại sẽ tiến hành xử lý lỗi đăng nhập |
| Luồng hoạt động | 1. Hệ thống hiển thị màn hình đăng nhập  2. Người dùng nhập tên đăng nhập và mật khẩu  3. Hệ thống kiểm tra tên đăng nhập và mật khẩu.  4. Nếu thành công thì hiển thị trang mà người dùng chọn trước đó, ngược lại hiển thị thông báo lỗi và quay lại bước 2.  5. Kết thúc use case. |
| Luồng ngoại lệ | **I -** **Mật khẩu không hợp lệ**  Khi người dùng nhập sai tên đăng nhập và mật khẩu:  1. Hệ thống hiển thị lại màn hình đăng nhập để người dùng nhập lại thông tin kèm theo thông báo lỗi.  2. Quay lại bước 2 trong luồng sự kiện chính. |

Bảng .: Đặc tả usecase đăng nhập

#### Usecase đăng xuất

|  |  |
| --- | --- |
| Mô tả | Use case cho phép người dùng đăng xuất khỏi hệ thống |
| Actor chính | Người dùng |
| Actor phụ | Không có |
| Tiền điều kiện | Người dùng phải được đăng nhập trước đó |
| Hậu điều kiện | Sau khi đăng xuất người dùng chỉ có thể tìm kiếm và xem thông tin sản phẩm |
| Luồng hoạt động | 1. Người dùng chọn chức năng đăng xuất  2. Hệ thống tiến hành đăng xuất, sau đó trả về trang chủ của hệ thống.  3. Kết thúc use case. |

Bảng .: Đặc tả usecase đăng xuất

#### Usecase đăng ký

|  |  |
| --- | --- |
| Mô tả | Use case cho phép người dùng đăng ký tài khoản |
| Actor chính | Khách hàng |
| Actor phụ | Không có |
| Tiền điều kiện | Tài khoản khách hàng đăng ký phải chưa được đăng ký trước đó. |
| Hậu điều kiện | Nếu khách hàng đăng ký thành công thì sẽ tiến hành đăng nhập vào hệ thống, ngược lại sẽ tiến hành xử lý lỗi đăng ký. |
| Luồng hoạt động | 1. Hệ thống hiển thị màn hình đăng ký  2. Khách hàng nhập các thông tin yêu cầu  3. Hệ thống kiểm tra thông tin khách hàng cung cấp  4. Nếu thông tin tài khoản hợp lệ thì hệ thống sẽ gửi mail đến địa chỉ email của khách hàng dùng để đăng ký để xác nhận địa chỉ email này có tồn tại hay không, ngược lại sẽ hiển thị thông báo lỗi và quay lại bước 2.  5. Khách hàng kiểm tra mail, nếu mail đã được gửi đến thì khách hàng truy cập đường dẫn được gửi trong mail để xác thực, ngược lại khách hàng yêu cầu hệ thống gửi lại mail xác thực và tiếp tục thực hiện lại bước 5.  5. Kết thúc use case. |
| Luồng ngoại lệ | **I –** **Tài khoản đã tồn tại:**  Khi hệ thống kiểm tra tài khoản này đã tồn tại:  1. Hệ thống hiển thị lại màn hình đăng ký để khách hàng nhập lại thông tin kèm theo thông báo lỗi.  2. Quay lại bước 2 trong luồng sự kiện chính. |

Bảng .: Đặc tả usecase đăng ký

#### Usecase đặt hàng

|  |  |
| --- | --- |
| Mô tả | Use case cho phép khách hàng đặt hàng trên hệ thống |
| Actor chính | Khách hàng |
| Actor phụ | Không có |
| Tiền điều kiện | Khách hàng phải đăng nhập thành công vào hệ thống |
| Luồng hoạt động | 1. Khách hàng nhấn vào mua hàng  2. Hệ thống thêm sản phẩm vào giỏ hàng và chuyển đến trang giỏ hàng.  3. Khách hàng chỉnh sửa số lượng và kích thước sản phẩm cần mua nếu cần và chọn phương thức thanh toán. Khách hàng cũng có thể thay đổi số điện thoại và địa chỉ nhận hàng.  4. Nếu thành công thì hệ thống sẽ hiển thị thông báo đặt hàng thành công, ngược lại hệ thống sẽ hiển thị thông báo đặt hàng thất bại.  5. Kết thúc use case. |

Bảng .: Đặc tả usecase đặt hàng

#### Usecase tra cứu

|  |  |
| --- | --- |
| Mô tả | Use case cho phép khách hàng tra cứu các sản phẩm theo tên sản phẩm, loại sản phẩm, nhãn hàng, xuất xứ. |
| Actor chính | Khách hàng |
| Actor phụ | Không có |
| Luồng hoạt động | 1. Khách hàng nhập từ khóa trên thanh tìm kiếm và nhấn tìm kiếm.  2. Hệ thống sẽ tìm những sản phẩm có tên sản phẩm, loại sản phẩm, nhãn hiệu hoặc xuất xứ trùng với từ khóa khách hàng nhập vào và hiển thị ở trang sản phẩm. Nếu không tìm thấy thì sẽ hiển thị thông báo không tìm thấy sản phẩm.  3. Kết thúc use case. |

Bảng .: Đặc tả usecase tra cứu

#### Usecase phản hồi đánh giá

|  |  |
| --- | --- |
| Mô tả | Use case cho phép khách hàng bình luận bằng cách viết và có thể thêm hình ảnh nếu có và đánh giá sản phẩm theo sao. |
| Actor chính | Khách hàng |
| Actor phụ | Không có |
| Tiền điều kiện | Khách hàng phải đăng nhập thành công vào hệ thống. |
| Luồng hoạt động | 1. Hệ thống hiển thị màn hình phản hồi và đánh giá sản phẩm.  2. Khách hàng viết bình luận và đánh giá sản phẩm bằng cách chọn sao. Khách hàng có thể thêm ảnh nếu có.  3. Hệ thống hiển thị trang chi tiết sản phẩm với những đánh giá sản phẩm của khách hàng.  5. Kết thúc use case. |

Bảng .: Đặc tả usecase phản hồi đánh giá

#### Usecase quản lý tài khoản

|  |  |
| --- | --- |
| Mô tả | Use case cho khách hàng dùng xem và cập nhật thông tin cá nhân. |
| Actor chính | Khách hàng |
| Actor phụ | Không có |
| Tiền điều kiện | Khách hàng phải đăng nhập thành công vào hệ thống |
| Luồng hoạt động | 1. Hệ thống hiển thị màn hình trang cá nhân. Ở trang này, có các tùy chọn chức năng để khách hàng có thể thực hiện quản lý tài khoản như: Hồ sơ (xem và chỉnh sửa thông tin cá nhân), Địa chỉ (xem và thay đổi địa chỉ), Đổi mật khẩu.  2. Khách hàng chọn một trong các tùy chọn trên thì một trong các luồng tương ứng sau sẽ thực hiện:  - Nếu khách hàng chọn “Hồ sơ” thì luồng phụ **Hồ sơ** được thực hiện.  - Nếu khách hàng chọn “Địa chỉ” thì luồng phụ **Địa chỉ** được thực hiện.  - Nếu khách hàng chọn “Đổi mật khẩu” thì luồng phụ **Đổi mật khẩu** được thực hiện.  \* Các luồng phụ:  **I – Hồ sơ**  1. Hệ thống hiển thị thông tin của khách hàng.  2. Nếu khách hàng cập nhật thông tin và chọn lưu thông tin, hệ thống sẽ cập nhật lại thông tin khách hàng và hiển thị thông báo cập nhật thành công. Ngược lại, thông tin khách hàng vẫn giữ nguyên.  **II – Địa chỉ**  1. Hệ thống hiển thị thông tin địa chỉ của khách hàng.  2. Nếu khách hàng cập nhật địa chỉ thì hệ thống cập nhật lại thông tin địa chỉ của khách hàng, ngược lại địa chỉ của khách hàng vẫn giữ nguyên.  **III – Đổi mật khẩu**  1. Hệ thống hiển thị giao diện đổi mật khẩu.  2. Khách hàng nhập mật khẩu hiện tại và mật khẩu mới và chọn lưu thay đổi thì hệ thống sẽ tiến hành cập nhật lại mật khẩu của khách hàng, ngược lại mật khẩu của khách hàng vẫn giữ nguyên. |

Bảng .: Đặc tả usecase quản lý tài khoản

#### Usecase thanh toán

|  |  |
| --- | --- |
| Mô tả | Use case cho phép khách hàng thanh toán khi mua hàng |
| Actor chính | Khách hàng |
| Actor phụ | Momo |
| Tiền điều kiện | Khách hàng phải đăng nhập thành công vào hệ thống |
| Luồng hoạt động | 1. Hệ thống yêu cầu người dùng chọn phương thức thanh toán.  2. Khách hàng chọn phương thức thanh toán.  3. Nếu khách hàng chọn “Thanh toán khi nhận hàng” thì luồng phụ Thanh toán khi nhận hàng sẽ được thực hiện. Nếu hệ thống chọn “Thanh toán Momo” thì luồng phụ Thanh toán Momo được thực hiện  \* Các luồng phụ:  **I – Thanh toán khi nhận hàng**  Usecase được thực hiện khi khách hàng chọn đặt hàng.  1. Nếu thành công, hệ thống sẽ hiển thị thông báo đặt hàng thành công. Ngược lại, sẽ thông báo đặt hàng thất bại.  2. Kết thúc usecase.  **II – Thanh toán Momo**  Use case này được thực hiện khi khách hàng chọn đặt hàng.  1. Hệ thống sẽ liên lạc với Hệ thống Momo tạo mã QR và hiển thị lên màn hình. Khách hàng quét mã QR này để thanh toán. Hệ thống gửi thông báo cho khách hàng rằng đơn hàng đã tạo thành công và thông báo cho admin và nhân viên có một đơn hàng được tạo để admin và nhân viên biết và duyệt đơn hàng.  2. Kết thúc usecase. |

Bảng .: Đặc tả usecase thanh toán

#### Usecase quản lý giỏ hàng

|  |  |
| --- | --- |
| Mô tả | Use case này cho phép khách hàng thực hiện các chức năng quản lý giỏ hàng, bao gồm các chức năng: thêm vào giỏ hàng, xem thông tin giỏ hàng, cập nhật giỏ hàng, xóa khỏi giỏ hàng |
| Actor chính | Khách hàng |
| Actor phụ | Không có |
| Tiền điều kiện | Khách hàng phải đăng nhập thành công vào hệ thống |
| Luồng hoạt động | 1. Khách hàng truy cập vào trang Giỏ hàng bằng cách nhấp chọn vào biểu tượng giỏ hàng.  2. Hệ thống sẽ hiển thị danh sách các sản phẩm đã được thêm vào giỏ, khi đó luồng phụ **Xem thông tin giỏ hàng** được thực hiện.  - Nếu giỏ hàng trống thì hệ thống sẽ hiển thị thông báo không có sản phẩm trong giỏ. Nếu khách hàng chọn “Tiếp tục mua sắm” thì hệ thống sẽ chuyển đến trang Sản phẩm, ngược lại hệ thống sẽ không thay đổi, use case kết thúc.  - Nếu có sản phẩm trong giỏ hàng, hệ thống sẽ hiển thị danh sách các sản phẩm trong giỏ hàng mà khách hàng đã thêm trước đó.  3. Nếu khách hàng đang ở trang Sản phẩm, khách hàng nhấp chọn “Thêm vào giỏ hàng” thì luồng phụ **Thêm vào giỏ hàng** sẽ được thực hiện. Nếu khách hàng đang xem danh sách sản phẩm trong giỏ, khách hàng chọn xóa sản phẩm khỏi giỏ thì luồng phụ **Xóa khỏi giỏ hàng** sẽ được thực hiện hoặc khách hàng chọn cập nhật số lượng sản phẩm trong giỏ thì luồng phụ **Cập nhật giỏ hàng** được thực hiện.  \* Các luồng phụ:  **I – Xem thông tin giỏ hàng**  Use case bắt đầu khi khách hàng chọn chức năng Quản lý giỏ hàng, hệ thống sẽ hiển thị danh sách các sản phẩm mà khách hàng đã thêm vào giỏ hàng trước đó.  **II – Thêm vào giỏ hàng**  Use case bắt đầu khi khách hàng nhấp chọn “Thêm vào giỏ hàng” của một sản phẩm cụ thể, hệ thống sẽ thêm sản phẩm vào giỏ hàng và thông báo cho khách hàng đã thêm vào giỏ hàng thành công.  **III – Cập nhật giỏ hàng**  1. Khách hàng nhấp chọn vào biểu tượng cập nhật của một sản phẩm trong giỏ hàng.  2. Hệ thống hiển thị giao diện cập nhật để khách hàng chỉnh sửa các thông tin.  3. Khách hàng chỉnh sửa size và số lượng sản phẩm nếu cần và chọn cập nhật.  4. Hệ thống cập nhật lại số lượng và size của sản phẩm đó trong giỏ nếu có chỉnh sửa và chuyển đến trang giỏ hàng. |

Bảng .: Đặc tả usecase quản lý giỏ hàng

#### Usecase quên mật khẩu

|  |  |
| --- | --- |
| Mô tả | Use case cho phép khách hàng đặt lại mật khẩu mới khi quên mật khẩu. |
| Actor chính | Khách hàng |
| Actor phụ | Không có |
| Tiền điều kiện | Khách hàng phải có tài khoản trên hệ thống. |
| Hậu điều kiện | Khách hàng đăng nhập để sử dụng một số chức năng. |
| Luồng hoạt động | 1. Hệ thống hiển thị giao diện để khách hàng nhập email đăng ký lấy lại mật khẩu.  2. Khách hàng nhập email đăng ký và chọn tiếp tục hoặc có thể chọn đăng nhập để quay về trang đăng nhập.  3. Nếu khách hàng chọn tiếp tục, hệ thống sẽ kiểm tra email vừa nhập có tồn tại với một trong những email đã đăng ký hay không. Nếu có, hệ thống sẽ gửi một mail đến email đó và hiển thị thông báo đã gửi mail. Ngược lại sẽ không gửi mail.  4. Khách hàng nhấp vào đường dẫn trong mail sẽ dẫn đến một trang cho phép khách hàng lấy lại mật khẩu. Khách hàng nhập mật khẩu mới và chọn đặt lại mật khẩu.  5. Hệ thống sẽ xóa mật khẩu cũ và lưu mật khẩu mới của khách hàng vừa đặt.  6. Kết thúc use case. |

Bảng .: Đặc tả usecase quên mật khẩu

#### Usecase xem thông tin sản phẩm

|  |  |
| --- | --- |
| Mô tả | Use case cho phép khách hàng xem thông tin sản phẩm |
| Actor chính | Khách hàng |
| Actor phụ | Không có |
| Luồng hoạt động | 1. Khách hàng nhấp chọn xem trên sản phẩm  2. Hệ thống sẽ chuyển đến trang chi tiết sản phẩm và hiển thị thông tin của sản phẩm đó.  3. Kết thúc use case. |

Bảng .: Đặc tả usecase xem thông tin sản phẩm

#### Usecase xem lịch sử mua hàng

|  |  |
| --- | --- |
| Mô tả | Use case cho phép khách hàng xem thông tin lịch sử mua hàng. |
| Actor chính | Khách hàng |
| Actor phụ | Không có |
| Tiền điều kiện | Khách hàng đã đăng nhập thành công vào hệ thống. |
| Luồng hoạt động | 1. Khách hàng nhấp chọn lịch sử mua hàng  2. Hệ thống sẽ chuyển đến trang lịch sử mua hàng và hiển thị thông tin các đơn hàng mà khách hàng đã mua.  3. Kết thúc use case. |

Bảng .: Đặc tả usecase lịch sử mua hàng

#### Usecase quản lý nhân viên

|  |  |
| --- | --- |
| Mô tả | Use case cho phép admin thực hiện các chức năng quản lý nhân viên. |
| Actor chính | Admin |
| Actor phụ | Không có |
| Tiền điều kiện | Admin đã đăng nhập thành công vào hệ thống |
| Luồng hoạt động | 1. Admin chọn chức năng quản lý nhân viên  2. Hệ thống sẽ hiển thị các tùy chọn chức năng gồm danh sách nhân viên, thêm nhân viên.  3. Nếu admin chọn “Thêm nhân viên” thì luồng phụ **Thêm nhân viên** sẽ được thực hiện. Nếu admin chọn “Danh sách nhân viên” thì hệ thống chuyển đến trang nhân viên và hiển thị danh sách nhân viên. Với mỗi nhân viên, ta có thể xem thông tin, sửa, thêm địa chỉ, xóa nhân viên.  - Nếu admin chọn biểu tượng sửa thì luồng phụ **Sửa thông tin nhân viên** sẽ được thực hiện.  - Nếu admin chọn biểu tượng xem thông tin thì luồng phụ **Xem thông tin nhân viên** sẽ được thực hiện.  - Nếu admin chọn thêm địa chỉ thì luồng phụ **Thêm địa chỉ** sẽ được thực hiện.  - Nếu admin chọn biểu tượng xóa thì luồng phụ **Xóa nhân viên** sẽ được thực hiện.  \* Các luồng phụ:  **I – Sửa thông tin nhân viên**  1. Hệ thống hiển thị các thông tin nhân viên cần chỉnh sửa.  2. Admin chỉnh sửa các thông tin của nhân viên nếu có và chọn lưu thay đổi.  3. Hệ thống cập nhật lại thông tin nhân viên và trả về trang danh sách nhân viên.  **II – Xem thông tin nhân viên**  Usecase này được thực hiện khi admin nhấp vào biểu tượng xem thông tin. Hệ thống sẽ hiển thị thông tin chi tiết của nhân viên cần xem.  **III – Thêm địa chỉ**  1. Hệ thống hiển thị giao diện để admin thêm địa chỉ cho nhân viên.  2. Admin chọn tên tỉnh thành, quận huyện, và điền địa chỉ cụ thể của nhân viên.  3. Hệ thống cập nhật địa chỉ của nhân viên.  **IV – Xóa nhân viên**  Usecase này được thực hiện khi admin chọn biểu tượng xóa. Hệ thống sẽ hỏi admin có chắc chắn xóa hay không. Nếu admin nhấp chọn xóa thì thuộc tính Is\_active của sản phẩm sẽ chuyển thành false. Nếu admin chọn quay lại thì sẽ chuyển đến trang danh sách sản phẩm. |

Bảng .: Đặc tả usecase quản lý nhân viên

#### Usecase quản lý sản phẩm

|  |  |
| --- | --- |
| Mô tả | Use case cho phép admin thực hiện các chức năng thêm, xóa và cập nhật thông tin sản phẩm của quản lý sản phẩm |
| Actor chính | Admin |
| Actor phụ | Không có |
| Tiền điều kiện | Admin đã đăng nhập thành công vào hệ thống |
| Luồng hoạt động | 1. Admin chọn chức năng quản lý sản phẩm  2. Hệ thống sẽ hiển thị các tùy chọn chức năng gồm danh mục sản phẩm, sản phẩm.  3. Nếu admin chọn “Danh mục sản phẩm” thì hệ thống sẽ chuyển đến trang danh mục sản phẩm và trang này hiển thị tất cả các loại sản phẩm. Tại đây, admin có thể tiến hành thêm danh mục, với mỗi danh mục admin có thể xóa, sửa hoặc xem danh sách sản phẩm theo từng danh mục sản phẩm.  Nếu admin chọn “Sản phẩm” thì hệ thống sẽ chuyển đến trang sản phẩm và trang này sẽ hiển thị tất cả các sản phẩm. Tại đây, admin có thể tiến hành thêm sản phẩm, với mỗi sản phẩm admin có thể tiến hành sửa, xóa, thêm hình ảnh, kích thước hoặc xem chi tiết sản phẩm. |

Bảng .: Đặc tả usecase quản lý sản phẩm

#### Usecase quản lý khuyến mãi

|  |  |
| --- | --- |
| Mô tả | Use case cho phép admin thực hiện các chức năng quản lý khuyến mãi như thêm, xóa, sửa. |
| Actor chính | Admin |
| Actor phụ | Không có |
| Tiền điều kiện | Admin đã đăng nhập thành công vào hệ thống. |
| Luồng hoạt động | 1. Admin chọn chức năng quản lý khuyến mãi  2. Hệ thống sẽ hiển thị các tùy chọn chức năng gồm chương trình khuyến mãi, thêm khuyến mãi.  3. Nếu admin chọn “Chương trình khuyến mãi” thì hệ thống sẽ chuyển đến trang danh sách khuyến mãi và trang này sẽ hiển thị tất cả các khuyến mãi. Tại đây, với từng khuyến mãi admin có thể tiến hành chỉnh sửa hoặc xóa.  Nếu admin chọn “Thêm khuyến mãi” thì luống phụ **Thêm khuyến mãi** sẽ được thực hiện.  \* Các luồng phụ:  **I – Thêm khuyến mãi**  1. Hệ thống hiển thị giao diện thêm khuyến mãi.  2. Admin nhập các thông tin của khuyến mãi và chọn thêm ưu đãi.  3. Hệ thống tiến hành lưu thông tin của khuyến mãi.  4. Kết thúc usecase. |

Bảng .: Đặc tả usecase quản lý khuyến mãi

#### Usecase thống kê

|  |  |
| --- | --- |
| Mô tả | Use case cho phép admin thống kê doanh thu theo năm và theo tháng. |
| Actor chính | Admin |
| Actor phụ | Không có |
| Tiền điều kiện | Admin đã đăng nhập thành công vào hệ thống. |
| Luồng hoạt động | 1. Khách hàng nhấp chọn lịch sử mua hàng  2. Hệ thống sẽ hiển thị các tùy chọn chức năng gồm thống kê theo tháng, thống kê theo năm.  3. Nếu admin chọn “Thống kê theo tháng” thì luồng phụ **Thống kê theo tháng** được thực hiện. Nếu admin chọn “Thống kê theo năm” thì luồng phụ **Thống kê theo năm** được thực hiện.  \* Các luồng phụ:  **I – Thống kê theo tháng**  1. Hệ thống hiển thị trang màn hình thống kê theo tháng, mặc định là tháng hiện tại.  2. Admin chọn tháng cần thống kê.  3. Hệ thống hiển thị thống kê doanh thu, lợi nhuận theo tháng mà admin yêu cầu.  **II – Thống kê theo năm**  1. Hệ thống hiển thị trang màn hình thống kê theo năm, mặc định là năm hiện tại.  2. Admin chọn năm cần thống kê.  3. Hệ thống hiển thị thống kê doanh thu, lợi nhuận theo năm mà admin yêu cầu |

Bảng .: Đặc tả usecase thống kê

#### Usecase tìm kiếm

|  |  |
| --- | --- |
| Mô tả | Use case cho phép admin và nhân viên tìm kiếm sản phẩm theo tên sản phẩm, loại sản phẩm, nhãn hiệu, xuất xứ. |
| Actor chính | Admin, nhân viên |
| Actor phụ | Không có |
| Tiền điều kiện | Admin hoặc nhân viên đã đăng nhập thành công vào hệ thống. |
| Luồng hoạt động | 1. Admin hoặc nhân viên nhập từ khóa trên thanh tìm kiếm của trang sản phẩm dành cho quản lý và nhấn tìm kiếm.  2. Hệ thống sẽ tìm những sản phẩm có tên sản phẩm, loại sản phẩm, nhãn hiệu hoặc xuất xứ trùng với từ khóa mà admin hay nhân viên nhập vào và hiển thị ở trang sản phẩm. Nếu không tìm thấy thì sẽ không trả về sản phẩm nào.  3. Kết thúc use case. |

Bảng .: Đặc tả usecase tìm kiếm của admin và nhân viên

#### Usecase quản lý kho

|  |  |
| --- | --- |
| Mô tả | Use case cho phép admin và nhân viên lượng hàng trong kho và nhập hàng. |
| Actor chính | Admin, nhân viên |
| Actor phụ | Không có |
| Tiền điều kiện | Admin hoặc nhân viên đã đăng nhập thành công vào hệ thống. |
| Luồng hoạt động | 1. Admin hoặc nhân viên chọn chức năng quản lý kho.  2. Hệ thống sẽ hiển thị các tùy chọn chức năng gồm số lượng hàng tồn và nhập kho.  3. Nếu admin hoặc nhân viên chọn “Số lượng hàng tồn” thì luồng phụ **Số lượng hàng tồn** sẽ được thực hiên. Nếu admin hoặc nhân viên chọn “Nhập kho” thì luông phụ **Nhập kho** sẽ được thực hiện.  \* Các luồng phụ:  **I – Số lượng hàng tồn**  Usecase được thực hiện khi admin hoặc nhân viên nhấp chọn số lượng hàng tồn. Hệ thống sẽ chuyển đến trang số lượng hàng tồn và trang này hiển thị số lượng hàng tồn của từng sản phẩm.  **II – Nhập kho**  1. Hệ thống hiển thị giao diện nhập kho.  2. Admin hoặc nhân viên cung cấp các thông tin yêu cầu như tên sản phẩm, số lượng nhập, đơn giá.  3. Hệ thống sẽ cộng dồn số lượng nhập vào số lượng có trong kho của sản phẩm đó. |

Bảng .: Đặc tả usecase quản lý kho

#### Usecase xem thông tin phản hồi

|  |  |
| --- | --- |
| Mô tả | Use case cho phép admin và nhân viên xem thông tin phản hồi của khách hàng. |
| Actor chính | Admin, nhân viên |
| Actor phụ | Không có |
| Tiền điều kiện | Admin hoặc nhân viên đã đăng nhập thành công vào hệ thống. |
| Luồng hoạt động | 1. Hệ thống hiển thị các phản hồi của khách hàng nếu có phản hồi, ngược lại sẽ hiển thị chưa có phản hồi.  3. Kết thúc use case. |

Bảng .: Đặc tả usecase xem thông tin phản hồi

#### Usecase quản lý khách hàng

|  |  |
| --- | --- |
| Mô tả | Use case cho phép admin và nhân viên xem danh sách khách hàng, với mỗi khách hàng admin và nhân viên có thể xem thông tin và lịch sử mua hàng của khách hàng. |
| Actor chính | Admin, nhân viên |
| Actor phụ | Không có |
| Tiền điều kiện | Admin hoặc nhân viên đã đăng nhập thành công vào hệ thống. |
| Luồng hoạt động | 1. Admin hoặc nhân viên chọn chức năng quản lý khách hàng  2. Hệ thống sẽ hiển thị tùy chọn chức năng là danh sách khách hàng.  3. Admin hoặc nhân viên chọn danh sách khách hàng, hệ thống chuyển đến trang danh sách khách hàng và trang này hiển thị danh sách tất các khách hàng của shop. Tại trang này, admin và nhân viên có thể xem thông tin và lịch sử mua hàng của từng khách hàng.  - Nếu admin hoặc nhân viên chọn biểu tượng xem thông tin thì luồng phụ **Xem thông tin khách hàng** được thực hiện.  - Nếu asmin hoặc nhân viên chọn “Lịch sử mua hàng” thì luồng phụ **Lịch sử mua hàng** được thực hiện.  \* Các luồng phụ:  **I – Xem thông tin khách hàng**  Usecase này được thực hiện khi admin hoặc nhân viên chọn vào biểu tượng xem thông tin của một khách hàng. Hệ thống sẽ hiển thị các thông tin của khách hàng cần xem.  **II – Lịch sử mua hàng**  Usecase này được thực hiện khi admin hoặc nhân viên chọn “Lịch sử mua hàng” của một khách hàng. Hệ thống sẽ hiển thị các thông tin các đơn hàng mà khách hàng đó đã mua nếu có, nếu không thì sẽ hiển thị chưa có đơn hàng nào được tạo. |

Bảng .: Đặc tả usecase quản lý khách hàng

#### Usecase xem thông tin đặt hàng

|  |  |
| --- | --- |
| Mô tả | Use case cho phép admin và nhân viên xem thông tin đặt hàng của khách hàng. |
| Actor chính | Admin, nhân viên |
| Actor phụ | Không có |
| Tiền điều kiện | Admin hoặc nhân viên đã đăng nhập thành công vào hệ thống. |
| Luồng hoạt động | 1. Admin hoặc nhân viên chọn chức năng thông tin đặt hàng.  2. Hệ thống sẽ hiển thị các tùy chọn chức năng gồm danh sách đơn hàng và đơn hàng chưa duyệt.  3. Nếu admin hoặc nhân viên chọn “Danh sách đơn hàng” thì luồng phụ Danh sách đơn hàng được thực hiện. Nếu admin hoặc nhân viên chọn “Đơn hàng chưa duyệt” thì luồng phụ Duyệt đơn hàng sẽ được thực hiện.  \* Các luồng phụ:  **I – Danh sách đơn hàng**  Usecase này được thực hiện khi admin hoặc nhân viên chọn “Danh sách đơn hàng”. Hệ thống sẽ chuyển đến trang danh sách đơn hàng và trang này hiển thị danh sách các đơn hàng của các khách hàng nếu có. Ngược lại sẽ hiển thị chưa có đơn hàng nào được tạo.  **II – Duyệt đơn hàng**  1. Hệ thống hiển thị danh sách các đơn hàng chưa được duyệt của các khách hàng. Tại đây, admin và nhân viên có thể xem thông tin đơn hàng và duyệt đơn hàng.  2. Nếu admin hoặc nhân viên chọn vào biểu tượng xem thông tin thì hệ thống sẽ chuyển đến trang thông tin đơn hàng và hiển thị thông tin của đơn hàng ở trang này. Nếu admin hoặc nhân viên chọn vào biểu tượng duyệt đơn hàng thì thuộc tính Is\_check của đơn hàng sẽ chuyển thành true và đơn hàng đã được duyệt. |

Bảng .: Đặc tả usecase xem thông tin đặt hàng

### Sơ đồ tuần tự

## Thiết kế cơ sở dữ liệu

## Thực nghiệm hệ thống

# KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN ĐỀ TÀI

## Kết luận

## Hướng phát triển đề tài

TÀI LIỆU THAM KHẢO

[1] TutorialsTeacher, “.NET Core Overview,”. [Trực tuyến]. Địa chỉ: <https://www.tutorialsteacher.com/core/dotnet-core> [Truy cập 5/10/2021]

[2] Trọng Đức, “Phân biệt .NET Core, .NET Framework, .NET Standard,” 18/11/2019. [Trực tuyến]. Địa chỉ: <https://tuhocict.com/phan-biet-net-cor-net-framework-net-standard/> [Truy cập 5/10/2021]

[3] Nguyễn Minh Tuấn, “Tổng quan về ASP.NET Core,”. [Trực tuyến]. Địa chỉ: <https://netcore.vn/bai-viet/tong-quan-ve-aspnet-core> [Truy cập 6/10/2021]

[4] Steve Smith, “Overview of ASP.NET Core MVC,” 12/2/2020. [Trực tuyến]. Địa chỉ: <https://docs.microsoft.com/en-us/aspnet/core/mvc/overview?view=aspnetcore-5.0> [Truy cập 6/10/2021]

[5] Bạch Ngọc Toàn, “Cơ chế Routing trong ASP.NET Core,” 7/7/2019. [Trực tuyến]. Địa chỉ: <https://tedu.com.vn/lap-trinh-aspnet-core/co-che-routing-trong-aspnet-core-227.html> [Truy cập 6/10/2021]

[6] Bạch Ngọc Toàn, “Cơ chế Model Binding: Truyền dữ liệu từ View lên Controller,” 13/8/2019. [Trực tuyến]. Địa chỉ: <https://tedu.com.vn/lap-trinh-aspnet-core/co-che-model-binding-truyen-du-lieu-tu-view-len-controller-252.html> [Truy cập 7/10/2021]

[7] Bạch Ngọc Toàn, “Model Validation trong ASP.NET Core,” 19/8/2019. [Trực tuyến]. Địa chỉ: <https://tedu.com.vn/lap-trinh-aspnet-core/model-validation-trong-aspnet-core-253.html> [Truy cập 7/10/2021]

[8] Admin, “Dependency Injection là gì?,” 7/2019. [Trực tuyến]. Địa chỉ: <https://www.dotnetcoban.com/2019/07/introduction-to-dependency-injection.html> [Truy cập 8/10/2021]

[9] Bạch Ngọc Toàn, “Cơ chế Dependency Injection trong ASP.NET Core,” 20/8/2019. [Trực tuyến]. Địa chỉ: <https://tedu.com.vn/lap-trinh-aspnet-core/co-che-dependency-injection-trong-aspnet-core-256.html> [Truy cập 8/10/2021]

[10] Rick Anderson, “Introduction to Identity on ASP.NET Core,” 15/9/2021. [Trực tuyến]. Địa chỉ: <https://docs.microsoft.com/en-us/aspnet/core/security/authentication/identity?view=aspnetcore-5.0&tabs=visual-studio> [Truy cập 9/10/2021]

[11] Piotr Malek, “Send and Receive Emails in ASP.NET C#,” 27/7/2019. [Trực tuyến]. Địa chỉ: <https://mailtrap.io/blog/send-emails-in-net/> [Truy cập 11/10/2021]

PHỤ LỤC