TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM

**TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

**KHOA CÔNG NGHỆ THÔNG TIN**

Logo, company name

Description automatically generated

**BÁO CÁO QUÁ TRÌNH 1**

**CÔNG NGHỆ .NET**

**WINFORM VÀ .NET CORE**

*Người hướng dẫn*: **TS PHẠM THÁI KỲ TRUNG**

*Người thực hiện*: **Đỗ Minh Quân – 521H0290**

*Khoá:* 25

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2023**

TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM

**TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

**KHOA CÔNG NGHỆ THÔNG TIN**

Logo, company name

Description automatically generated

**BÁO CÁO QUÁ TRÌNH 1**

**CÔNG NGHỆ .NET**

**WINFORM VÀ .NET CORE**

*Người hướng dẫn*: **TS PHẠM THÁI KỲ TRUNG**

*Người thực hiện*: **Đỗ Minh Quân – 521H0290**

*Khoá*  **:25**

**THÀNH PHỐ HỒ CHÍ MINH, NĂM 2023**

# LỜI CẢM ƠN

Trước hết, với tình cảm sâu sắc và chân thành nhất, cho phép tôi được bày tỏ lòng biết ơn đến tất cả các cá nhân và tổ chức đã tạo điều kiện hỗ trợ, giúp đỡ tôi trong suốt quá trình học tập và nghiên cứu đề tài này. Trong suốt thời gian từ khi bắt đầu học tập tại trường đến nay, tôi đã nhận được rất nhiều sự quan tâm, giúp đỡ của Thầy và bạn bè.

Với lòng biết ơn sâu sắc nhất, tôi xin gửi đến quý Thầy Cô ở Khoa Công nghệ thông tin đã truyền đạt vốn kiến thức quý báu cho chúng tôi trong suốt thời gian học tập tại trường. Nhờ có những lời hướng dẫn, dạy bảo của các thầy cô nên đề tài nghiên cứu của tôi mới có thể hoàn thiện tốt đẹp.

Một lần nữa, tôi xin chân thành cảm ơn thầy–Phạm Thái Kỳ Trung người đã trực tiếp giúp đỡ, quan tâm, hướng dẫn tôi hoàn thành tốt bài báo cáo này trong thời gian qua.

Bài báo cáo cuối kì thực hiện trong khoảng thời gian gần 2 tuần. Bước đầu đi vào thực tế của tôi còn hạn chế và còn nhiều bỡ ngỡ nên không tránh khỏi những thiếu sót, tôi rất mong nhận được những ý kiến đóng góp quý báu của quý Thầy để kiến thức của tôi trong lĩnh vực này được hoàn thiện hơn đồng thời có điều kiện bổ sung, nâng cao ý thức của mình.

Tôi xin chân thành cảm ơn !

*TP. Hồ Chí Minh, ngày 10 tháng 9 năm 2023*

*Author*

*(Sign and write full name)*



*Đỗ Minh Quân*

**BÁO CÁO ĐƯỢC HOÀN THÀNH**

**TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG**

Tôi xin cam đoan đây là sản phẩm đồ án của riêng tôi và được sự hướng dẫn của thầy Phạm Thái Kỳ Trung. Các nội dung nghiên cứu, kết quả trong đề tài này là trung thực và chưa công bố dưới bất kỳ hình thức nào trước đây. Những số liệu trong các bảng biểu phục vụ cho việc phân tích, nhận xét, đánh giá được chính tác giả thu thập từ các nguồn khác nhau có ghi rõ trong phần tài liệu tham khảo.

Ngoài ra, trong đồ án còn sử dụng một số nhận xét, đánh giá cũng như số liệu của các tác giả khác, cơ quan tổ chức khác đều có trích dẫn và chú thích nguồn gốc.

**Nếu phát hiện có bất kỳ sự gian lận nào chúng tôi xin hoàn toàn chịu trách nhiệm về nội dung đồ án của mình.** Trường đại học Tôn Đức Thắng không liên quan đến những vi phạm tác quyền, bản quyền do chúng tôi gây ra trong quá trình thực hiện (nếu có).

*TP. Hồ Chí Minh, ngày 20 tháng 10 năm 2023*

*Tác giả*

*(ký tên và ghi rõ họ tên)*



*Đỗ Minh Quân*

# PHẦN XÁC NHẬN VÀ ĐÁNH GIÁ CỦA GIẢNG VIÊN

**Phần xác nhận của GV hướng dẫn**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Tp. Hồ Chí Minh, ngày tháng năm

(kí và ghi họ tên)

**Phần đánh giá của GV chấm bài**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Tp. Hồ Chí Minh, ngày tháng năm

(kí và ghi họ tên)

# TÓM TẮT

Đây là báo cáo về môn Công nghệ .NET của Khoa Công nghệ Thông tin, Đại học Tôn Đức Thắng. Nhờ sự hỗ trợ của thầy Phạm Thái Kỳ Trung mà báo cáo này đã được hoàn thành vào ngày 20 tháng 10 năm 2023. Báo cáo cơ bản về Winform và ASP.NET với mssql để hoàn thành 1 ứng dụng và 1 web đầy đủ các chức năng thiết yếu như đăng nhập, CRUD, filter.

# MỤC LỤC

[LỜI CẢM ƠN i](#_Toc148800683)

[PHẦN XÁC NHẬN VÀ ĐÁNH GIÁ CỦA GIẢNG VIÊN iii](#_Toc148800684)

[TÓM TẮT iv](#_Toc148800685)

[MỤC LỤC 1](#_Toc148800686)

[CHƯƠNG 1 – GIỚI THIỆU YÊU CẦU ĐỀ BÀI 3](#_Toc148800687)

[1.1 Giới thiệu các chức năng. 3](#_Toc148800688)

[1.1.1 Mô tả ứng dụng và thang diểm. 3](#_Toc148800689)

[1.1.2 Windows Application Project. 3](#_Toc148800690)

[1.1.3 ASP.Net Core with Razor project. 3](#_Toc148800691)

[CHƯƠNG 2 – WINFORM APPLICATION AND ASP.NET CORE 3](#_Toc148800692)

[2.1. Tạo cở sở dữ liệu hoàn chỉnh. 4](#_Toc148800693)

[2.2. Winform Application. 4](#_Toc148800694)

[2.2.1. Xử lý đăng nhập. 4](#_Toc148800695)

[2.2.2. Tạo mainform sau khi xác thực đăng nhập xong thì chuyển đến main form. 7](#_Toc148800696)

[2.2.3. Thêm sản phẩm(Products) và đọc dữ liệu sản phẩm. Dưới đây là giải thích cho từng phương thức trong mã trên: 9](#_Toc148800697)

[2.2.4 Thêm khách hàng(Customers) và đọc dữ liệu khác hàng. 15](#_Toc148800698)

[2.2.5 Thêm đơn hàng(Orders) và đọc dữ liệu đơn hàng. 19](#_Toc148800699)

[2.2.6 Tạo báo cáo để in đơn hàng 26](#_Toc148800700)

[2.2.6. Tạo một form để lọc các mặt hàng tốt nhất, các mặt hàng được khách hàng mua, khách hàng mua các mặt hàng 28](#_Toc148800701)

[2.3 ASP.NET CORE 33](#_Toc148800702)

[2.3.1. Tạo cở sở dữ liệu hoàn chỉnh. 33](#_Toc148800703)

[2.3.2. API 34](#_Toc148800704)

[2.3.2.1 34](#_Toc148800705)

[CHƯƠNG 4 TÀI LIỆU THAM KHẢO 34](#_Toc148800706)

CHƯƠNG 1 – GIỚI THIỆU YÊU CẦU ĐỀ BÀI

* 1. Giới thiệu các chức năng.
     1. Mô tả ứng dụng và thang diểm.

Để cho phép người dùng nhập dữ liệu về các mặt hàng và khách hàng, và cung cấp khả năng nhập đơn hàng khi khách hàng muốn đặt hàng, trong đó một khách hàng có thể đặt nhiều mặt hàng trong một đơn hàng.

Bạn có thể sử dụng các bảng dữ liệu sau:

• Item(ItemID, ItemName, Size,…..)

• Customer(CustID,CustName, Address)

• Order(OrderID, OrderDate, CustID)

• OrderDetail(ID, OrderID, ItemID, Quantity, UnitAmount)

**• Database: 1.5 điểm**

**• Winform: 3.5 điểm**

**• ASP.Net Core with Razor: 5 điểm**

* + 1. Windows Application Project.

1. Form Đăng nhập đến Form Chính (nên sử dụng menu trips)

2. Tạo Windows Form để thêm sản phẩm, khách hàng và một form đặt hàng (mối quan hệ một-nhiều) (Đơn hàng và Chi tiết đơn hàng)

3. Tạo báo cáo để in đơn hàng

4. Tạo một form để lọc các mặt hàng tốt nhất, các mặt hàng được khách hàng mua, khách hàng mua các mặt hàng

* + 1. ASP.Net Core with Razor project.

1. Chức năng đăng nhập, xác nhận email

2. Tạo biểu mẫu web để thêm sản phẩm, đại lý và biểu mẫu đơn hàng (một-nhiều) (Đơn hàng và Chi tiết đơn hàng)

3. Tạo một nút trên biểu mẫu Đơn hàng sau đó liên kết đến biểu mẫu đơn hàng (để in) hoặc báo cáo

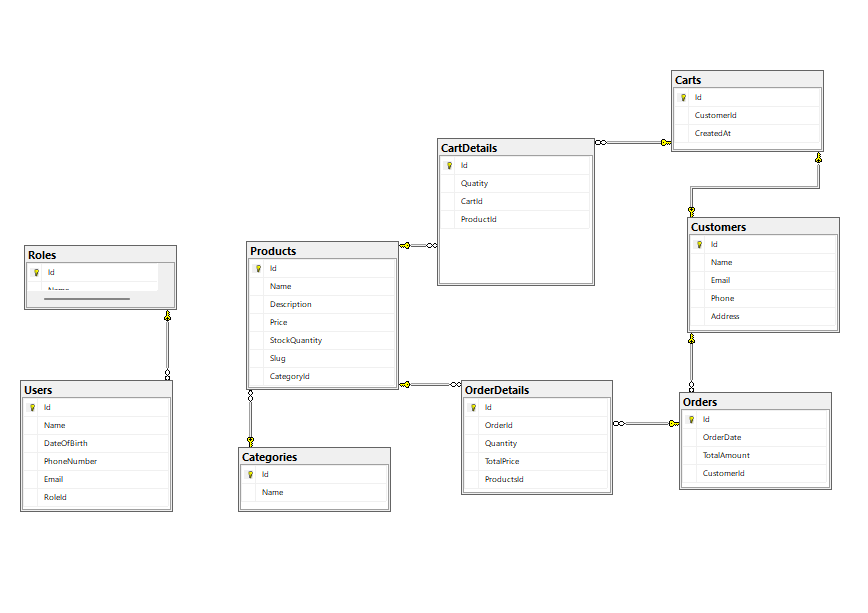
4. Tạo một biểu mẫu web để lọc các mặt hàng tốt nhất, các mặt hàng đã mua bởi khách hàng, khách hàng mua các mặt hàng

5. Quản lý quy trình thanh toán (có thể thử nghiệm thanh toán trực tuyến qua VNPay)

CHƯƠNG 2 – WINFORM APPLICATION AND ASP.NET CORE

2.1 Tạo cở sở dữ liệu hoàn chỉnh.

Cở sở dữ liệu của winform application được hình thành bởi 9 bảng được mở rộng ra từ 4 bảng chính mà đề bài đã cho để có thể làm ra 1 sản phẩm thực tế cũng như tạo ra 1 sản phẩm hoàn thiện .



2.2 Winform Application.

2.2.1 Xử lý đăng nhập.

* Phương thức clearButton\_Click: Đây là phương thức xử lý sự kiện khi người dùng nhấp vào nút "clearButton" (nút Xóa) trên giao diện. Khi được gọi, phương thức này sẽ xóa nội dung của các trường văn bản txtUsername và txtPassword, sau đó đặt trỏ chuột vào trường txtUsername để người dùng có thể nhập liệu mới.

private *void* clearButton\_Click(*object* *sender*, *EventArgs* *e*)

        {

            txtUsername.Text = "";

            txtPassword.Text = "";

            txtUsername.Focus();

        }

* Phương thức checkboxShowPass\_CheckedChanged: Phương thức này xử lý sự kiện khi người dùng thay đổi trạng thái của ô chọn "checkboxShowPass" (checkbox Hiển thị mật khẩu). Nếu ô chọn được chọn, phương thức sẽ đặt thuộc tính PasswordChar của trường văn bản txtPassword thành ký tự null ('\0'), làm cho mật khẩu hiển thị dưới dạng văn bản thường. Ngược lại, nếu ô chọn không được chọn, phương thức sẽ đặt PasswordChar thành ký tự '\*' để ẩn mật khẩu.

private *void* checkboxShowPass\_CheckedChanged(*object* *sender*, *EventArgs* *e*)

        {

            if (checkboxShowPass.Checked)

            {

                txtPassword.PasswordChar = '\0';

            }

            else

            {

                txtPassword.PasswordChar = '\*';

            }

        }

* Phương thức LoginButton\_Click: Đây là phương thức xử lý sự kiện khi người dùng nhấp vào nút "LoginButton" (nút Đăng nhập). Khi được gọi, phương thức này sẽ thiết lập kết nối đến cơ sở dữ liệu SQL Server thông qua chuỗi kết nối được cung cấp. Sau đó, nó thực hiện truy vấn SQL để kiểm tra xem có bản ghi nào trong bảng "Users" có email và số điện thoại trùng khớp với thông tin đăng nhập được nhập vào (txtUsername.Text và txtPassword.Text). Nếu có bản ghi khớp, nó sẽ hiển thị Form1 (giao diện chính) bằng cách tạo một đối tượng Form1 và gọi phương thức Show(). Ngược lại, nếu không có bản ghi khớp, nó sẽ hiển thị một hộp thoại cảnh báo cho người dùng với thông báo nội dung không chính xác.

Lưu ý rằng trong mã trên, kết nối đến cơ sở dữ liệu SQL Server được thiết lập bằng cách cung cấp chuỗi kết nối "Data Source=MYASUS\SQLEXPRESS;Initial Catalog=company;Integrated Security=True". Bạn có thể thay đổi chuỗi kết nối này để phù hợp với môi trường cơ sở dữ liệu của bạn.

private *void* LoginButton\_Click(*object* *sender*, *EventArgs* *e*)

        {

            using (connection = new *SqlConnection*("Data Source=MYASUS\\SQLEXPRESS;Initial Catalog=company;Integrated Security=True"))

            {

                try

                {

                    connection.Open();

                    Console.WriteLine("Access to database successfully");

*string* selectQuery = "SELECT Email, PhoneNumber FROM Users WHERE Email = @Email AND PhoneNumber = @PhoneNumber";

                    using (*SqlCommand* command = new *SqlCommand*(selectQuery, connection))

                    {

                        command.Parameters.AddWithValue("@Email", txtUsername.Text);

                        command.Parameters.AddWithValue("@PhoneNumber", txtPassword.Text);

                        using (*SqlDataReader* reader = command.ExecuteReader())

                        {

                            if (reader.HasRows)

                            {

*Form1* mainForm = new *Form1*();

                                mainForm.Show(); // Show the main form

                            }

                            else

                            {

                                MessageBox.Show("Incorrect login information. Please check your email and password.", "Notification", MessageBoxButtons.OK, MessageBoxIcon.Error);

                            }

                        }

                    }

                    connection.Close();

                }

                catch (*Exception* ex)

                {

                    Console.WriteLine("Error connecting to the database: " + ex.Message);

                }

            }

        }

2.2.2 Tạo mainform sau khi xác thực đăng nhập xong thì chuyển đến main form.

1.Phương thức loadform: Phương thức này được sử dụng để tải một form được chỉ định vào một Panel (được gọi là mainpanel). Khi được gọi, phương thức kiểm tra xem mainpanel đã chứa một thành phần con (Controls.Count > 0) hay chưa. Nếu có, nó loại bỏ thành phần con đầu tiên (Controls.RemoveAt(0)) trước khi thêm form mới. Sau đó, phương thức chuyển đổi đối tượng Form thành một đối tượng Form thực sự (Form f = Form as Form;). Tiếp theo, nó thiết lập các thuộc tính TopLevel và Dock của form để nó trở thành một thành phần con của mainpanel và lấp đầy kích thước của mainpanel. Sau đó, form được thêm vào mainpanel (mainpanel.Controls.Add(f)) và Tag của mainpanel được đặt thành form (mainpanel.Tag = f). Cuối cùng, form được hiển thị (f.Show()).

Các phương thức xử lý sự kiện cho các nút (btnemp\_Click, btnreports\_Click, btnclose\_Click, button1\_Click, button2\_Click, button3\_Click, button4\_Click, button5\_Click, FILTER\_Click): Các phương thức này được gọi khi người dùng nhấp vào các nút tương ứng trên giao diện. Mỗi phương thức gọi phương thức loadform với một form cụ thể làm đối số. Ví dụ: loadform(new CustomerForm()) sẽ tải form CustomerForm vào mainpanel.

public *void* loadform(*object* *Form*)

        {

            if (*this*.mainpanel.Controls.Count > 0)

*this*.mainpanel.Controls.RemoveAt(0);

*Form* f = Form as *Form*;

            f.TopLevel = false;

            f.Dock = DockStyle.Fill;

*this*.mainpanel.Controls.Add(f);

*this*.mainpanel.Tag = f;

            f.Show();

        }

        private *void* btnemp\_Click(*object* *sender*, *EventArgs* *e*)

        {

            loadform(new *CustomerForm*());

        }

        private *void* btnreports\_Click(*object* *sender*, *EventArgs* *e*)

        {

            loadform(new *ViewReport*());

        }

        private *void* btnclose\_Click(*object* *sender*, *EventArgs* *e*)

        {

            Application.Exit();

        }

        private *void* button1\_Click(*object* *sender*, *EventArgs* *e*)

        {

            loadform(new *productform*());

        }

        private *void* tableLayoutPanel1\_Paint(*object* *sender*, *PaintEventArgs* *e*)

        {

        }

        private *void* mainpanel\_Paint(*object* *sender*, *PaintEventArgs* *e*)

        {

        }

        private *void* button2\_Click(*object* *sender*, *EventArgs* *e*)

        {

            loadform(new *ListProduct*());

        }

        private *void* button3\_Click(*object* *sender*, *EventArgs* *e*)

        {

            loadform(new *ListCustomers*());

        }

        private *void* button4\_Click(*object* *sender*, *EventArgs* *e*)

        {

            loadform(new *OrderForm*());

        }

        private *void* button5\_Click(*object* *sender*, *EventArgs* *e*)

        {

            loadform(new *ListOrder*());

        }

        private *void* FILTER\_Click(*object* *sender*, *EventArgs* *e*)

        {

            loadform(new *FilterForm*());

        }

2.2.3 Thêm sản phẩm(Products) và đọc dữ liệu sản phẩm. Dưới đây là giải thích cho từng phương thức trong mã trên:

1. Phương thức `checkedListBox1\_SelectedIndexChanged`: Phương thức này xử lý sự kiện khi người dùng chọn một mục trong `checkedListBox1`. Khi được gọi, phương thức này lấy chỉ mục của mục đã chọn (`index`) và tổng số mục trong `checkedListBox1` (`count`). Sau đó, nó duyệt qua tất cả các mục trong `checkedListBox1` và đặt trạng thái kiểm tra (`CheckState`) của các mục khác với mục đã chọn thành `CheckState.Unchecked` (không được chọn).

private *void* checkedListBox1\_SelectedIndexChanged(*object* *sender*, *EventArgs* *e*)

        {

*int* index = checkedListcate.SelectedIndex;

*int* count = checkedListcate.Items.Count;

            for (*int* x = 0; x < count; x++)

            {

                if (index != x)

                {

                    checkedListcate.SetItemCheckState(x, CheckState.Unchecked);

                }

            }

        }

2. Phương thức `clear\_Click`: Đây là phương thức xử lý sự kiện khi người dùng nhấp vào nút "clear" (nút Xóa) trên giao diện. Khi được gọi, phương thức này đặt giá trị rỗng cho các trường văn bản `txtname`, `txtprice`, `txtdes` và `txtquantity`, sau đó xóa lựa chọn của `checkedListcate`.

private *void* clear\_Click(*object* *sender*, *EventArgs* *e*)

        {

            txtname.Text = *string*.Empty;

            txtprice.Text = *string*.Empty;

            txtdes.Text = *string*.Empty;

            txtquantity.Text = *string*.Empty;

            checkedListcate.ClearSelected();

        }

3. Phương thức `add\_Click`: Đây là phương thức xử lý sự kiện khi người dùng nhấp vào nút "add" (nút Thêm) trên giao diện. Khi được gọi, phương thức này kiểm tra xem người dùng đã chọn một danh mục (`checkedListcate.SelectedItem`) hay chưa. Nếu không, nó hiển thị một hộp thoại cảnh báo yêu cầu người dùng chọn một danh mục và kết thúc phương thức. Nếu người dùng đã chọn một danh mục, phương thức tiếp tục lấy giá trị đã nhập từ các trường văn bản `txtname`, `txtdes`, `txtprice` và `txtquantity`. Nếu bất kỳ trường nào là rỗng, phương thức hiển thị một hộp thoại cảnh báo yêu cầu người dùng nhập đầy đủ thông tin và kết thúc phương thức. Nếu các trường không rỗng, phương thức kiểm tra xem `txtprice` có thể được chuyển đổi thành một số thập phân (`decimal.TryParse`) và `txtquantity` có thể được chuyển đổi thành một số nguyên (`int.TryParse`). Nếu không, nó hiển thị một hộp thoại cảnh báo yêu cầu người dùng nhập đúng định dạng số và kết thúc phương thức. Cuối cùng, nếu tất cả thông tin hợp lệ, phương thức thiết lập kết nối đến cơ sở dữ liệu SQL Server, thực hiện một truy vấn SQL để chèn dữ liệu vào bảng "Products" và hiển thị một hộp thoại thông báo thành công. Sau đó, phương thức gọi phương thức `clear\_Click` để xóa nội dung các trường văn bản.

private *void* add\_Click(*object* *sender*, *EventArgs* *e*)

        {

*string* selectedCategory = checkedListcate.SelectedItem?.ToString();

            if (*string*.IsNullOrEmpty(selectedCategory))

            {

                MessageBox.Show("Please select a category");

                return;

            }

*string* name = txtname.Text.Trim();

            if (*string*.IsNullOrEmpty(name))

            {

                MessageBox.Show("Please enter a name");

                return;

            }

*string* description = txtdes.Text.Trim();

            if (*string*.IsNullOrEmpty(description))

            {

                MessageBox.Show("Please enter a description");

                return;

            }

*string* price = txtprice.Text.Trim();

            if (*string*.IsNullOrEmpty(price))

            {

                MessageBox.Show("Please enter a price");

                return;

            }

            if (!*decimal*.TryParse(price, out *decimal* parsePrice))

            {

                MessageBox.Show("Price must be a number");

                return;

            }

*string* quantity = txtquantity.Text.Trim();

            if (*string*.IsNullOrEmpty(quantity))

            {

                MessageBox.Show("Please enter a quantity");

                return;

            }

            if (!*int*.TryParse(quantity, out *int* parseQuantity))

            {

                MessageBox.Show("Quantity must be a whole number");

                return;

            }

*int* categoryId = GetIdCategory(selectedCategory);

*string* connectionString = "Data Source=MYASUS\\SQLEXPRESS;Initial Catalog=company;Integrated Security=True";

            using (*SqlConnection* conn = new *SqlConnection*(connectionString))

            {

                try

                {

                    conn.Open();

                    Console.WriteLine("Connection successful");

*string* insertQuery = "INSERT INTO Products (Name, Price, Description, StockQuantity, CategoryId) VALUES (@Name, @Price, @Description, @Quantity, @CategoryId)";

                    using (*SqlCommand* insertCommand = new *SqlCommand*(insertQuery, conn))

                    {

                        insertCommand.Parameters.AddWithValue("@Name", name);

                        insertCommand.Parameters.AddWithValue("@Price", parsePrice);

                        insertCommand.Parameters.AddWithValue("@Description", description);

                        insertCommand.Parameters.AddWithValue("@Quantity", parseQuantity);

                        insertCommand.Parameters.AddWithValue("@CategoryId", categoryId);

*int* rowsInserted = insertCommand.ExecuteNonQuery();

                        MessageBox.Show("Inserted data successfully");

                        clear\_Click(sender,e);

                    }

                }

                catch (*Exception* ex)

                {

                    Console.WriteLine("Connection failed: " + ex.Message);

                }

            }

        }

4. Phương thức `GetIdCategory`: Đây là một phương thức hỗ trợ được sử dụng để lấy ID của một danh mục dựa trên tên danh mục. Phương thức này nhận một tham số `categoryName` là tên của danh mục và trả về ID của danh mục tương ứng. Đầu tiên, phương thức thiết lập kết nối đến cơ sở dữ liệu SQL Server. Sau đó, nó thực hiện một truy vấn SQL để lấy ID từ bảng "Categories" dựa trên tên danh mục được cung cấp. Kết quả truy vấn được trả về là một đối tượng, và phương thức kiểm tra xem kết quả có khác null và DBNull không. Nếu kết quảkhác null hoặc DBNull, phương thức chuyển đổi kết quả thành một số nguyên và trả về ID của danh mục. Nếu không có danh mục khớp, giá trị trả về là -1 để chỉ không tìm thấy danh mục phù hợp.

private *int* GetIdCategory(*string* *categoryName*)

        {

*int* categoryId = -1; // Default value to indicate no matching category found

            using (*SqlConnection* conn = new *SqlConnection*("Data Source=MYASUS\\SQLEXPRESS;Initial Catalog=company;Integrated Security=True"))

            {

                conn.Open();

*string* query = "SELECT Id FROM Categories WHERE Name = @CategoryName";

                using (*SqlCommand* cmd = new *SqlCommand*(query, conn))

                {

                    cmd.Parameters.AddWithValue("@CategoryName", categoryName);

                    // Execute the query to retrieve the category ID

*object* result = cmd.ExecuteScalar();

                    if (result != null && result != DBNull.Value)

                    {

                        categoryId = (*int*)result; // Convert the result to an integer

                    }

                }

            }

            return categoryId;

        }

5. Liệt kê sản phẩm: Để liệt kê các sản phẩm ra thì tạo 1 Data Set trong đó có bảng Product và liên kết với database bằng querstring do Visual Studio 2022 và SSMS hỗ trợ.

private *void* ListProduct\_Load(*object* *sender*, *EventArgs* *e*)

        {

            // TODO: This line of code loads data into the 'companyDataSet.Products' table. You can move, or remove it, as needed.

*this*.productsTableAdapter.Fill(*this*.companyDataSet.Products);

        }

        private *void* fillByToolStripButton\_Click(*object* *sender*, *EventArgs* *e*)

        {

            try

            {

*this*.productsTableAdapter.FillBy(*this*.companyDataSet.Products);

            }

            catch (*System*.*Exception* ex)

            {

                System.Windows.Forms.MessageBox.Show(ex.Message);

            }

        }

        private *void* fillBy1ToolStripButton\_Click(*object* *sender*, *EventArgs* *e*)

        {

            try

            {

*this*.productsTableAdapter.FillBy1(*this*.companyDataSet.Products);

            }

            catch (*System*.*Exception* ex)

            {

                System.Windows.Forms.MessageBox.Show(ex.Message);

            }

        }

        private *void* productsToolStripButton\_Click(*object* *sender*, *EventArgs* *e*)

        {

            try

            {

*this*.productsTableAdapter.Products(*this*.companyDataSet.Products);

            }

            catch (*System*.*Exception* ex)

            {

                System.Windows.Forms.MessageBox.Show(ex.Message);

            }

        }

2.2.4 Thêm khách hàng(Customers) và đọc dữ liệu khác hàng.

1. Phương thức `clear\_Click`: Phương thức này được gọi khi người dùng nhấp vào một nút (gọi là `clear`) để xóa nội dung của một số ô văn bản trên giao diện. Cụ thể, phương thức đặt giá trị của các ô văn bản `txtname`, `txtaddress`, `txtemail` và `txtphone` thành chuỗi rỗng (`string.Empty`).

 private *void* clear\_Click(*object* *sender*, *EventArgs* *e*)

        {

            txtname.Text = *string*.Empty;

            txtaddress.Text = *string*.Empty;

            txtemail.Text = *string*.Empty;

            txtphone.Text = *string*.Empty;

        }

2. Phương thức `add\_Click`: Phương thức này được gọi khi người dùng nhấp vào một nút (gọi là `add`) để thêm khách hàng vào cơ sở dữ liệu. Các bước thực hiện trong phương thức này như sau:

- Trước tiên, phương thức lấy giá trị từ các ô văn bản `txtname`, `txtaddress`, `txtemail` và `txtphone` và lưu chúng vào các biến tương ứng (`name`, `address`, `email` và `phone`).

- Sau đó, phương thức kiểm tra xem số điện thoại (`phone`) có đúng định dạng 10 chữ số hay không bằng cách sử dụng biểu thức chính quy (`Regex.IsMatch(phone, "^[0-9]{10}$")`). Nếu số điện thoại không hợp lệ, một hộp thoại thông báo sẽ xuất hiện và phương thức sẽ kết thúc (`return`).

- Tiếp theo, phương thức tạo một chuỗi kết nối cơ sở dữ liệu (`connectionString`) để kết nối đến cơ sở dữ liệu. Chuỗi kết nối được cung cấp với tên máy chủ, tên cơ sở dữ liệu và thông tin xác thực.

- Sau đó, phương thức tạo một đối tượng `SqlConnection` và mở kết nối đến cơ sở dữ liệu (`conn.Open()`).

- Tiếp theo, phương thức kiểm tra xem số điện thoại (`phone`) đã tồn tại trong bảng `Customers` hay chưa. Điều này được thực hiện thông qua câu truy vấn SQL `SELECT COUNT(\*) FROM Customers WHERE Phone = @Phone`. Đối số `@Phone` được sử dụng để truyền giá trị của số điện thoại vào câu truy vấn, ngăn chặn tấn công SQL injection. Kết quả của câu truy vấn được trả về bằng cách sử dụng `ExecuteScalar()` và gán cho biến `customerCount`.

- Nếu `customerCount` lớn hơn 0, tức là số điện thoại đã tồn tại trong cơ sở dữ liệu, một hộp thoại thông báo sẽ xuất hiện và phương thức sẽ kết thúc (`return`).

- Nếu số điện thoại chưa tồn tại trong cơ sở dữ liệu, phương thức tiếp tục để thực hiện câu lệnh `INSERT` SQL để thêm khách hàng mới vào bảng `Customers`. Câu lệnh `INSERT` được thực hiện thông qua câu truy vấn SQL `INSERT INTO Customers (Name, Address, Email, Phone) VALUES (@Name, @Address, @Email, @Phone)`. Các đối số `@Name`, `@Address`, `@Email` và `@Phone` được sử dụng để truyền các giá trị tương ứng từ các biến vào câu truy vấn.

- Sau đó, phương thức thực thi câu lệnh SQL bằng cách sử dụng `ExecuteNonQuery()` vàkiểm tra số hàng bị ảnh hưởng (`rowsAffected`). Nếu số hàng bị ảnh hưởng lớn hơn 0, tức là khách hàng đã được thêm thành công vào cơ sở dữ liệu, một hộp thoại thông báo sẽ xuất hiện và phương thức `clear\_Click` được gọi để xóa nội dung các ô văn bản. Nếu số hàng bị ảnh hưởng là 0, tức là thất bại khi thêm khách hàng vào cơ sở dữ liệu, một hộp thoại thông báo sẽ xuất hiện.

Dòng `using` được sử dụng trong mã để đảm bảo rằng các đối tượng `SqlConnection` và `SqlCommand` sẽ được giải phóng ngay sau khi không còn sử dụng để tránh rò rỉ tài nguyên.

private *void* add\_Click(*object* *sender*, *EventArgs* *e*)

        {

            // Get the values from the text fields

*string* name = txtname.Text;

*string* address = txtaddress.Text;

*string* email = txtemail.Text;

*string* phone = txtphone.Text;

            // Validate the phone number

            if (!Regex.IsMatch(phone, "^[0-9]{10}$"))

            {

                MessageBox.Show("Phone number must be a 10-digit number.");

                return; // Exit the function if the phone number is invalid

            }

            // Connection string

*string* connectionString = "Data Source=MYASUS\\SQLEXPRESS;Initial Catalog=company;Integrated Security=True";

            using (*SqlConnection* conn = new *SqlConnection*(connectionString))

            {

                conn.Open();

                // Check if the phone number already exists in the Customers table

*string* checkQuery = "SELECT COUNT(\*) FROM Customers WHERE Phone = @Phone";

                using (*SqlCommand* checkCmd = new *SqlCommand*(checkQuery, conn))

                {

                    checkCmd.Parameters.AddWithValue("@Phone", phone);

*int* customerCount = (*int*)checkCmd.ExecuteScalar();

                    if (customerCount > 0)

                    {

                        MessageBox.Show("Phone number already exists in the database.");

                        return; // Exit the function if the phone number already exists

                    }

                }

                // SQL INSERT statement

*string* insertQuery = "INSERT INTO Customers (Name, Address, Email, Phone) VALUES (@Name, @Address, @Email, @Phone)";

                using (*SqlCommand* cmd = new *SqlCommand*(insertQuery, conn))

                {

                    // Add parameters to the SQL query to prevent SQL injection

                    cmd.Parameters.AddWithValue("@Name", name);

                    cmd.Parameters.AddWithValue("@Address", address);

                    cmd.Parameters.AddWithValue("@Email", email);

                    cmd.Parameters.AddWithValue("@Phone", phone);

                    // Execute the SQL query

*int* rowsAffected = cmd.ExecuteNonQuery();

                    if (rowsAffected > 0)

                    {

                        MessageBox.Show("Customer added successfully.");

                        clear\_Click(sender, e);

                    }

                    else

                    {

                        MessageBox.Show("Failed to add the customer.");

                    }

                }

            }

        }

3. Liệt kê Khách hàng: Tương tự như product chúng ta cũng tạo 1 dataset liên kết với database bằng query string và chọn bảng dữ liệu và các cột để liệt kê.

 private *void* ListCusandOrder\_Load(*object* *sender*, *EventArgs* *e*)

        {

            // TODO: This line of code loads data into the 'companyDataSet.Customers' table. You can move, or remove it, as needed.

*this*.customersTableAdapter.Fill(*this*.companyDataSet.Customers);

        }

        private *void* customersToolStripButton\_Click(*object* *sender*, *EventArgs* *e*)

        {

            try

            {

*this*.customersTableAdapter.Customers(*this*.companyDataSet.Customers);

            }

            catch (*System*.*Exception* ex)

            {

                System.Windows.Forms.MessageBox.Show(ex.Message);

            }

        }

2.2.5 Thêm đơn hàng(Orders) và đọc dữ liệu đơn hàng.

1. Sự kiện txtcusid\_TextChanged: Được gọi khi nội dung của TextBox txtcusid thay đổi. Sự kiện này được sử dụng để kiểm tra nếu TextBox không rỗng, sau đó cập nhật giá trị của TextBox orderdate với thời gian và ngày hiện tại bằng cách sử dụng DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss.ffffff"). Sau đó, phương thức LatestOrderID() được gọi để cập nhật giá trị của TextBox orderid.

 private *void* txtcusid\_TextChanged(*object* *sender*, *EventArgs* *e*)

        {

            // Check if the txtcusid TextBox is not empty

            if (!*string*.IsNullOrEmpty(txtcusid.Text))

            {

                // Assign the orderdate.Text with the current date and time

                orderdate.Text = DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss.ffffff");

                LatestOrderID();

            }

            else

            {

                // Clear the orderdate.Text if txtcusid is empty

                orderdate.Text = *string*.Empty;

            }

        }

2. Phương thức clear\_Click: Được gọi khi nút clear được nhấp. Phương thức này được sử dụng để xóa nội dung của các TextBox (txtcusid, orderdate, txttotalamount) và xóa tất cả các hàng trong dataGridView1.

private *void* clear\_Click(*object* *sender*, *EventArgs* *e*)

        {

            txtcusid.Text = *string*.Empty;

            orderdate.Text = *string*.Empty;

            txttotalamount.Text = *string*.Empty;

            dataGridView1.Rows.Clear();

        }

3. Phương thức UpdateTotal: Tính toán tổng số tiền cho các mặt hàng trong dataGridView1 và cập nhật giá trị của TextBox txttotalamount.

private *decimal* UpdateTotal()

        {

*decimal* total = 0;

            foreach (*DataGridViewRow* row in dataGridView1.Rows)

            {

                if (row.Cells["totalprice"].Value != null)

                {

                    total += Convert.ToDecimal(row.Cells["totalprice"].Value);

                }

            }

            txttotalamount.Text = total.ToString();

            return total;

        }

4. Sự kiện DataGridView1\_CellValueChanged: Được gọi khi giá trị của một ô trong dataGridView1 thay đổi. Sự kiện này được sử dụng để tính toán giá trị tổng tiền (totalprice) cho mỗi hàng dựa trên giá (price) và số lượng (quantity) được nhập. Sau đó, phương thức UpdateTotal được gọi để cập nhật tổng số tiền.

private *void* DataGridView1\_CellValueChanged(*object* *sender*, *DataGridViewCellEventArgs* *e*)

        {

            if (e.RowIndex >= 0 && e.ColumnIndex == dataGridView1.Columns["quantity"].Index)

            {

*int* quantity = Convert.ToInt32(dataGridView1.Rows[e.RowIndex].Cells["quantity"].Value);

*int* productId = Convert.ToInt32(dataGridView1.Rows[e.RowIndex].Cells["product"].Value);

*decimal* price = GetPriceFromProductId(productId);

*decimal* totalprice = quantity \* price;

                dataGridView1.Rows[e.RowIndex].Cells["price"].Value = price;

                dataGridView1.Rows[e.RowIndex].Cells["totalprice"].Value = totalprice;

                UpdateTotal();

            }

        }

5. Phương thức GetPriceFromProductId: Truy vấn cơ sở dữ liệu để lấy giá của sản phẩm dựa trên ID sản phẩm (productId). Kết nối cơ sở dữ liệu được thiết lập và truy vấn được thực thi bằng sử dụng SqlConnection và SqlCommand. Giá trị giá được trả về từ truy vấn và được chuyển đổi thành kiểu decimal.

 private *decimal* GetPriceFromProductId(*int* *productId*)

        {

*string* connectionString = "Data Source=MYASUS\\SQLEXPRESS;Initial Catalog=company;Integrated Security=True";

*decimal* price = 0;

            using (*SqlConnection* conn = new *SqlConnection*(connectionString))

            {

                try

                {

                    conn.Open();

                    Console.WriteLine("Connection successful");

*string* selectQuery = "SELECT Price FROM Products WHERE ID = @ProductId";

                    using (*SqlCommand* selectCommand = new *SqlCommand*(selectQuery, conn))

                    {

                        selectCommand.Parameters.AddWithValue("@ProductId", productId);

*object* result = selectCommand.ExecuteScalar();

                        if (result != null)

                        {

                            price = Convert.ToDecimal(result);

                        }

                    }

                }

                catch (*Exception* ex)

                {

                    Console.WriteLine("Connection failed: " + ex.Message);

                }

            }

            return price;

        }

6. Phương thức LatestOrderID: Truy vấn cơ sở dữ liệu để lấy ID đơn hàng lớn nhất từ bảng Orders và trả về giá trị ID đơn hàng mới nhất (tăng lên 1). Kết nối cơ sở dữ liệu được thiết lập và truy vấn được thực thi bằng sử dụng SqlConnection và SqlCommand. Giá trị ID đơn hàng được trả về từ truy vấn và được chuyển đổi thành kiểu int.

private *int* LatestOrderID()

        {

*string* connectionString = "Data Source=MYASUS\\SQLEXPRESS;Initial Catalog=company;Integrated Security=True";

*int* latestOrderID = 0;

            using (*SqlConnection* conn = new *SqlConnection*(connectionString))

            {

                conn.Open();

*string* query = "SELECT MAX(Id) FROM Orders"; //

                using (*SqlCommand* cmd = new *SqlCommand*(query, conn))

                {

                    var result = cmd.ExecuteScalar();

                    if (result != DBNull.Value)

                    {

                        latestOrderID = Convert.ToInt32(result) + 1 ;

                        orderid.Text = latestOrderID.ToString() ;

                    }

                }

            }

            return latestOrderID;

        }

7. Sự kiện button1\_Click: Được gọi khi nút button1 được nhấp. Sự kiện này được sử dụng để thêm đơn hàng và chi tiết đơn hàng vào cơ sở dữ liệu. Trước tiên, kiểm tra tính hợp lệ của thông tin khách hàng và tổng số tiền. Sau đó, một kết nối cơ sở dữ liệu được thiết lập và một giao dịch SQL bắt đầu. Các truy vấn INSERT được sử dụng để chèn thông tin đơn hàng và chi tiết đơn hàng vào cơ sở dữ liệu. Nếu có bất kỳ lỗi nào xảy ra, giao dịch sẽ bị rollback. Nếu không có lỗi, giao dịch sẽ được commit và thông báo thành công sẽ hiển thị.

 private *void* button1\_Click(*object* *sender*, *EventArgs* *e*)

        {

*decimal* totalAmount = UpdateTotal();

*string* cusId = txtcusid.Text;

            // Validate customer ID

            if (*string*.IsNullOrEmpty(cusId))

            {

                MessageBox.Show("Please enter a customer ID.");

                return;

            }

            if (!*int*.TryParse(cusId, out *int* parseCusID))

            {

                MessageBox.Show("Customer ID must be a number.");

                return;

            }

*string* dateTime = orderdate.Text;

*string* connectionString = "Data Source=MYASUS\\SQLEXPRESS;Initial Catalog=company;Integrated Security=True";

            using (*SqlConnection* conn = new *SqlConnection*(connectionString))

            {

                conn.Open();

*SqlTransaction* transaction = conn.BeginTransaction(); // Start a SQL transaction for inserting orders and order details

                try

                {

                    Console.WriteLine("Connection successful");

                    // Insert the order

*string* insertOrderQuery = "INSERT INTO Orders (OrderDate, TotalAmount, CustomerId) VALUES (@OrderDate, @TotalAmount, @CustomerId); SELECT SCOPE\_IDENTITY()";

                    using (*SqlCommand* insertOrderCommand = new *SqlCommand*(insertOrderQuery, conn, transaction))

                    {

                        insertOrderCommand.Parameters.AddWithValue("@OrderDate", dateTime);

                        insertOrderCommand.Parameters.AddWithValue("@TotalAmount", totalAmount);

                        insertOrderCommand.Parameters.AddWithValue("@CustomerId", parseCusID);

                        for (*int* i = 0; i < dataGridView1.Rows.Count - 1; i++)

                        {

*DataGridViewRow* row = dataGridView1.Rows[i];

*int* productId = Convert.ToInt32(row.Cells["product"].Value);

                            if (!ProductExists(productId))

                            {

                                MessageBox.Show($"Product with ID {productId} does not exist.");

                                transaction.Rollback();

                                return;

                            }

*int* quantity = Convert.ToInt32(row.Cells["quantity"].Value);

*decimal* totalPrice = Convert.ToDecimal(row.Cells["totalprice"].Value);

*int* orderId = LatestOrderID() - 1;

                            // Insert OrderDetail

*string* insertOrderDetailQuery = "INSERT INTO OrderDetails (OrderId, Quantity, TotalPrice, ProductsId) VALUES (@OrderId, @Quantity, @TotalPrice, @ProductsId)";

                            using (*SqlCommand* insertOrderDetailCommand = new *SqlCommand*(insertOrderDetailQuery, conn, transaction))

                            {

                                insertOrderDetailCommand.Parameters.AddWithValue("@OrderId", orderId);

                                insertOrderDetailCommand.Parameters.AddWithValue("@Quantity", quantity);

                                insertOrderDetailCommand.Parameters.AddWithValue("@TotalPrice", totalPrice);

                                insertOrderDetailCommand.Parameters.AddWithValue("@ProductsId", productId);

                                insertOrderDetailCommand.ExecuteNonQuery(); // Insert the OrderDetail

                            }

                        }

                        transaction.Commit(); // Commit the transaction

                        MessageBox.Show("Order and OrderDetails inserted successfully.");

                    }

                }

                catch (*Exception* ex)

                {

                    transaction.Rollback(); // Rollback the transaction if an error occurs

                    Console.WriteLine("Connection failed: " + ex.Message);

                    MessageBox.Show("An error occurred while inserting the order.");

                }

            }

        }

8. Phương thức ProductExists: Kiểm tra xem một sản phẩm có tồn tại trong cơ sở dữ liệu dựa trên ID sản phẩm (productId). Một truy vấn SELECT được sử dụng để kiểm tra sự tồn tại của sảnphẩm. Kết nối cơ sở dữ liệu được thiết lập và truy vấn được thực thi bằng sử dụng SqlConnection và SqlCommand. Kết quả truy vấn được kiểm tra, và phương thức trả về giá trị boolean (true nếu sản phẩm tồn tại, false nếu không tồn tại).

 private *bool* ProductExists(*int* *productId*)

        {

*string* connectionString = "Data Source=MYASUS\\SQLEXPRESS;Initial Catalog=company;Integrated Security=True";

*bool* exists = false;

            using (*SqlConnection* conn = new *SqlConnection*(connectionString))

            {

                conn.Open();

                // Define the SQL query to check if the product with the given ID exists

*string* query = "SELECT 1 FROM Products WHERE Id = @ProductId";

                using (*SqlCommand* cmd = new *SqlCommand*(query, conn))

                {

                    cmd.Parameters.AddWithValue("@ProductId", productId);

                    // Execute the query and check if any rows are returned

*SqlDataReader* reader = cmd.ExecuteReader();

                    exists = reader.HasRows;

                }

            }

            return exists;

        }

2.2.6 Tạo báo cáo để in đơn hàng

Để tạo báo cáo đơn hàng và in ra cần các bước sau:

1.Tạo một báo cáo: Đầu tiên, bạn cần tạo một báo cáo sử dụng công cụ báo cáo có sẵn trong Visual Studio, chẳng hạn như Crystal Reports hoặc Microsoft Report Viewer. Tạo các mẫu báo cáo, định dạng và thiết kế trên báo cáo này theo yêu cầu của bạn. Bạn có thể thêm các thông tin đơn hàng như số đơn hàng, tên khách hàng, sản phẩm, số lượng, giá, v.v.

2.Truy xuất dữ liệu: Tiếp theo, bạn cần truy xuất dữ liệu đơn hàng từ cơ sở dữ liệu của bạn. Sử dụng các lệnh SQL hoặc các phương thức truy vấn dữ liệu từ cơ sở dữ liệu để lấy thông tin về đơn hàng cần in.

3.Gắn kết dữ liệu: Sau khi có dữ liệu đơn hàng, bạn cần gắn kết dữ liệu này vào báo cáo. Sử dụng các phương thức như SetDataSource (đối với Crystal Reports) hoặc gán dữ liệu vào bộ dữ liệu của báo cáo (đối với Microsoft Report Viewer) để gắn kết dữ liệu từ nguồn dữ liệu vào báo cáo.

4.Hiển thị báo cáo: Bạn có thể hiển thị báo cáo trực tiếp trên một hộp thoại in bằng cách tạo một đối tượng báo cáo và gọi phương thức in. Hoặc bạn có thể hiển thị báo cáo trên một trình xem báo cáo (report viewer) trong giao diện người dùng của ứng dụng WinForms.

5.In báo cáo: Cuối cùng, bạn cần cung cấp tùy chọn để người dùng có thể in báo cáo. Sử dụng các phương thức in có sẵn trong công cụ báo cáo hoặc sử dụng các phương thức in từ hệ thống của WinForms để in báo cáo.

 private *void* ViewReport\_Load(*object* *sender*, *EventArgs* *e*)

        {

*DataSet1* report\_ds = new *DataSet1*();

            // Define the connection string to your SQL Server database

*string* connection\_string = "Data Source=MYASUS\\SQLEXPRESS;Initial Catalog=company;Integrated Security=True";

*string* query = "SELECT Id, OrderDate, TotalAmount, CustomerId FROM  Orders";

*SqlConnection* conn = new *SqlConnection*(connection\_string);

*SqlDataAdapter* adptr = new *SqlDataAdapter*(query, conn);

            adptr.Fill(report\_ds, report\_ds.Tables[0].TableName);

*ReportDataSource* rds = new *ReportDataSource*("DataSet1", report\_ds.Tables[0]);

            reportViewer1.LocalReport.ReportPath = "Report1.rdlc";

            reportViewer1.LocalReport.DataSources.Clear();

            reportViewer1.LocalReport.DataSources.Add(rds);

            reportViewer1.LocalReport.Refresh();

            reportViewer1.RefreshReport();

*this*.reportViewer1.RefreshReport();

        }

2.2.6 Tạo một form để lọc các mặt hàng tốt nhất, các mặt hàng được khách hàng mua, khách hàng mua các mặt hàng

1. Sự kiện `checkedListcate\_SelectedIndexChanged`: Được gọi khi người dùng thay đổi lựa chọn trong `checkedListcate` (CheckedListBox). Sự kiện này được sử dụng để xử lý việc chọn một danh mục cụ thể và thực hiện các hành động tương ứng.

- Lấy chỉ mục (index) của mục được chọn (`checkedListcate.SelectedIndex`).

- Lấy số lượng mục trong `checkedListcate` (`checkedListcate.Items.Count`).

- Duyệt qua tất cả các mục trong `checkedListcate` bằng một vòng lặp `for`.

- Kiểm tra nếu chỉ mục hiện tại không giống với chỉ mục được chọn (`index != x`), thì đặt trạng thái check của mục hiện tại thành "Unchecked" (`checkedListcate.SetItemCheckState(x, CheckState.Unchecked)`).

- Kiểm tra nếu mục "Best product" được chọn (`checkedListcate.CheckedItems.Contains("Best product")`):

- Đặt `txtfilter` thành chỉ đọc (`txtfilter.ReadOnly = true`).

- Xóa nội dung của `txtfilter` (`txtfilter.Text = ""`).

- Ngược lại, nếu mục "Best product" không được chọn:

- Cho phép chỉnh sửa `txtfilter` (`txtfilter.ReadOnly = false`).

2. Sự kiện `button1\_Click`: Được gọi khi nút `button1` được nhấp. Sự kiện này được sử dụng để tìm kiếm thông tin trong cơ sở dữ liệu dựa trên các lựa chọn đã chọn trong `checkedListcate` và giá trị của `txtfilter`.

- Khởi tạo chuỗi kết nối đến cơ sở dữ liệu (`connectionString`).

- Sử dụng `SqlConnection` để mở kết nối đến cơ sở dữ liệu.

- Kiểm tra các lựa chọn trong `checkedListcate`:

- Nếu "Products" được chọn:

- Thiết lập `selectedTable` thành "Products".

- Tạo truy vấn SQL để lấy thông tin sản phẩm dựa trên tên sản phẩm (`query`).

- Sử dụng `SqlCommand` để thực thi truy vấn SQL và lấy dữ liệu vào `DataTable` (`dt`).

- Đặt nguồn dữ liệu của `dataGridView1` thành `dt`.

- Hiển thị thông báo dựa trên số lượng hàng trả về từ truy vấn.

- Nếu "Customers" được chọn:

- Thiết lập `selectedTable` thành "Customers".

- Tạo truy vấn SQL để lấy thông tin khách hàng dựa trên tên khách hàng (`query`).

- Sử dụng `SqlCommand` để thực thi truy vấn SQL và lấy dữ liệu vào `DataTable` (`dt`).

- Đặt nguồn dữ liệu của `dataGridView1` thành `dt`.

- Hiển thị thông báo dựa trên số lượng hàng trả về từ truy vấn.

- Nếu "Best product" được chọn:

- Thiết lập `selectedTable` thành "Best product".

- Tạo truy vấn SQL để lấy thông tin sản phẩm có giá cao nhất (`query`).

- Sử dụng `SqlCommand` để thực thi truy vấn SQL và lấy dữ liệu vào `DataTable` (`dt`).

- Đặt nguồn dữ liệu của `dataGridView1` thành `dt`.

- Hiển thị thông báo dựa trên số lượng hàng trả về từ truy vấn.

- Nếu không có lựa chọn nào được chọn, hiển thị thông báo yêu cầu người dùng chọn loại dữ liệu để tìm kiếm.

3. Trong mã, có sử dụng các đối tượng `SqlConnection`, `SqlCommand`, `SqlDataAdapter`, `DataTable` và `DataGridView` để thao tác với cơ sở dữ liệu SQL Server

- Đối tượng `SqlConnection` được sử dụng để thiết lập và quản lý kết nối đến cơ sở dữ liệu.

- Đối tượng `SqlCommand` được sử dụng để thực thi truy vấn SQL và truyền các tham số.

- Đối tượng `SqlDataAdapter` được sử dụng để lấy dữ liệu từ `SqlCommand` và điền vào `DataTable`.

- Đối tượng `DataTable` được sử dụng để lưu trữ dữ liệu từ kết quả truy vấn.

- Đối tượng `DataGridView` được sử dụng để hiển thị dữ liệu từ `DataTable` trên giao diện người dùng.

private *void* checkedListcate\_SelectedIndexChanged(*object* *sender*, *EventArgs* *e*)

        {

*int* index = checkedListcate.SelectedIndex;

*int* count = checkedListcate.Items.Count;

            for (*int* x = 0; x < count; x++)

            {

                if (index != x)

                {

                    checkedListcate.SetItemCheckState(x, CheckState.Unchecked);

                }

            }

            if (checkedListcate.CheckedItems.Contains("Best product"))

            {

                // Nếu được chọn, chuyển txtfilter thành read-only

                txtfilter.ReadOnly = true;

                txtfilter.Text = "";

            }

            else

            {

                // Nếu không được chọn, cho phép chỉnh sửa txtfilter

                txtfilter.ReadOnly = false;

            }

        }

        private *void* button1\_Click(*object* *sender*, *EventArgs* *e*)

        {

*string* selectedTable = "";

*string* connectionString = "Data Source=MYASUS\\SQLEXPRESS;Initial Catalog=company;Integrated Security=True";

            using (*SqlConnection* conn = new *SqlConnection*(connectionString))

            {

                try

                {

                    conn.Open();

                    Console.WriteLine("Connection successful");

                    if (checkedListcate.CheckedItems.Contains("Products"))

                    {

                        selectedTable = "Products";

                        Console.WriteLine("Products checkbox status: " + checkedListcate.CheckedItems.Contains("Products"));

*string* query = "SELECT C.Name AS CustomerName, O.OrderDate, P.Name AS ProductName, OD.Quantity AS QuantityPurchased FROM Customers C JOIN Orders O ON C.Id = O.CustomerId JOIN OrderDetails OD ON O.Id = OD.OrderId JOIN Products P ON OD.ProductsId = P.Id WHERE P.Name LIKE @ProductName ORDER BY C.Name, O.OrderDate;";

                        using (*SqlCommand* sqlCommand = new *SqlCommand*(query, conn))

                        {

                            sqlCommand.Parameters.AddWithValue("@ProductName", "%" + txtfilter.Text.Trim() + "%");

                            // Create a DataTable to hold the query results

*DataTable* dt = new *DataTable*();

                            // Use a SqlDataAdapter to fill the DataTable

*SqlDataAdapter* da = new *SqlDataAdapter*(sqlCommand);

                            da.Fill(dt);

                            // Set the DataGridView's data source to the DataTable

                            dataGridView1.DataSource = dt;

                            if (dt.Rows.Count > 0)

                            {

                                MessageBox.Show("Đã tìm thấy sản phẩm");

                            }

                            else

                            {

                                MessageBox.Show("Không tìm thấy sản phẩm");

                            }

                        }

                    }

                    else if (checkedListcate.CheckedItems.Contains("Customers"))

                    {

                        selectedTable = "Customers";

                        Console.WriteLine("Products checkbox status: " + checkedListcate.CheckedItems.Contains("Customers"));

*string* query = "SELECT C.Name AS CustomerName, P.Name AS ProductName, OD.Quantity AS QuantityPurchased, O.OrderDate FROM Customers C JOIN Orders O ON C.Id = O.CustomerId JOIN OrderDetails OD ON O.Id = OD.OrderId JOIN Products P ON OD.ProductsId = P.Id WHERE C.Name LIKE @CustomerName ORDER BY C.Name, O.OrderDate";

                        using (*SqlCommand* sqlCommand = new *SqlCommand*(query, conn))

                        {

                            sqlCommand.Parameters.AddWithValue("@CustomerName", "%" + txtfilter.Text.Trim() + "%");

                            // Create a DataTable to hold the query results

*DataTable* dt = new *DataTable*();

                            // Use a SqlDataAdapter to fill the DataTable

*SqlDataAdapter* da = new *SqlDataAdapter*(sqlCommand);

                            da.Fill(dt);

                            // Set the DataGridView's data source to the DataTable

                            dataGridView1.DataSource = dt;

                            if (dt.Rows.Count > 0)

                            {

                                MessageBox.Show("Đã tìm thấy khách hàng");

                            }

                            else

                            {

                                MessageBox.Show("Không tìm thấy khách hàng");

                            }

                        }

                    }

                    else if (checkedListcate.CheckedItems.Contains("Best product"))

                    {

                        selectedTable = "Best product";

                        Console.WriteLine("Products checkbox status: " + checkedListcate.CheckedItems.Contains("Best product"));

*string* query = "SELECT \* FROM Products WHERE Price = (SELECT MAX(Price) FROM Products)";

                        using (*SqlCommand* sqlCommand = new *SqlCommand*(query, conn))

                        {

                            // Create a DataTable to hold the query results

*DataTable* dt = new *DataTable*();

                            // Use a SqlDataAdapter to fill the DataTable

*SqlDataAdapter* da = new *SqlDataAdapter*(sqlCommand);

                            da.Fill(dt);

                            // Set the DataGridView's data source to the DataTable

                            dataGridView1.DataSource = dt;

                            if (dt.Rows.Count > 0)

                            {

                                MessageBox.Show("Đã tìm thấy sản phẩm");

                            }

                            else

                            {

                                MessageBox.Show("Không tìm thấy sản phẩm");

                            }

                        }

                    }

                    else

                    {

                        MessageBox.Show("Vui lòng chọn loại dữ liệu để tìm kiếm");

                    }

                }

                catch (*Exception* ex)

                {

                    Console.WriteLine("Connection failed: " + ex.Message);

                }

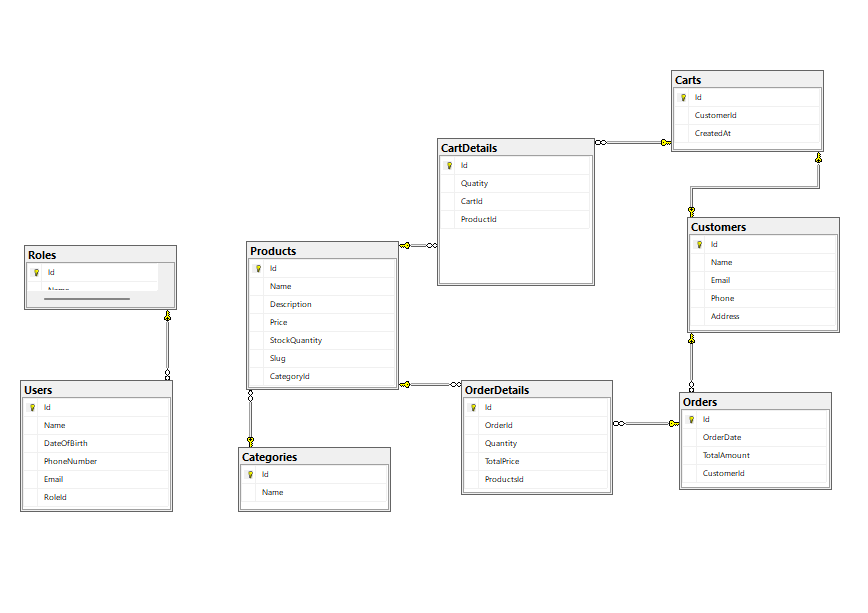
            }

        }

2.3 ASP.NET CORE

2.3.1 Tạo cở sở dữ liệu hoàn chỉnh.

Cở sở dữ liệu của ASP.NET CORE tương tự với winform application



2.3.2 API

2.3.2.1 Controller.

2.3.2.1.1 CartController.

1. Phương thức GetCartDetails(int cartId) được đánh dấu bằng [HttpGet("{cartId}/details")] để chỉ định rằng nó xử lý yêu cầu HTTP GET với đường dẫn URL có định dạng "/cart/{cartId}/details". Phương thức này gọi phương thức GetCartDetails của đối tượng \_cartService để lấy thông tin chi tiết của giỏ hàng với cartId tương ứng.

2. Phương thức AddToCart(int cartId, [FromBody] int productId) được đánh dấu bằng [HttpPost("{cartId}")] để chỉ định rằng nó xử lý yêu cầu HTTP POST với đường dẫn URL có định dạng "/cart/{cartId}". Phương thức này nhận một productId từ phần thân yêu cầu (request body) và gọi phương thức AddToCart của \_cartService để thêm sản phẩm có productId vào giỏ hàng với cartId tương ứng.

3. Phương thức RemoveFromCart(int cartId, int cartDetailId) được đánh dấu bằng [HttpDelete("{cartDetailId}")] để chỉ định rằng nó xử lý yêu cầu HTTP DELETE với đường dẫn URL có định dạng "/cart/{cartDetailId}". Phương thức này gọi phương thức RemoveFromCart của \_cartService để xóa một mục hàng trong giỏ hàng (cartDetailId) của giỏ hàng có cartId tương ứng.

4. Các phương thức GetCartByCustomerId() và RemoveFromCart() đã bị tắt bằng cách đặt dấu chú thích (comment) trong mã để tạm thời không sử dụng

namespace *Services*.*API*.*Controllers*

{

      [*ApiController*]

      [*Route*("[controller]")]

      public class CartController : *ControllerBase*

      {

            private readonly *CartService* \_cartService;

            public CartController(*MyDbContext* *db*)

            {

                  \_cartService = new *CartService*(db);

            }

            //[HttpGet]

            //public IActionResult GetCartByCustomerId()

            //{

            //    //var cart = \_cartService.GetCartByPhone(phone);

            //    return Ok();

            //}

            [*HttpGet*("{cartId}/details")]

            public *IActionResult* GetCartDetails(*int* *cartId*)

            {

                  try

                  {

                        var cartDetails = \_cartService.GetCartDetails(cartId);

                        return Ok(cartDetails);

                  }

                  catch (*Exception* ex)

                  {

                        return StatusCode(500, $"Internal server error: {ex.Message}");

                  }

            }

            [*HttpPost*("{cartId}")]

            public *IActionResult* AddToCart(*int* *cartId*, [*FromBody*] *int* *productId*)

            {

                  try

                  {

                        \_cartService.AddToCart(cartId, productId);

                        return Ok("Item added to cart successfully.");

                  }

                  catch (*Exception* ex)

                  {

                        return StatusCode(500, $"Internal server error: {ex.Message}");

                  }

            }

            [*HttpDelete*("{cartDetailId}")]

            public *IActionResult* RemoveFromCart(*int* *cartId*, *int* *cartDetailId*)

            {

                  try

                  {

                        \_cartService.RemoveFromCart(cartId, cartDetailId);

                        return Ok("Item removed from cart successfully.");

                  }

                  catch (*Exception* ex)

                  {

                        return StatusCode(500, $"Internal server error: {ex.Message}");

                  }

            }

      }

}

2.3.2.1.2 CategoryController.

1. `[ApiController]` và `[Route("[controller]")]`:

- `[ApiController]` đánh dấu lớp `CategoryController` là một controller trong Web API.

- `[Route("[controller]")]` đặt một tiền tố cho tất cả các route của controller dựa trên tên của controller. Trong trường hợp này, `[controller]` sẽ thay thế bằng "Category" trong các route.

2. Constructor:

- Constructor này nhận một đối tượng `MyDbContext` (được giả định là một DbContext của Entity Framework) thông qua dependency injection. Điều này đảm bảo rằng `CategoryController` có quyền truy cập đến một instance của `MyDbContext` để tương tác với cơ sở dữ liệu.

3. `GetCategories()` (HTTP GET):

- Phương thức này xử lý yêu cầu HTTP GET tới `/Category`.

- Nó gọi `\_categoryService.GetCategories()` để truy xuất danh sách các thực thể "Category" từ cơ sở dữ liệu và chuyển chúng thành danh sách `CategoryResponse`.

- Sau đó, nó trả về danh sách các "Category" dưới dạng kết quả OK.

4. `Create(CategoryRequest category)` (HTTP POST):

- Phương thức này xử lý yêu cầu HTTP POST tới `/Category`.

- Nó kiểm tra xem đối tượng `category` được gửi lên có giá trị null không và sau đó tạo một thực thể "Category" từ dữ liệu trong `category`.

- Nó gọi `\_categoryService.Create(c)` để lưu thực thể "Category" mới vào cơ sở dữ liệu và trả về thông báo thành công hoặc lỗi.

5. `Update(int id, CategoryRequest category)` (HTTP PUT):

- Phương thức này xử lý yêu cầu HTTP PUT tới `/Category`.

- Nó kiểm tra xem đối tượng `category` có giá trị null không và kiểm tra xem `id` trong URL trùng với `Id` của `category` hay không.

- Nếu thông tin hợp lệ, nó cập nhật thực thể "Category" trong cơ sở dữ liệu và trả về thông báo thành công hoặc lỗi.

6. `GetById(int id)` (HTTP GET):

- Phương thức này xử lý yêu cầu HTTP GET tới `/Category/{id}`.

- Nó gọi `\_categoryService.GetById(id)` để lấy thông tin một thực thể "Category" dựa trên `id` và trả về dưới dạng `CategoryResponse`.

7. `Delete(int id)` (HTTP DELETE):

- Phương thức này xử lý yêu cầu HTTP DELETE tới `/Category/{id}`.

- Nó gọi `\_categoryService.Delete(id)` để xóa thực thể "Category" dựa trên `id` và trả về thông báo thành công hoặc lỗi.

namespace *Services*.*API*.*Controllers*

{

      [*ApiController*]

      [*Route*("[controller]")]

      public class CategoryController : *ControllerBase*

      {

            private readonly *CategoryService* \_categoryService;

            public CategoryController(*MyDbContext* *db*)

            {

                  \_categoryService = new *CategoryService*(db);

            }

            [*HttpGet*]

            public *IActionResult* GetCategories()

            {

                  var result = new *List*<*CategoryResponse*>();

                  foreach (var x in \_categoryService.GetCategories())

                  {

                        result.Add(new *CategoryResponse*() { Id = x.Id, Name = x.Name });

                  }

                  return Ok(result);

            }

            [*HttpPost*]

            public *IActionResult* Create([*FromBody*] *CategoryRequest* *category*)

            {

                  try

                  {

                        if (category == null)

                              return BadRequest("Category object is null");

                        var c = new *Category* { Id = category.Id, Name = category.Name };

                        \_categoryService.Create(c);

                        return Ok("Category created successfully.");

                  }

                  catch (*Exception* ex)

                  {

                        return StatusCode(500, $"Internal server error: {ex.Message}");

                  }

            }

            [*HttpPut*]

            public *IActionResult* Update(*int* *id*, [*FromBody*] *CategoryRequest* *category*)

            {

                  try

                  {

                        if (category == null || id != category.Id)

                              return BadRequest("Invalid category data");

                        var c = new *Category* { Id = category.Id, Name = category.Name };

                        \_categoryService.Update(c);

                        return Ok("Category updated successfully.");

                  }

                  catch (*Exception* ex)

                  {

                        return StatusCode(500, $"Internal server error: {ex.Message}");

                  }

            }

            [*HttpGet*("{id}")]

            public *IActionResult* GetById(*int* *id*)

            {

                  try

                  {

                        var category = \_categoryService.GetById(id);

                        return Ok(new *CategoryResponse*() { Id = category.Id, Name = category.Name });

                  }

                  catch (*Exception* ex)

                  {

                        return StatusCode(500, $"Internal server error: {ex.Message}");

                  }

            }

            [*HttpDelete*("{id}")]

            public *IActionResult* Delete(*int* *id*)

            {

                  try

                  {

                        \_categoryService.Delete(id);

                        return Ok("Category deleted successfully.");

                  }

                  catch (*Exception* ex)

                  {

                        return StatusCode(500, $"Internal server error: {ex.Message}");

                  }

            }

      }

}

2.3.2.1.3 CustomerController.

1. `CustomerController` là một lớp điều khiển (controller) trong MVC và được đánh dấu với các thuộc tính `[Route("api/[controller]")]` và `[ApiController]`. Thuộc tính `[Route("api/[controller]")]` định nghĩa đường dẫn URL mà các phương thức trong điều khiển này sẽ xử lý. Thuộc tính `[ApiController]` cho biết lớp này là một điều khiển API.

2. Trong hàm tạo (`CustomerController`), một đối tượng `CustomerService` được khởi tạo với một đối tượng `MyDbContext` được truyền vào. Có thể suy ra rằng `CustomerService` là một lớp được sử dụng để xử lý logic liên quan đến khách hàng.

3. Phương thức `GetAllCustomers()` được đánh dấu bằng `[HttpGet]` để chỉ định rằng nó xử lý yêu cầu HTTP GET với đường dẫn URL "/api/customer". Phương thức này gọi phương thức `GetAllCustomers` của đối tượng `\_customerService` để lấy danh sách tất cả khách hàng.

4. Phương thức `GetCustomerById(int id)` được đánh dấu bằng `[HttpGet("{id}")]` để chỉ định rằng nó xử lý yêu cầu HTTP GET với đường dẫn URL "/api/customer/{id}". Phương thức này nhận một `id` từ đường dẫn URL và gọi phương thức `GetCustomerById` của `\_customerService` để lấy thông tin khách hàng với `id` tương ứng.

5. Phương thức `AddCustomer([FromBody] CustomerRequest customer)` được đánh dấu bằng `[HttpPost]` để chỉ định rằng nó xử lý yêu cầu HTTP POST với đường dẫn URL "/api/customer". Phương thức này nhận một đối tượng `CustomerRequest` từ phần thân yêu cầu (request body) và gọi phương thức `AddCustomer` của `\_customerService` để thêm một khách hàng mới vào cơ sở dữ liệu.

6. Phương thức `UpdateUserPassword(int userId, [FromBody] PasswordUpdateRequest passwordUpdateRequest)` được đánh dấu bằng `[HttpPut("{userId}/update-password")]` để chỉ định rằng nó xử lý yêu cầu HTTP PUT với đường dẫn URL "/api/customer/{userId}/update-password". Phương thức này nhận một `userId` từ đường dẫn URL và một đối tượng `PasswordUpdateRequest` từ phần thân yêu cầu, và gọi phương thức `UpdateCustomer` của `\_customerService` để cập nhật mật khẩu của khách hàng với `userId` tương ứng.

7. Phương thức `UpdateCustomer(int id, [FromBody] CustomerRequest customer)` được đánh dấu bằng `[HttpPut("{id}")]` để chỉ định rằng nó xử lý yêu cầu HTTP PUT với đường dẫn URL "/api/customer/{id}". Phương thức này nhận một `id` từ đường dẫn URL và một đối tượng `CustomerRequest` từ phần thân yêu cầu, và gọi phương thức `UpdateCustomer` của `\_customerService` để cập nhật thông tin khách hàng với `id` tương ứng.

8. Phương thức `DeleteCustomer(int id)` được đánh dấu bằng `[HttpDelete("{id}")]` để chỉ định rằng nó xử lý yêu cầu HTTP DELETE với đường dẫn URL "/api/customer/{id}". Phương thức này nhận một `id` từ đường dẫn URL và gọi phương thức `DeleteCustomer` của `\_customerService` để xóa khách hàng với `id` tương ứng.

namespace *Services*.*API*.*Controller*

{

      [*Route*("api/[controller]")]

      [*ApiController*]

      public class CustomerController : *ControllerBase*

      {

            private readonly *CustomerService* \_customerService;

            public CustomerController(*MyDbContext* *db*)

            {

                  \_customerService = new *CustomerService*(db);

            }

            [*HttpGet*]

            public *IActionResult* GetAllCustomers()

            {

                  try

                  {

                        var customers = \_customerService.GetAllCustomers();

                        return Ok(customers);

                  }

                  catch (*Exception* ex)

                  {

                        return StatusCode(500, $"Internal server error: {ex.Message}");

                  }

            }

            [*HttpGet*("{id}")]

            public *IActionResult* GetCustomerById(*int* *id*)

            {

                  try

                  {

                        //var customer = \_customerService.GetCustomerById(id);

                        return Ok();

                  }

                  catch (*Exception* ex)

                  {

                        return StatusCode(500, $"Internal server error: {ex.Message}");

                  }

            }

            [*HttpPost*]

            public *IActionResult* AddCustomer([*FromBody*] *CustomerRequest* *customer*)

            {

                  if (!customer.Password.Equals(customer.ConfirmPassword))

                  {

                        return BadRequest("Confirm passowrd doesn't match password");

                  }

                  try

                  {

                        if (customer == null)

                              return BadRequest("Customer object is null");

                        var newCus = new *Customer*()

                        {

                              Name = customer.FullName,

                              Address = customer.Address,

                              Email = customer.Email,

                              Phone = customer.Phone,

                              Password = Generated.GenerateHashedPassword(customer.Password)

                        };

                        \_customerService.AddCustomer(newCus);

                        return Ok("Customer added successfully.");

                  }

                  catch (*Exception* ex)

                  {

                        return StatusCode(500, $"Internal server error: {ex.Message}");

                  }

            }

            [*HttpPut*("{userId}/update-password")]

            public *IActionResult* UpdateUserPassword(*int* *userId*, [*FromBody*] *PasswordUpdateRequest* *passwordUpdateRequest*)

            {

                  try

                  {

                        //var user = \_customerService.GetCustomerById(userId);

                        //if (user == null)

                        //    return NotFound("User not found.");

                        //string hashedOldPassword = Generated.GenerateHashedPassword(passwordUpdateRequest.OldPassword);

                        //if (hashedOldPassword != user.Password)

                        //    return BadRequest("Old password is incorrect.");

                        //string hashedNewPassword = Generated.GenerateHashedPassword(passwordUpdateRequest.NewPassword);

                        //user.Password = hashedNewPassword;

                        //\_customerService.UpdateCustomer(user);

                        return Ok("User password updated successfully.");

                  }

                  catch (*Exception* ex)

                  {

                        return StatusCode(500, $"Internal server error: {ex.Message}");

                  }

            }

            [*HttpPut*("{id}")]

            public *IActionResult* UpdateCustomer(*int* *id*, [*FromBody*] *CustomerRequest* *customer*)

            {

                  try

                  {

                        if (customer == null)

                              return BadRequest("Customer object is null");

                        if (id != customer.Id)

                              return BadRequest("Customer Id mismatch");

                        var newCus = new *Customer*()

                        {

                              Name = customer.FullName,

                              Address = customer.Address,

                              Email = customer.Email,

                              Phone = customer.Phone,

                              Password = Generated.GenerateHashedPassword(customer.Password)

                        };

                        \_customerService.UpdateCustomer(newCus);

                        return Ok("Customer updated successfully.");

                  }

                  catch (*Exception* ex)

                  {

                        return StatusCode(500, $"Internal server error: {ex.Message}");

                  }

            }

            [*HttpDelete*("{id}")]

            public *IActionResult* DeleteCustomer(*int* *id*)

            {

                  try

                  {

                        \_customerService.DeleteCustomer(id);

                        return Ok("Customer deleted successfully.");

                  }

                  catch (*Exception* ex)

                  {

                        return StatusCode(500, $"Internal server error: {ex.Message}");

                  }

            }

      }

}

2.3.2.1.4 ProductController.

1. `[ApiController]` và `[Route("api/[controller]")]`:

- `[ApiController]` đánh dấu lớp `ProductController` là một controller trong Web API.

- `[Route("api/[controller]")]` xác định tiền tố cho tất cả các route của controller, bắt đầu bằng "api" và tiếp theo là tên của controller, trong trường hợp này là "Product".

2. Constructor:

- Constructor nhận một đối tượng `MyDbContext` thông qua dependency injection. Điều này đảm bảo rằng `ProductController` có quyền truy cập đến một instance của `MyDbContext` để tương tác với cơ sở dữ liệu.

3. `GetProducts()` (HTTP GET):

- Phương thức này xử lý yêu cầu HTTP GET tới `/api/Product`.

- Nó gọi `\_productService.GetAll()` để truy xuất danh sách tất cả các thực thể "Product" từ cơ sở dữ liệu và trả về danh sách này dưới dạng kết quả OK.

4. `GetProductsByPage(int pageNumber, int pageSize)` (HTTP GET):

- Phương thức này xử lý yêu cầu HTTP GET tới `/api/Product/page` và nhận tham số `pageNumber` và `pageSize`.

- Nó gọi `\_productService.GetProductsByPage(pageNumber, pageSize)` để lấy danh sách sản phẩm theo trang và trả về kết quả dưới dạng OK hoặc thông báo lỗi.

5. `FilterProducts()` (HTTP GET):

- Phương thức này xử lý yêu cầu HTTP GET tới `/api/Product/filter` và nhận các tham số truy vấn (query parameters) để lọc sản phẩm theo giá, số lượng tồn kho và danh mục.

- Nó gọi `\_productService.FilterProducts(...)` để lọc sản phẩm dựa trên các thông số truyền vào và trả về danh sách sản phẩm sau khi lọc.

6. `CreateProduct([FromBody] ProductRequest product)` (HTTP POST):

- Phương thức này xử lý yêu cầu HTTP POST tới `/api/Product`.

- Nó kiểm tra xem đối tượng `product` gửi lên có giá trị null không. Nếu hợp lệ, nó gọi `\_productService.CreateProduct(...)` để tạo sản phẩm mới và trả về kết quả OK hoặc thông báo lỗi.

7. `UpdateProduct(int id, [FromBody] ProductRequest product)` (HTTP PUT):

- Phương thức này xử lý yêu cầu HTTP PUT tới `/api/Product/{id}` và nhận một `id` để xác định sản phẩm cần cập nhật.

- Nó gọi `\_productService.UpdateProduct(...)` để cập nhật thông tin sản phẩm dựa trên `id` và dữ liệu mới được gửi lên. Sau đó, nó trả về kết quả OK.

8. `DeleteProduct(int id)` (HTTP DELETE):

- Phương thức này xử lý yêu cầu HTTP DELETE tới `/api/Product/{id}` và nhận một `id` để xác định sản phẩm cần xóa.

- Nó gọi `\_productService.DeleteProduct(id)` để xóa sản phẩm dựa trên `id` và trả về kết quả OK.

namespace *Services*.*API*.*Controller*

{

      [*ApiController*]

      [*Route*("api/[controller]")]

      public class ProductController : *ControllerBase*

      {

            private *ProductService* \_productService;

            public ProductController(*MyDbContext* *db*)

            {

                  \_productService = new *ProductService*(db);

            }

            // GET

            [*HttpGet*]

            public *ActionResult* GetProducts()

            {

                  var products = \_productService.GetAll();

                  return Ok(products);

            }

            // GET

            [*HttpGet*("page")]

            public *IActionResult* GetProductsByPage(*int* *pageNumber*, *int* *pageSize*)

            {

                  try

                  {

                        var products = \_productService.GetProductsByPage(pageNumber, pageSize);

                        return Ok(products);

                  }

                  catch (*Exception* ex)

                  {

                        return StatusCode(500, "An error occurred while fetching the products.");

                  }

            }

            // GET

            [*HttpGet*("filter")]

            public *IActionResult* FilterProducts([*FromQuery*] *decimal*? *minPrice*, [*FromQuery*] *decimal*? *maxPrice*,

                                                                  [*FromQuery*] *int*? *minStockQuantity*, [*FromQuery*] *int*? *maxStockQuantity*,

                                                                  [*FromQuery*] *int*? *categoryId*)

            {

                  try

                  {

                        var products = \_productService.FilterProducts(minPrice, maxPrice,

                                                                                            minStockQuantity,

                                                                                            maxStockQuantity,

                                                                                            categoryId);

                        //var response = new ApiResponse<IEnumerable<Product>>

                        //{

                        //    Success = true,

                        //    Data = products,

                        //    ErrorMessage = null

                        //};

                        return Ok(products);

                  }

                  catch (*Exception* ex)

                  {

                        //var response = new ApiResponse<IEnumerable<Product>>

                        //{

                        //    Success = false,

                        //    Data = null,

                        //    ErrorMessage = "An error occurred while filtering products."

                        //};

                        return StatusCode(500);

                  }

            }

            // POST

            [*HttpPost*]

            public *ActionResult* CreateProduct([*FromBody*] *ProductRequest* *product*)

            {

                  if (product == null)

                  {

                        return BadRequest("Invalid product data");

                  }

                  try

                  {

                        \_productService.CreateProduct(product.Name,

                                          product.Description,

                                          product.Price,

                                          product.StockQuantity,

                                          product.CategoryId);

                        return Ok();

                  }

                  catch (*Exception* ex)

                  {

                        return StatusCode(500, "An error occurred while creating the product.");

                  }

            }

            // PUT

            [*HttpPut*("{id}")]

            public *IActionResult* UpdateProduct(*int* *id*, [*FromBody*] *ProductRequest* *product*)

            {

                  \_productService.UpdateProduct(id, product.Name,

                                          product.Description,

                                          product.Price,

                                          product.StockQuantity,

                                          product.CategoryId);

                  return Ok();

            }

            // DELETE

            [*HttpDelete*("{id}")]

            public *IActionResult* DeleteProduct(*int* *id*)

            {

                  \_productService.DeleteProduct(id);

                  return Ok();

            }

      }

}

2.3.2.1.5 UserController.

1. `UserController` là một lớp điều khiển (controller) trong MVC và được đánh dấu với các thuộc tính `[Route("[controller]")]` và `[ApiController]`. Thuộc tính `[Route("[controller]")]` định nghĩa đường dẫn URL mà các phương thức trong điều khiển này sẽ xử lý. Thuộc tính `[ApiController]` cho biết lớp này là một điều khiển API.

2. Trong hàm tạo (`UserController`), một đối tượng `UserService` được khởi tạo với một đối tượng `MyDbContext` được truyền vào. Có thể suy ra rằng `UserService` là một lớp được sử dụng để xử lý logic liên quan đến người dùng.

3. Phương thức `GetAllUsers()` được đánh dấu bằng `[HttpGet]` để chỉ định rằng nó xử lý yêu cầu HTTP GET với đường dẫn URL "/user". Phương thức này gọi phương thức `GetAllUsers` của đối tượng `\_userService` để lấy danh sách tất cả người dùng.

4. Phương thức `GetUserByPhone(string phoneNumber)` được đánh dấu bằng `[HttpGet("{phoneNumber}")]` để chỉ định rằng nó xử lý yêu cầu HTTP GET với đường dẫn URL "/user/{phoneNumber}". Phương thức này nhận một `phoneNumber` từ đường dẫn URL và gọi phương thức `GetUserByPhone` của `\_userService` để lấy thông tin người dùng với số điện thoại tương ứng.

5. Phương thức `GetUserByEmail(string email)` được đánh dấu bằng `[HttpGet("email/{email}")]` để chỉ định rằng nó xử lý yêu cầu HTTP GET với đường dẫn URL "/user/email/{email}". Phương thức này nhận một `email` từ đường dẫn URL và gọi phương thức `GetUserByEmail` của `\_userService` để lấy thông tin người dùng với email tương ứng.

6. Phương thức `AddUser([FromBody] UserRequest userRequest)` được đánh dấu bằng `[HttpPost]` để chỉ định rằng nó xử lý yêu cầu HTTP POST với đường dẫn URL "/user". Phương thức này nhận một đối tượng `UserRequest` từ phần thân yêu cầu (request body) và gọi phương thức `AddUser` của `\_userService` để thêm một người dùng mới vào cơ sở dữ liệu.

7. Phương thức `UpdateUserPassword(int userId, [FromBody] string newPassword)` được đánh dấu bằng `[HttpPut("{userId}/update-password")]` để chỉ định rằng nó xử lý yêu cầu HTTP PUT với đường dẫn URL "/user/{userId}/update-password". Phương thức này nhận một `userId` từ đường dẫn URL và mật khẩu mới từ phần thân yêu cầu và gọi phương thức `UpdateUserPassword` của `\_userService` để cập nhật mật khẩu của người dùng với `userId` tương ứng.

8. Phương thức `UpdateUserRole(int userId, int roleId)` được đánh dấu bằng `[HttpPut("{userId}/update-role/{roleId}")]` để chỉ định rằng nó xử lý yêu cầu HTTP PUT với đường dẫn URL "/user/{userId}/update-role/{roleId}". Phương thức này nhận một `userId` và một `roleId` từ đường dẫn URL và gọi phương thức `UpdateUserRole` của `\_userService` để cập nhật vai trò của người dùng với `userId` tương ứng.

9. Phương thức `UpdateUser([FromBody] User user)` được đánh dấu bằng `[HttpPut]` để chỉ định rằng nó xử lý yêu cầu HTTP PUT với đường dẫn URL "/user". Phương thức này nhận một đối tượng `User` từ phần thân yêu cầu và gọi phương thức `UpdateUser` của `\_userService` để cập nhật thông tin người dùng.

10. Phương thức `DeleteUser(string phoneNumber)` được đánh dấu bằng `[HttpDelete("{phoneNumber}")]` để chỉ định rằng nó xử lý yêu cầu HTTP DELETE với đường dẫn URL "/user/{phoneNumber}". Phương thức này nhận một `phoneNumber` từ đường dẫn URL và gọi phương thức `DeleteUser` của `\_userService` để xóa người dùng có số điện thoại tương ứng.

namespace *Services*.*API*.*Controllers*

{

      [*ApiController*]

      [*Route*("[controller]")]

      public class UserController : *ControllerBase*

      {

            private readonly *UserService* \_userService;

            public UserController(*MyDbContext* *db*)

            {

                  \_userService = new *UserService*(db);

            }

            [*HttpGet*]

            public *IActionResult* GetAllUsers()

            {

                  try

                  {

                        var users = \_userService.GetAllUsers();

                        return Ok(users);

                  }

                  catch (*Exception* ex)

                  {

                        return StatusCode(500, $"Internal server error: {ex.Message}");

                  }

            }

            [*HttpGet*("{phoneNumber}")]

            public *IActionResult* GetUserByPhone(*string* *phoneNumber*)

            {

                  try

                  {

                        var user = \_userService.GetUserByPhone(phoneNumber);

                        return Ok(user);

                  }

                  catch (*Exception* ex)

                  {

                        return StatusCode(500, $"Internal server error: {ex.Message}");

                  }

            }

            [*HttpGet*("email/{email}")]

            public *IActionResult* GetUserByEmail(*string* *email*)

            {

                  try

                  {

                        var user = \_userService.GetUserByEmail(email);

                        return Ok(user);

                  }

                  catch (*Exception* ex)

                  {

                        return StatusCode(500, $"Internal server error: {ex.Message}");

                  }

            }

            [*HttpPost*]

            public *IActionResult* AddUser([*FromBody*] *UserRequest* *userRequest*)

            {

                  if (userRequest == null)

                        return BadRequest("User request object is null");

                  // Check if password and confirm password match

                  if (userRequest.Password != userRequest.ConfirmPassword)

                        return BadRequest("Password and ConfirmPassword do not match");

                  var user = new *User*

                  {

                        Id = userRequest.Id,

                        Name = userRequest.FullName,

                        Email = userRequest.Email,

                        PhoneNumber = userRequest.Phone,

                        DateOfBirth = DateTime.ParseExact(userRequest.Dob, "dd/MM/yyyy", CultureInfo.InvariantCulture),

                        Password = Generated.GenerateHashedPassword(userRequest.Password),

                  };

                  try

                  {

                        if (user == null)

                              return BadRequest("User object is null");

                        \_userService.AddUser(user, userRequest.RoleId);

                        return Ok("User added successfully.");

                  }

                  catch (*Exception* ex)

                  {

                        return StatusCode(500, $"Internal server error: {ex.StackTrace}");

                  }

            }

            [*HttpPut*("{userId}/update-password")]

            public *IActionResult* UpdateUserPassword(*int* *userId*, [*FromBody*] *string* *newPassword*)

            {

                  try

                  {

                        \_userService.UpdateUserPassword(userId, Generated.GenerateHashedPassword(newPassword));

                        return Ok("User password updated successfully.");

                  }

                  catch (*Exception* ex)

                  {

                        return StatusCode(500, $"Internal server error: {ex.Message}");

                  }

            }

            [*HttpPut*("{userId}/update-role/{roldeId}")]

            public *IActionResult* UpdateUserRole(*int* *userId*, *int* *roleId*)

            {

                  try

                  {

                        \_userService.UpdateUserRole(userId, roleId);

                        return Ok("User role updated successfully.");

                  }

                  catch (*Exception* ex)

                  {

                        return StatusCode(500, $"Internal server error: {ex.Message}");

                  }

            }

            [*HttpPut*]

            public *IActionResult* UpdateUser([*FromBody*] *User* *user*)

            {

                  try

                  {

                        if (user == null)

                              return BadRequest("User object is null");

                        \_userService.UpdateUser(user);

                        return Ok("User updated successfully.");

                  }

                  catch (*Exception* ex)

                  {

                        return StatusCode(500, $"Internal server error: {ex.Message}");

                  }

            }

            [*HttpDelete*("{phoneNumber}")]

            public *IActionResult* DeleteUser(*string* *phoneNumber*)

            {

                  try

                  {

                        \_userService.DeleteUser(phoneNumber);

                        return Ok("User deleted successfully.");

                  }

                  catch (*Exception* ex)

                  {

                        return StatusCode(500, $"Internal server error: {ex.Message}");

                  }

            }

      }

}

2.3.2.2 Request Entities.

2.3.2.2.1 CategoryRequest.

public class CategoryRequest

      {

            public *int* Id { get; set; }

            public *string* Name { get; set; }

      }

2.3.2.2.2 CustomerRequest.

public class CustomerRequest

      {

            public required *int* Id { get; set; }

            public required *string* FullName { get; set; }

            public required *string* Email { get; set; }

            public required *string* Phone { get; set; }

            public required *string* Address { get; set; }

            public required *string* Password { get; set; }

            public required *string* ConfirmPassword { get; set; }

      }

}

2.3.2.2.3 PasswordUpdateRequest.

public class PasswordUpdateRequest

      {

            public *string* OldPassword { get; set; }

            public *string* NewPassword { get; set; }

      }

2.3.2.2.4 Payment

public class Payment

      {

      }

2.3.2.2.5 ProductRequest.

public class ProductRequest

      {

            public *int* Id { get; set; }

            public required *string* Name { get; set; }

            public required *string* Description { get; set; }

            public *decimal* Price { get; set; }

            public *int* StockQuantity { get; set; }

            public *int* CategoryId { get; set; }

      }

2.3.2.2.6 UserRequest.

public class UserRequest

      {

            public required *int* Id { get; set; }

            public required *string* FullName { get; set; }

            public required *string* Email { get; set; }

            public required *string* Dob { get; set; }

            public required *string* Phone { get; set; }

            public required *string* Address { get; set; }

            public required *int* RoleId { get; set; }

            public required *string* Password { get; set; }

            public required *string* ConfirmPassword { get; set; }

      }

}

2.3.2.3 Response Entities.

2.3.2.3.1 CartDetailResponse.

public class CartDetailResponse

      {

            public *int* Id { get; set; }

            public required *ProductDTO* Product { get; set; }

            public *int* Quatity { get; set; }

      }

2.3.2.3.2 CategoryResponse.

public class CategoryResponse

      {

            public *int* Id { get; set; }

            public *string* Name { get; set; }

      }

2.3.2.3.3 OrderDetailResponse.

public class OrderDetailResponse

      {

            public *int* Id { get; set; }

            public *ProductDTO* Product { get; set; }

            public *int* Quantity { get; set; }

            public *decimal* TotalPrice { get; set; }

      }

2.3.2.3.4 ProductResponse.

public class ProductDTO

      {

            public *int* Id { get; set; }

            public required *string*? Name { get; set; }

            public required *string*? Description { get; set; }

            public *decimal* Price { get; set; }

            public required *long* StockQuatity { get; set; }

            public required *string* Slug { get; set; }

            public required *CategoryResponse* Category { get; set; }

            public ProductDTO(*Product* *p*)

            {

                  Id = p.Id;

                  Name = p.Name;

                  Description = p.Description;

                  Price = p.Price;

                  StockQuatity = p.StockQuantity;

                  Slug = p.Slug;

                  Category = new *CategoryResponse*() { Id = p.Category.Id, Name = p.Category.Name };

            }

            public *string* GenerateSlug()

            {

*string* slug = Name?.ToLower().Replace(" ", "-");

                  slug = new *string*(slug?.Where(*c* => *char*.IsLetterOrDigit(c) || c == '-').ToArray());

                  return slug?.Trim('-'); ;

            }

      }

2.3.3 Repository.

2.3.3.1 CartRepository.

1. Hàm tạo (`CartRepository`) nhận một đối tượng `MyDbContext` và khởi tạo đối tượng `ProductRepository` bằng cách truyền đối tượng `db` vào. Điều này cho phép `CartRepository` sử dụng các phương thức trong `ProductRepository` để truy vấn thông tin sản phẩm.

2. Phương thức `GetCartByCustomerId(int customerId)` truy vấn và trả về đối tượng `Cart` theo `customerId` (ID của khách hàng). Nó tìm kiếm và trả về đối tượng `Cart` đầu tiên trong bảng `Carts` trong cơ sở dữ liệu mà có `Customer.Id` tương ứng với `customerId`.

3. Phương thức `GetById(int Id)` truy vấn và trả về đối tượng `Cart` theo `Id` (ID của giỏ hàng). Nó tìm kiếm và trả về đối tượng `Cart` đầu tiên trong bảng `Carts` trong cơ sở dữ liệu mà có `Id` tương ứng với `Id` được truyền vào.

4. Phương thức `Create(Customer c)` tạo một đối tượng `Cart` mới trong cơ sở dữ liệu. Nó thêm một đối tượng `Cart` mới vào bảng `Carts` với thông tin khách hàng (`Customer`) được truyền vào và thời gian tạo (`CreatedAt`) được đặt là thời điểm hiện tại.

5. Phương thức `UpdateItem(int cartId, int productId, int quan)` cập nhật số lượng (`Quantity`) của một sản phẩm trong giỏ hàng. Nó tìm kiếm và cập nhật đối tượng `CartDetail` đầu tiên trong bảng `CartDetails` có `Product.Id` và `CartId` tương ứng với `productId` và `cartId` được truyền vào.

6. Phương thức `GetCartDetails(int cartId)` truy vấn và trả về danh sách các đối tượng `CartDetail` từ giỏ hàng. Nó lấy tất cả các đối tượng `CartDetail` trong bảng `CartDetails` trong cơ sở dữ liệu mà có `CartId` tương ứng với `cartId` được truyền vào.

7. Phương thức `AddToCart(int cartId, int productId)` thêm một sản phẩm vào giỏ hàng. Nó truy xuất thông tin sản phẩm từ `ProductRepository` bằng cách sử dụng phương thức `GetProductById(productId)` và sau đó tạo một đối tượng `CartDetail` mới với thông tin giỏ hàng (`CartId`), sản phẩm và số lượng là 1. Đối tượng `CartDetail` mới được thêm vào bảng `CartDetails` trong cơ sở dữ liệu.

8. Phương thức `RemoveFromCart(int cartId, int productId)` xóa một sản phẩm khỏi giỏ hàng. Nó tìm kiếm và xóa đối tượng `CartDetail` đầu tiên trong bảng `CartDetails` có `CartId` và `Product.Id` tương ứng với `cartId` và `productId` được truyền vào.

namespace *Services*.*Repository*

{

      public class CartRepository

      {

            private readonly *MyDbContext* \_db;

            private readonly *ProductRepository* \_productRepository;

            public CartRepository(*MyDbContext* *db*)

            {

                  \_db = db;

                  \_productRepository = new *ProductRepository*(db);

            }

            // get infor from cart by phone of customer

            public *Cart*? GetCartByCustomerId(*int* *customerId*)

            {

                  return \_db.Carts.FirstOrDefault(*c* => c.Customer.Id == customerId);

            }

            // get by cart id

            public *Cart*? GetById(*int* *Id*)

            {

                  return \_db.Carts.FirstOrDefault(*c* => c.Id == Id);

            }

            // create the cart not cart detail

            public *void* Create(*Customer* *c*)

            {

                  \_db.Carts.Add(new *Cart*() { Customer = c, CreatedAt = DateTime.Now});

                  \_db.SaveChanges();

            }

            // updating product exist in cart

            public *void* UpdateItem(*int* *cartId*, *int* *productId*, *int* *quan*)

            {

                  var item = \_db.CartDetails.FirstOrDefault(*cd* => cd.Product.Id == productId && cd.CartId == cartId);

                  if (item != null)

                  {

                        item.Quantity = quan;

                        \_db.SaveChanges();

                  }

            }

            // get all product from a cart

            public *List*<*CartDetail*> GetCartDetails(*int* *cartId*)

            {

                  var res = \_db.CartDetails.Where(*c* => c.CartId == cartId).ToList();

                  return res;

            }

            public *void* AddToCart(*int* *cartId*, *int* *productId*)

            {

                  var product = \_productRepository.GetProductById(productId);

                  if (product != null)

                  {

                        var detail = new *CartDetail*() { CartId = cartId, Product = product, Quantity = 1 };

                        \_db.CartDetails.Add(detail);

                        \_db.SaveChanges();

                  }

            }

            public *void* RemoveFromCart(*int* *cartId*, *int* *productId*)

            {

                  var cartDetail = \_db.CartDetails.FirstOrDefault(*cd* => cd.CartId == cartId && cd.Product.Id == productId);

                  if (cartDetail != null)

                  {

                        \_db.CartDetails.Remove(cartDetail);

                        \_db.SaveChanges();

                  }

            }

      }

}

2.3.3.2 CategoryRepository.

1. Hàm tạo (CategoryRepository) nhận một đối tượng MyDbContext và lưu trữ đối tượng db để sử dụng cho các truy vấn cơ sở dữ liệu.

2. Phương thức GetAll() truy vấn và trả về một danh sách các đối tượng Category. Nó lấy tất cả các đối tượng Category từ bảng Categories trong cơ sở dữ liệu và chuyển chúng thành một danh sách.

3. Phương thức GetById(int id) truy vấn và trả về đối tượng Category dựa trên id (ID của danh mục). Nó tìm kiếm và trả về đối tượng Category đầu tiên trong bảng Categories trong cơ sở dữ liệu có Id tương ứng với id được truyền vào. Nếu không tìm thấy danh mục, nó sẽ ném một ngoại lệ với thông báo "Category not found.".

4. Phương thức Create(Category category) tạo một đối tượng Category mới trong cơ sở dữ liệu. Nó thêm đối tượng Category mới vào bảng Categories và lưu các thay đổi trong cơ sở dữ liệu.

5. Phương thức Update(Category category) cập nhật một đối tượng Category trong cơ sở dữ liệu. Nó tìm kiếm đối tượng Category hiện có bằng cách sử dụng phương thức GetById với category.Id và sau đó cập nhật thông tin Name của danh mục với thông tin từ category. Sau đó, nó cập nhật đối tượng Category trong bảng Categories và lưu các thay đổi trong cơ sở dữ liệu.

6. Phương thức Delete(int id) xóa một đối tượng Category khỏi cơ sở dữ liệu. Nó tìm kiếm và lấy đối tượng Category dựa trên id và sau đó xóa đối tượng đó khỏi bảng Categories và lưu các thay đổi trong cơ sở dữ liệu.

namespace *Services*.*Repository*

{

      public class CategoryRepository

      {

            private readonly *MyDbContext* \_db;

            public CategoryRepository(*MyDbContext* *db*)

            {

                  \_db = db;

            }

            public *List*<*Category*> GetAll()

            {

                  return \_db.Categories.ToList();

            }

            public *Category* GetById(*int* *id*)

            {

                  var category = \_db.Categories?.FirstOrDefault(*c* => c.Id == id);

                  if (category == null)

                  {

                        throw new *Exception*("Category not found.");

                  }

                  return category;

            }

            public *void* Create(*Category* *category*)

            {

                  \_db.Categories.Add(category);

                  \_db.SaveChanges();

            }

            public *void* Update(*Category* *category*)

            {

                  var existingCategory = GetById(category.Id);

                  if (existingCategory != null)

                  {

                        existingCategory.Name = category.Name;

                        \_db.Categories.Update(existingCategory);

                        \_db.SaveChanges();

                  }

            }

            public *void* Delete(*int* *id*)

            {

                  var cart = *this*.GetById(id);

                  \_db.Categories.Remove(cart);

                  \_db.SaveChanges();

            }

      }

}

2.3.3.3 Customer Repository.

1. Hàm tạo (`CustomerRepository`) nhận một đối tượng `MyDbContext` và lưu trữ đối tượng `db` để sử dụng cho các truy vấn cơ sở dữ liệu.

2. Phương thức `GetAllCustomers()` truy vấn và trả về một danh sách các đối tượng `Customer`. Nó lấy tất cả các đối tượng `Customer` từ bảng `Customers` trong cơ sở dữ liệu và chuyển chúng thành một danh sách.

3. Phương thức `GetCustomerById(int customerId)` truy vấn và trả về đối tượng `Customer` dựa trên `customerId` (ID của khách hàng). Nó tìm kiếm và trả về đối tượng `Customer` đầu tiên trong bảng `Customers` trong cơ sở dữ liệu có `Id` tương ứng với `customerId` được truyền vào.

4. Phương thức `GetCustomerByEmail(string email)` truy vấn và trả về đối tượng `Customer` dựa trên `email` (địa chỉ email của khách hàng). Nó tìm kiếm và trả về đối tượng `Customer` đầu tiên trong bảng `Customers` trong cơ sở dữ liệu có `Email` tương ứng với `email` được truyền vào.

5. Phương thức `GetCustomerByPhone(string phone)` truy vấn và trả về đối tượng `Customer` dựa trên `phone` (số điện thoại của khách hàng). Nó tìm kiếm và trả về đối tượng `Customer` đầu tiên trong bảng `Customers` trong cơ sở dữ liệu có `Phone` tương ứng với `phone` được truyền vào.

6. Phương thức `AddCustomer(Customer customer)` thêm một đối tượng `Customer` mới vào cơ sở dữ liệu. Nó thêm đối tượng `Customer` mới vào bảng `Customers` và lưu các thay đổi trong cơ sở dữ liệu.

7. Phương thức `UpdateCustomer(Customer customer)` cập nhật một đối tượng `Customer` trong cơ sở dữ liệu. Nó đánh dấu đối tượng `Customer` hiện có là đã được sửa đổi bằng cách thay đổi trạng thái của đối tượng thành `Modified`, sau đó lưu các thay đổi trong cơ sở dữ liệu.

8. Phương thức `DeleteCustomer(int customerId)` xóa một đối tượng `Customer` khỏi cơ sở dữ liệu. Nó tìm kiếm và lấy đối tượng `Customer` dựa trên `customerId` và sau đó xóa đối tượng đó khỏi bảng `Customers` và lưu các thay đổi trong cơ sở dữ liệu.

namespace *Services*.*Repository*

{

      public class CustomerRepository

      {

            private readonly *MyDbContext* \_db;

            public CustomerRepository(*MyDbContext* *db*)

            {

                  \_db = db;

            }

            public *IEnumerable*<*Customer*> GetAllCustomers()

            {

                  return \_db.Customers.ToList();

            }

            public *Customer*? GetCustomerById(*int* *customerId*)

            {

                  return \_db.Customers.FirstOrDefault(*c* => c.Id == customerId);

            }

            public *Customer*? GetCustomerByEmail(*string* *email*)

            {

                  return \_db.Customers.FirstOrDefault(*c* => c.Email == email);

            }

            public *Customer* GetCustomerByPhone(*string* *phone*)

            {

                  return \_db.Customers.FirstOrDefault(*c* => c.Phone.Equals(phone));

            }

            public *void* AddCustomer(*Customer* *customer*)

            {

                  \_db.Customers.Add(customer);

                  \_db.SaveChanges();

            }

            public *void* UpdateCustomer(*Customer* *customer*)

            {

                  \_db.Entry(customer).State = EntityState.Modified;

                  \_db.SaveChanges();

            }

            public *void* DeleteCustomer(*int* *customerId*)

            {

                  var customer = GetCustomerById(customerId);

                  if (customer != null)

                  {

                        \_db.Customers.Remove(customer);

                        \_db.SaveChanges();

                  }

            }

      }

}

2.3.3.4 MyDbContext.

1. Các thuộc tính `DbSet`:

- `Users`: Đại diện cho bảng `Users` trong cơ sở dữ liệu.

- `Roles`: Đại diện cho bảng `Roles` trong cơ sở dữ liệu.

- `Customers`: Đại diện cho bảng `Customers` trong cơ sở dữ liệu.

- `Categories`: Đại diện cho bảng `Categories` trong cơ sở dữ liệu.

- `Products`: Đại diện cho bảng `Products` trong cơ sở dữ liệu.

- `Orders`: Đại diện cho bảng `Orders` trong cơ sở dữ liệu.

- `OrderDetails`: Đại diện cho bảng `OrderDetails` trong cơ sở dữ liệu.

- `Carts`: Đại diện cho bảng `Carts` trong cơ sở dữ liệu.

- `CartDetails`: Đại diện cho bảng `CartDetails` trong cơ sở dữ liệu.

2. Hàm tạo (`MyDbContext`) có hai phiên bản:

- Phiên bản thứ nhất nhận đối tượng `DbContextOptions` và gọi hàm tạo của lớp cơ sở (`DbContext`) với tham số `options`.

- Phiên bản thứ hai không nhận tham số và sử dụng `DbContextOptionsBuilder` để cấu hình các tùy chọn kết nối cơ sở dữ liệu SQL Server. Trong trường hợp này, chuỗi kết nối được cung cấp là "Data Source=.;Initial Catalog=dotnet\_QT1;Integrated Security=True".

3. Phương thức `OnConfiguring(DbContextOptionsBuilder optionsBuilder)` được gọi khi cấu hình DbContext. Nó sử dụng `UseLazyLoadingProxies` để sử dụng cung cấp trì hoãn và `UseSqlServer` để cấu hình tùy chọn kết nối cơ sở dữ liệu.

4. Phương thức `OnModelCreating(ModelBuilder modelBuilder)` được gọi khi xây dựng mô hình dữ liệu. Trong trường hợp này, nó định nghĩa một chỉ mục duy nhất trên trường `Phone` trong bảng `Customers` bằng cách sử dụng phương thức `HasIndex` và `IsUnique`.

namespace *Services*.*Repository*

{

      public class MyDbContext : *DbContext*

      {

            public *DbSet*<*User*> Users { get; set; }

            public *DbSet*<*Role*> Roles { get; set; }

            public *DbSet*<*Customer*> Customers { get; set; }

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

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

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

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

            public *DbSet*<*Cart*> Carts { get; set; }

            public *DbSet*<*CartDetail*> CartDetails { get; set; }

            public MyDbContext(*DbContextOptions* *options*) : base(options)

            {

            }

            public MyDbContext() : base(new *DbContextOptionsBuilder*<*MyDbContext*>()

                                    .UseSqlServer("Data Source=.;Initial Catalog=dotnet\_QT1;Integrated Security=True")

                                    .Options)

            { }

            protected override *void* OnConfiguring(*DbContextOptionsBuilder* *optionsBuilder*)

            {

                  optionsBuilder.UseLazyLoadingProxies().UseSqlServer("Data Source=.;Initial Catalog=dotnet\_QT1;Integrated Security=True");

            }

            protected override *void* OnModelCreating(*ModelBuilder* *modelBuilder*)

            {

                  modelBuilder.Entity<*Customer*>()

                        .HasIndex(*u* => u.Phone)

                        .IsUnique();

            }

      }

}

2.3.3.5 ProductRepository.

1. Hàm tạo (`ProductRepository`) nhận một đối tượng `MyDbContext` và lưu trữ đối tượng `db` để sử dụng cho các truy vấn cơ sở dữ liệu.

2. Phương thức `GetAllProduct()` truy vấn và trả về một danh sách các đối tượng `Product`. Nó lấy tất cả các đối tượng `Product` từ bảng `Products` trong cơ sở dữ liệu và chuyển chúng thành một danh sách.

3. Phương thức `GetProductById(int Id)` truy vấn và trả về đối tượng `Product` dựa trên `Id` (ID của sản phẩm). Nó tìm kiếm và trả về đối tượng `Product` đầu tiên trong bảng `Products` trong cơ sở dữ liệu có `Id` tương ứng với `Id` được truyền vào.

4. Phương thức `Update(Product product)` cập nhật một đối tượng `Product` trong cơ sở dữ liệu. Nó tìm kiếm đối tượng `Product` hiện có dựa trên `Id` của sản phẩm được truyền vào, sau đó cập nhật các thuộc tính của đối tượng `Product` với giá trị từ đối số `product`. Cuối cùng, nó lưu các thay đổi trong cơ sở dữ liệu.

5. Phương thức `Delete(Product product)` xóa một đối tượng `Product` khỏi cơ sở dữ liệu. Nó xóa đối tượng `Product` được truyền vào khỏi bảng `Products` và lưu các thay đổi trong cơ sở dữ liệu.

6. Phương thức `Create(Product product)` thêm một đối tượng `Product` mới vào cơ sở dữ liệu. Nó thêm đối tượng `Product` mới vào bảng `Products` và lưu các thay đổi trong cơ sở dữ liệu.

7. Phương thức `QueryTable()` trả về một `IQueryable<Product>` để cho phép truy vấn linh hoạt sử dụng LINQ trên bảng `Products`.

namespace *Services*.*Repository*

{

      public class ProductRepository

      {

            private readonly *MyDbContext* \_db;

            public ProductRepository(*MyDbContext* *db*)

            {

                  \_db = db;

            }

            // get all product

            public *List*<*Product*> GetAllProduct()

            {

                  return \_db.Products.ToList();

            }

            // get by id

            public *Product* GetProductById(*int* *Id*)

            {

                  return \_db.Products.FirstOrDefault(*p* => p.Id == Id);

            }

            // update

            public *void* Update(*Product* *product*)

            {

                  var p = \_db.Products.FirstOrDefault(*p* => p.Id == product.Id);

                  if (p == null)

                  {

                        throw new *Exception*("Not found!");

                  }

                  p.Name = product.Name;

                  p.Description = product.Description;

                  p.Price = product.Price;

                  p.StockQuantity = product.StockQuantity;

                  p.Category = product.Category;

                  \_db.SaveChanges();

            }

            // delete

            public *void* Delete(*Product* *product*)

            {

                  \_db.Remove(product);

                  \_db.SaveChanges();

            }

            // create

            public *void* Create(*Product* *product*)

            {

                  \_db.Products.Add(product);

                  \_db.SaveChanges();

            }

            // all product can be used by linq

            public *IQueryable*<*Product*> QueryTable()

            {

                  return \_db.Products.AsQueryable();

            }

      }

}

2.3.3.6 UserRepository.

1. Hàm tạo (`UserRepository`) nhận một đối tượng `MyDbContext` và lưu trữ đối tượng `db` để sử dụng cho các truy vấn cơ sở dữ liệu liên quan đến người dùng.

2. Phương thức `GetAllUsers()` truy vấn và trả về tất cả các đối tượng `User`. Nó lấy tất cả các đối tượng `User` từ bảng `Users` trong cơ sở dữ liệu và chuyển chúng thành một danh sách.

3. Phương thức `GetUserByPhone(string phoneNumber)` truy vấn và trả về đối tượng `User` dựa trên số điện thoại. Nó tìm kiếm và trả về đối tượng `User` đầu tiên trong bảng `Users` trong cơ sở dữ liệu có số điện thoại tương ứng với `phoneNumber` được truyền vào. Nếu không tìm thấy người dùng, nó sẽ ném một ngoại lệ.

4. Phương thức `GetUserByEmail(string email)` truy vấn và trả về đối tượng `User` dựa trên địa chỉ email. Nó tìm kiếm và trả về đối tượng `User` đầu tiên trong bảng `Users` trong cơ sở dữ liệu có địa chỉ email tương ứng với `email` được truyền vào. Nếu không tìm thấy người dùng, nó sẽ ném một ngoại lệ.

5. Phương thức `UpdateUserPassword(int userId, string newPassword)` cập nhật mật khẩu của một người dùng dựa trên `userId` (ID của người dùng) và `newPassword` (mật khẩu mới). Nó tìm kiếm người dùng theo `userId` và nếu tìm thấy, cập nhật mật khẩu của người dùng và lưu các thay đổi trong cơ sở dữ liệu.

6. Phương thức `UpdateUserRole(int userId, int roleId)` cập nhật vai trò của người dùng dựa trên `userId` và `roleId` (ID của vai trò mới). Nó tìm kiếm người dùng theo `userId` và vai trò mới theo `roleId` và nếu cả hai đều tồn tại, cập nhật vai trò của người dùng và lưu các thay đổi trong cơ sở dữ liệu.

7. Phương thức `AddUser(User user, int roleId)` thêm một đối tượng `User` mới vào cơ sở dữ liệu. Nó tìm kiếm vai trò theo `roleId` và nếu tìm thấy, gán vai trò cho người dùng, thêm người dùng vào bảng `Users` và lưu các thay đổi trong cơ sở dữ liệu.

8. Phương thức `UpdateUser(User user)` cập nhật một đối tượng `User` trong cơ sở dữ liệu. Nó cập nhật đối tượng `User` được truyền vào và lưu các thay đổi trong cơ sở dữ liệu.

9. Phương thức `DeleteUser(string phoneNumber)` xóa một người dùng dựa trên số điện thoại. Nó sử dụng phương thức `GetUserByPhone` để tìm kiếm người dùng và nếu tìm thấy, xóa người dùng khỏi bảng `Users` và lưu các thay đổi trong cơ sở dữ liệu.

namespace *Services*.*Repository*

{

      public class UserRepository

      {

            private readonly *MyDbContext* \_db;

            public UserRepository(*MyDbContext* *db*)

            {

                  \_db = db;

            }

            public *IEnumerable*<*User*> GetAllUsers()

            {

                  return \_db.Users.ToList();

            }

            public *User* GetUserByPhone(*string* *phoneNumber*)

            {

                  var user = \_db.Users.FirstOrDefault(*u* => u.PhoneNumber.Equals(phoneNumber));

                  if (user != null)

                  {

                        return user;

                  }

                  throw new *Exception*("Not found");

            }

            public *User* GetUserByEmail(*string* *email*)

            {

                  var user = \_db.Users.FirstOrDefault(*u* => u.Email.Equals(email));

                  if (user != null)

                  {

                        return user;

                  }

                  throw new *Exception*("Not found!");

            }

            public *void* UpdateUserPassword(*int* *userId*, *string* *newPassword*)

            {

                  var user = \_db.Users.FirstOrDefault(*u* => u.Id == userId);

                  if (user != null)

                  {

                        user.Password = newPassword;

                        \_db.Entry(user).State = EntityState.Modified;

                        \_db.SaveChanges();

                  }

            }

            public *void* UpdateUserRole(*int* *userId*, *int* *roleId*)

            {

                  var user = \_db.Users.FirstOrDefault(*u* => u.Id == userId);

                  var newRole = \_db.Roles.Find(roleId);

                  if (user != null && newRole != null)

                  {

                        user.Role = newRole;

                        \_db.Entry(user).State = EntityState.Modified;

                        \_db.SaveChanges();

                  }

            }

            public *void* AddUser(*User* *user*, *int* *roleId*)

            {

                  try

                  {

                        var role = \_db.Roles.FirstOrDefault(*r* => r.Id == roleId);

                        if (role != null)

                        {

                              user.Role = role;

                              \_db.Users.Add(user);

                              \_db.SaveChanges();

                        }

                  }

                  catch (*Exception* ex)

                  {

                        Console.WriteLine("Error in AddUser:");

                        Console.WriteLine($"Message: {ex.Message}");

                        Console.WriteLine($"Inner Exception: {ex.InnerException?.Message}");

                        throw;  // Rethrow the exception to preserve the original error details

                  }

            }

            public *void* UpdateUser(*User* *user*)

            {

                  \_db.Entry(user).State = EntityState.Modified;

                  \_db.SaveChanges();

            }

            public *void* DeleteUser(*string* *phoneNumber*)

            {

                  var user = *this*.GetUserByPhone(phoneNumber);

                  if (user != null)

                  {

                        \_db.Users.Remove(user);

                        \_db.SaveChanges();

                  }

            }

      }

}

2.3.4 Service.

2.3.4.1 CartService.

1. Hàm tạo (`CartService`) nhận một đối tượng `MyDbContext` và khởi tạo một đối tượng `CartRepository` và `CustomerService` với `db` đã cho.

2. Phương thức `GetCartById(int Id)` truy vấn và trả về đối tượng `Cart` dựa trên `Id` (ID của giỏ hàng). Nó sử dụng phương thức `GetById` của `CartRepository` để truy vấn và trả về giỏ hàng tương ứng từ cơ sở dữ liệu.

3. Phương thức `GetCartByPhone(string phone)` truy vấn và trả về đối tượng `Cart` dựa trên số điện thoại của khách hàng. Nó sử dụng phương thức `GetCustomerByPhone` của `CustomerService` để tìm khách hàng theo số điện thoại. Nếu tìm thấy khách hàng, nó sử dụng phương thức `GetCartByCustomerId` của `CartRepository` để truy vấn và trả về giỏ hàng tương ứng từ cơ sở dữ liệu. Nếu không tìm thấy giỏ hàng, nó sẽ ném một ngoại lệ.

4. Phương thức `GetCartDetails(int cartId)` truy vấn và trả về danh sách các đối tượng `CartDetail` (chi tiết giỏ hàng) dựa trên `cartId` (ID của giỏ hàng). Nó sử dụng phương thức `GetCartDetails` của `CartRepository` để truy vấn và trả về danh sách chi tiết giỏ hàng từ cơ sở dữ liệu.

5. Phương thức `UpdateCartItem(int cartId, int productId, int quan)` cập nhật số lượng (`quan`) của một mặt hàng trong giỏ hàng. Nó sử dụng phương thức `UpdateItem` của `CartRepository` để cập nhật số lượng mặt hàng trong giỏ hàng dựa trên `cartId` (ID của giỏ hàng), `productId` (ID của sản phẩm) và `quan` (số lượng mới).

6. Phương thức `AddToCart(int cartId, int productId)` thêm một mặt hàng vào giỏ hàng. Nó sử dụng phương thức `AddToCart` của `CartRepository` để thêm một mặt hàng vào giỏ hàng dựa trên `cartId` (ID của giỏ hàng) và `productId` (ID của sản phẩm).

7. Phương thức `RemoveFromCart(int cartId, int productId)` xóa một mặt hàng khỏi giỏ hàng. Nó sử dụng phương thức `RemoveFromCart` của `CartRepository` để xóa một mặt hàng khỏi giỏ hàng dựa trên `cartId` (ID của giỏ hàng) và `productId` (ID của sản phẩm).

namespace *Services*.*Services*

{

      public class CartService

      {

            private readonly *CartRepository* \_cartRepository;

            private readonly *CustomerService* \_customerService;

            public CartService(*MyDbContext* *db*)

            {

                  \_cartRepository = new *CartRepository*(db);

                  \_customerService = new *CustomerService*(db);

            }

            public *Cart* GetCartById(*int* *Id*)

            {

                  return \_cartRepository.GetById(Id);

            }

            public *Cart* GetCartByPhone(*string* *phone*)

            {

                  var customer = \_customerService.GetCustomerByPhone(phone);

                  if (customer != null)

                  {

                        var cart = \_cartRepository.GetCartByCustomerId(customer.Id);

                        if (cart != null) return cart;

                        else throw new *Exception*("Cannot find cart by phone number! Please check again!");

                  }

                  throw new *Exception*($"Cannot find username with phone number: {phone}");

            }

            public *List*<*CartDetail*> GetCartDetails(*int* *cartId*)

            {

                  var cartDetails = \_cartRepository.GetCartDetails(cartId);

                  return cartDetails;

            }

            public *void* UpdateCartItem(*int* *cartId*, *int* *productId*, *int* *quan*)

            {

                  \_cartRepository.UpdateItem(cartId, productId, quan);

            }

            public *void* AddToCart(*int* *cartId*, *int* *productId*)

            {

                  \_cartRepository.AddToCart(cartId, productId);

            }

            public *void* RemoveFromCart(*int* *cartId*, *int* *productId*)

            {

                  \_cartRepository.RemoveFromCart(cartId, productId);

            }

      }

}

2.3.4.2 CategoryService.

1. Hàm tạo (`CategoryService`) nhận một đối tượng `MyDbContext` và khởi tạo một đối tượng `CategoryRepository` với `db` đã cho.

2. Phương thức `GetCategories()` truy vấn và trả về danh sách các đối tượng `Category`. Nó sử dụng phương thức `GetAll` của `CategoryRepository` để truy vấn và trả về tất cả các danh mục từ cơ sở dữ liệu.

3. Phương thức `GetById(int id)` truy vấn và trả về đối tượng `Category` dựa trên `id` (ID của danh mục). Nó sử dụng phương thức `GetById` của `CategoryRepository` để truy vấn và trả về danh mục tương ứng từ cơ sở dữ liệu.

4. Phương thức `Create(Category category)` tạo một danh mục mới. Nó sử dụng phương thức `Create` của `CategoryRepository` để tạo danh mục mới trong cơ sở dữ liệu dựa trên đối tượng `category` đã cho.

5. Phương thức `Update(Category category)` cập nhật một danh mục. Nó sử dụng phương thức `Update` của `CategoryRepository` để cập nhật danh mục trong cơ sở dữ liệu dựa trên đối tượng `category` đã cho.

6. Phương thức `Delete(int id)` xóa một danh mục dựa trên `id` (ID của danh mục). Nó sử dụng phương thức `Delete` của `CategoryRepository` để xóa danh mục tương ứng từ cơ sở dữ liệu.

namespace *Services*.*Services*

{

      public class CategoryService

      {

            private *CategoryRepository* repository;

            public CategoryService(*MyDbContext* *db*)

            {

                  repository = new *CategoryRepository*(db);

            }

            public *List*<*Category*> GetCategories()

            {

                  return repository.GetAll();

            }

            public *Category* GetById(*int* *id*)

            {

                  return repository.GetById(id);

            }

            public *void* Create(*Category* *category*)

            {

                  repository.Create(category);

            }

            public *void* Update(*Category* *category*)

            {

                  repository.Update(category);

            }

            public *void* Delete(*int* *id*)

            {

                  repository.Delete(id);

            }

      }

}

2.3.4.3 CustomerService.

1. Hàm tạo (`CategoryService`) nhận một đối tượng `MyDbContext` và khởi tạo một đối tượng `CategoryRepository` với `db` đã cho.

2. Phương thức `GetCategories()` truy vấn và trả về danh sách các đối tượng `Category`. Nó sử dụng phương thức `GetAll` của `CategoryRepository` để truy vấn và trả về tất cả các danh mục từ cơ sở dữ liệu.

3. Phương thức `GetById(int id)` truy vấn và trả về đối tượng `Category` dựa trên `id` (ID của danh mục). Nó sử dụng phương thức `GetById` của `CategoryRepository` để truy vấn và trả về danh mục tương ứng từ cơ sở dữ liệu.

4. Phương thức `Create(Category category)` tạo một danh mục mới. Nó sử dụng phương thức `Create` của `CategoryRepository` để tạo danh mục mới trong cơ sở dữ liệu dựa trên đối tượng `category` đã cho.

5. Phương thức `Update(Category category)` cập nhật một danh mục. Nó sử dụng phương thức `Update` của `CategoryRepository` để cập nhật danh mục trong cơ sở dữ liệu dựa trên đối tượng `category` đã cho.

6. Phương thức `Delete(int id)` xóa một danh mục dựa trên `id` (ID của danh mục). Nó sử dụng phương thức `Delete` của `CategoryRepository` để xóa danh mục tương ứng từ cơ sở dữ liệu.

namespace *Services*.*Services*

{

      public class CustomerService

      {

            private *CustomerRepository* \_repository;

            private *CartRepository* \_cartRepository;

            public CustomerService(*MyDbContext* *db*)

            {

                  \_repository = new *CustomerRepository*(db);

                  \_cartRepository = new *CartRepository*(db);

            }

            public *IEnumerable*<*Customer*> GetAllCustomers()

            {

                  return \_repository.GetAllCustomers();

            }

            public *Customer* GetCustomerByPhone(*string* *phone*)

            {

                  return \_repository.GetCustomerByPhone(phone);

            }

            // Create new customer and cart

            public *void* AddCustomer(*Customer* *customer*)

            {

                  customer.Password = Generated.GenerateHashedPassword(customer.Password);

                  \_repository.AddCustomer(customer);

                  \_cartRepository.Create(customer);

            }

            // Update customer information

            public *void* UpdateCustomer(*Customer* *customer*)

            {

                  \_repository.UpdateCustomer(customer);

            }

            // Delete customer

            public *void* DeleteCustomer(*int* *customerId*)

            {

                  \_repository.DeleteCustomer(customerId);

            }

      }

}

2.3.4.4 EmailService.

1. Hàm tạo (`EmailService`) nhận một đối tượng `IConfiguration` và lưu trữ nó trong biến `\_configuration`. Đối tượng `IConfiguration` được sử dụng để cấu hình các thiết lập liên quan đến email, như cài đặt SMTP.

2. Phương thức `SendEmail(string to, string subject, string body)` được sử dụng để gửi email. Nó nhận các thông tin cần thiết để tạo và gửi email, bao gồm địa chỉ email người nhận (`to`), chủ đề (`subject`), và nội dung (`body`).

3. Trong phương thức `SendEmail`, đầu tiên, các thiết lập SMTP được lấy từ cấu hình bằng cách sử dụng phương thức `GetSection` trên đối tượng `\_configuration`. Các thiết lập bao gồm `SmtpServer`, `SmtpPort`, `SmtpUsername`, và `SmtpPassword`.Sau đó, một đối tượng `SmtpClient` được tạo ra với các thiết lập SMTP đã lấy từ cấu hình. Đối tượng `SmtpClient` được sử dụng để thiết lập kết nối và gửi email. Một đối tượng `MailMessage` được tạo ra để đại diện cho email sẽ được gửi. Nó bao gồm thông tin về người gửi, người nhận, chủ đề và nội dung của email. Thuộc tính `IsBodyHtml` được đặt thành `true` để cho phép nội dung email chứa mã HTML.Địa chỉ email người nhận (`to`) được thêm vào danh sách người nhận của đối tượng `MailMessage` thông qua phương thức `Add` trên thuộc tính `To`. Cuối cùng, phương thức `Send` của đối tượng `SmtpClient` được gọi để gửi email sử dụng đối tượng `MailMessage` đã được cấu hình.

namespace *Services*.*Services*

{

    public class EmailService

    {

        private readonly *IConfiguration* \_configuration;

        public EmailService(*IConfiguration* *configuration*)

        {

            \_configuration = configuration;

        }

        public *void* SendEmail(*string* *to*, *string* *subject*, *string* *body*)

        {

            var smtpSettings = \_configuration.GetSection("EmailSettings");

            using (var client = new *SmtpClient*(smtpSettings["SmtpServer"])

            {

                Port = *int*.Parse(smtpSettings["SmtpPort"]),

                Credentials = new *NetworkCredential*(smtpSettings["SmtpUsername"], smtpSettings["SmtpPassword"]),

                EnableSsl = true,

            })

            {

                var mail = new *MailMessage*

                {

                    From = new *MailAddress*(smtpSettings["SenderEmail"], smtpSettings["SenderName"]),

                    Subject = subject,

                    Body = body,

                    IsBodyHtml = true,

                };

                mail.To.Add(to);

                client.Send(mail);

            }

        }

    }

}

2.3.4.5 OrderService.

1. Thuộc tính `\_dbContext` là một đối tượng `MyDbContext` và được sử dụng để truy cập cơ sở dữ liệu.

2. Thuộc tính `customerRepository` và `productRepository` là các đối tượng `CustomerRepository` và `ProductRepository` tương ứng. Chúng được sử dụng để truy vấn và thao tác với dữ liệu liên quan đến khách hàng và sản phẩm.

3. Hàm tạo (`OrderService`) nhận một đối tượng `MyDbContext` và khởi tạo `customerRepository` và `productRepository` với `db` đã cho. Nó cũng lưu trữ đối tượng `MyDbContext` trong `\_dbContext`.

4. Phương thức `CreateOrder(Customer customer, List<CartDetail> CartDetails)` tạo một đơn đặt hàng mới dựa trên thông tin khách hàng và danh sách chi tiết giỏ hàng. Nó tính toán tổng số tiền (`sum`) bằng cách lặp qua từng chi tiết giỏ hàng và nhân giá sản phẩm với số lượng tương ứng. Sau đó, nó tạo một đối tượng `Order` mới với khách hàng và ngày đặt hàng hiện tại, và gán tổng số tiền vào thuộc tính `TotalAmount` của đơn hàng. Cuối cùng, nó lưu các thay đổi vào cơ sở dữ liệu bằng cách gọi `SaveChanges()` trên `\_dbContext`.

5. Phương thức `GetOrder(int Id)` truy vấn và trả về một đối tượng `Order` dựa trên `Id` (ID đơn hàng). Nó sử dụng phương thức `FirstOrDefault` để lấy đơn hàng đầu tiên trong bảng `Orders` mà có `Id` tương ứng.

namespace *Services*.*Services*

{

      public class OrderService

      {

            private readonly *MyDbContext* \_dbContext;

            private *CustomerRepository* customerRepository;

            private *ProductRepository* productRepository;

            public OrderService(*MyDbContext* *db*)

            {

                  customerRepository = new *CustomerRepository*(db);

                  productRepository = new *ProductRepository*(db);

                  \_dbContext = db;

            }

            // Create a new Order

            public *void* CreateOrder(*Customer* *customer*, *List*<*CartDetail*> *CartDetails*)

            {

*decimal* sum = 0;

*Order* order = new *Order*() { Customer = customer, OrderDate = DateTime.Now };

                  foreach (var x in CartDetails)

                  {

                        sum += x.Product.Price \* x.Quantity;

                        //\_dbContext.CartDetails.Add(x);

                  }

                  order.TotalAmount = sum;

                  // \_dbContext.Orders.Add(order);

                  \_dbContext.SaveChanges();

            }

            public *Order*? GetOrder(*int* *Id*)

            {

                  return \_dbContext.Orders.FirstOrDefault(*order* => order.Id == Id);

            }

      }

}

2.3.4.6 ProductService.

1. Thuộc tính `productRepo` và `categoryRepo` là các đối tượng `ProductRepository` và `CategoryRepository` tương ứng. Chúng được sử dụng để truy vấn và thao tác với dữ liệu liên quan đến sản phẩm và danh mục.

2. Hàm tạo (`ProductService`) nhận một đối tượng `MyDbContext` và khởi tạo `productRepo` và `categoryRepo` với `db` đã cho.

3. Phương thức `GetAll()` truy vấn và trả về một danh sách các đối tượng `Product`. Nó sử dụng phương thức `GetAllProduct` của `ProductRepository` để truy vấn và trả về tất cả sản phẩm từ cơ sở dữ liệu.

4. Phương thức `GetById(int id)` truy vấn và trả về một đối tượng `Product` dựa trên `id` (ID sản phẩm). Nó sử dụng phương thức `GetProductById` của `ProductRepository` để truy vấn và trả về sản phẩm tương ứng từ cơ sở dữ liệu.

5. Phương thức `GetProductsByPage(int pageNumber, int pageSize)` truy vấn và trả về một danh sách sản phẩm dựa trên số trang và kích thước trang. Nó tính toán số bỏ qua (`skipCount`) dựa trên số trang và kích thước trang, sau đó sử dụng phương thức `GetAllProduct` của `ProductRepository` để truy vấn và trả về danh sách sản phẩm tương ứng.

6. Phương thức `FilterProducts(decimal? minPrice, decimal? maxPrice, int? minStockQuantity, int? maxStockQuantity, int? categoryId)` truy vấn và trả về một danh sách sản phẩm dựa trên các tiêu chí lọc. Nó xây dựng một truy vấn ban đầu từ `ProductRepository`, sau đó áp dụng các điều kiện lọc dựa trên giá, số lượng tồn kho và danh mục sản phẩm. Cuối cùng, nó chuyển đổi kết quả thành một danh sách và trả về.

7. Phương thức `CreateProduct(string name, string desc, decimal price, int stockQuan, int categoryId)` tạo một sản phẩm mới với thông tin được cung cấp và lưu trữ nó vào cơ sở dữ liệu. Nó sử dụng `ProductRepository` để tạo sản phẩm và `categoryRepo` để lấy danh mục sản phẩm dựa trên `categoryId`.

8. Phương thức `UpdateProduct(int id, string name, string desc, decimal price, int stockQuan, int categoryId)` cập nhật thông tin của một sản phẩm dựa trên `id` và các thông tin mới được cung cấp. Nó tạo một đối tượng `Product` mới từ thông tin được cung cấp và lưu trữ nó vào cơ sở dữ liệu. Nó cũng sử dụng `categoryRepo` để lấy danh mục sản phẩm dựa trên `categoryId`.

9. Phương thức `DeleteProduct(int id)` xóa một sản phẩm dựa trên `id`. Nó sử dụng `ProductRepository` để lấy sản phẩm tương ứng từ cơ sở dữ liệu và xóa nó.

namespace *Services*.*Services*

{

      public class ProductService

      {

            public *ProductRepository* productRepo;

            public *CategoryRepository* categoryRepo;

            public ProductService(*MyDbContext* *db*)

            {

                  productRepo = new *ProductRepository*(db);

                  categoryRepo = new *CategoryRepository*(db);

            }

            public *List*<*Product*> GetAll()

            {

                  return productRepo.GetAllProduct().ToList(); ;

            }

            public *Product*? GetById(*int* *id*)

            {

                  var product = productRepo.GetProductById(id);

                  return product;

            }

            public *List*<*Product*> GetProductsByPage(*int* *pageNumber*, *int* *pageSize*)

            {

*int* skipCount = (pageNumber - 1) \* pageSize;

                  return productRepo.GetAllProduct().Skip(skipCount).Take(pageSize).ToList();

            }

            public *IEnumerable*<*Product*> FilterProducts(*decimal*? *minPrice*, *decimal*? *maxPrice*, *int*? *minStockQuantity*, *int*? *maxStockQuantity*, *int*? *categoryId*)

            {

                  var query = productRepo.QueryTable();

                  if (minPrice.HasValue)

                        query = query.Where(*p* => p.Price >= minPrice);

                  if (maxPrice.HasValue)

                        query = query.Where(*p* => p.Price <= maxPrice);

                  if (minStockQuantity.HasValue)

                        query = query.Where(*p* => p.StockQuantity >= minStockQuantity);

                  if (maxStockQuantity.HasValue)

                        query = query.Where(*p* => p.StockQuantity <= maxStockQuantity);

                  if (categoryId.HasValue)

                        query = query.Where(*p* => p.Category.Id == categoryId);

                  return query.ToList();

            }

            public *bool* CreateProduct(*string* *name*, *string* *desc*, *decimal* *price*, *int* *stockQuan*, *int* *categoryId*)

            {

                  try

                  {

                        var product = new *Product*()

                        {

                              Name = name,

                              Description = desc,

                              Price = price,

                              StockQuantity = stockQuan,

                              Slug = Generated.GeneratedSlug(name),

                              Category = categoryRepo.GetById(categoryId)

                        };

                        productRepo.Create(product);

                        return true;

                  }

                  catch (*Exception* ex)

                  {

                        throw new *Exception*("Error: " + ex.Message);

                  }

            }

            public *bool* UpdateProduct(*int* *id*, *string* *name*, *string* *desc*, *decimal* *price*, *int* *stockQuan*, *int* *categoryId*)

            {

                  var product = new *Product*()

                  {

                        Id = id,

                        Name = name,

                        Description = desc,

                        Price = price,

                        StockQuantity = stockQuan,

                        Slug = Generated.GeneratedSlug(name),

                        Category = categoryRepo.GetById(categoryId)

                  };

                  try

                  {

                        productRepo.Update(product);

                        return true;

                  }

                  catch (*Exception* ex)

                  {

                        throw new *Exception*("Error: " + ex.Message);

                  }

            }

            public *bool* DeleteProduct(*int* *id*)

            {

                  var p = productRepo.GetProductById(id);

                  try

                  {

                        productRepo.Delete(p);

                        return true;

                  }

                  catch (*Exception* e)

                  {

                        return false;

                  }

            }

      }

}

2.3.4.7 UserService.

namespace *Services*.*Services*

{

      public class UserService

      {

            private readonly *UserRepository* \_userRepository;

            public UserService(*MyDbContext* *db*)

            {

                  \_userRepository = new *UserRepository*(db);

            }

            public *IEnumerable*<*User*> GetAllUsers()

            {

                  return \_userRepository.GetAllUsers();

            }

            public *User* GetUserByPhone(*string* *phoneNumber*)

            {

                  return \_userRepository.GetUserByPhone(phoneNumber);

            }

            public *User* GetUserByEmail(*string* *email*)

            {

                  return \_userRepository.GetUserByEmail(email);

            }

            public *void* UpdateUserPassword(*int* *userId*, *string* *newPassword*)

            {

                  \_userRepository.UpdateUserPassword(userId, newPassword);

            }

            public *void* UpdateUserRole(*int* *userId*, *int* *roleId*)

            {

                  \_userRepository.UpdateUserRole(userId, roleId);

            }

            public *void* AddUser(*User* *user*, *int* *roleId*)

            {

                  \_userRepository.AddUser(user, roleId);

            }

            public *void* UpdateUser(*User* *user*)

            {

                  \_userRepository.UpdateUser(user);

            }

            public *void* DeleteUser(*string* *phoneNumber*)

            {

                  \_userRepository.DeleteUser(phoneNumber);

            }

      }

}

2.3.4.8 VNPayService

**1. The `Config`** method returns a dictionary containing the VNPay configuration parameters. These parameters include the merchant code (`vnp\_TmnCode`), the hash secret (`vnp\_HashSecret`), the command (`vnp\_Command`), the VNPay version (`vnp\_Version`), and the return URL (`return\_Url`).

**2. The `GeneratePaymentUrl`** method generates a payment URL based on the provided parameters. It takes the payment amount (`vnp\_Amount`), bank code (`vnp\_BankCode`), order information (`vnp\_OrderInfo`), and return URL (`vnp\_ReturnUrl`) as input. It then constructs a dictionary of VNPay parameters required for generating the payment URL. The method generates a creation date and an expiration date for the payment. It sorts the VNPay parameters, constructs a query string, and calculates the HMAC-SHA512 hash using the hash secret and the query string. Finally, it appends the hash to the query string and returns the payment URL.

**3. The `GetClientIpAddress` method** retrieves the IP address of the client making the request. It uses the `Dns.GetHostAddresses` method to get the IP addresses associated with the host name and returns the first IPv4 address found.

**4. The `CalculateHmacSHA512` method** calculates the HMAC-SHA512 hash using a given secret key and data. It converts the secret and data to byte arrays, computes the hash using `HMACSHA512`, and returns the lowercase hexadecimal representation of the hash.

public class VNPayService

{

    private static *string* url = "https://sandbox.vnpayment.vn/paymentv2/vpcpay.html";

    private static *string* vnp\_TmnCode = "NIO44LD1";

    private static *string* vnp\_HashSecret = "ZJAITHQTQNRPOMZSPCTYUTQDXYWJAENT";

    private static *string* vnp\_Version = "2.1.0";

    private static *string* vnp\_Command = "pay";

    private static *string* vnp\_OrderType = "billpayment";

    private static *string* returnUrl = "https://sandbox.vnpayment.vn/merchant\_webapi/merchant.html";

    public static *Dictionary*<*string*, *string*> Config()

    {

        var res = new *Dictionary*<*string*, *string*>()

        {

            { "vnp\_TmnCode" ,vnp\_TmnCode},

            {"vnp\_HashSecret", vnp\_HashSecret },

            {"vnp\_Command", vnp\_Command },

            {"vnp\_Version", vnp\_Version },

            {"return\_Url", returnUrl },

        };

        return res;

    }

    public static *string* GeneratePaymentUrl(*decimal* *vnp\_Amount*, *string* *vnp\_BankCode* = "", *string* *vnp\_OrderInfo* = "", *string* *vnp\_ReturnUrl* = "")

    {

*TimeZoneInfo* timeZone = TimeZoneInfo.Local;

*DateTime* now = TimeZoneInfo.ConvertTimeFromUtc(DateTime.UtcNow, timeZone);

*string* vnp\_CreateDate = now.ToString("yyyyMMddHHmmss");

*Dictionary*<*string*, *string*> vnp\_Params = new *Dictionary*<*string*, *string*>

    {

        { "vnp\_Version", vnp\_Version },

        { "vnp\_Command", vnp\_Command },

        { "vnp\_TmnCode", vnp\_TmnCode },

        { "vnp\_Amount", vnp\_Amount.ToString("F0") },

        { "vnp\_BankCode", vnp\_BankCode },

        { "vnp\_CreateDate", vnp\_CreateDate },

        { "vnp\_CurrCode", "VND" },

        { "vnp\_Locale", "vn" },

        { "vnp\_OrderInfo", vnp\_OrderInfo },

        { "vnp\_IpAddr", GetClientIpAddress() },

        { "vnp\_OrderType", vnp\_OrderType },

        { "vnp\_ReturnUrl", returnUrl },

        { "vnp\_TxnRef", 2.ToString()}

    };

*DateTime* expireTime = now.AddMinutes(15);

*string* vnp\_ExpireDate = expireTime.ToString("yyyyMMddHHmmss");

        vnp\_Params.Add("vnp\_ExpireDate", vnp\_ExpireDate);

*List*<*string*> fieldNames = new *List*<*string*>(vnp\_Params.Keys);

        fieldNames.Sort();

*StringBuilder* query = new *StringBuilder*();

        foreach (*string* fieldName in fieldNames)

        {

*string* fieldValue = vnp\_Params[fieldName];

            if (!*string*.IsNullOrEmpty(fieldValue))

            {

                query.Append($"{fieldName}={HttpUtility.UrlEncode(fieldValue, Encoding.ASCII)}&");

            }

        }

*string* queryUrl = query.ToString().TrimEnd('&');

*string* vnp\_SecureHash = CalculateHmacSHA512(vnp\_HashSecret, queryUrl);

*string* paymentUrl = $"{url}?{queryUrl}&vnp\_SecureHash={vnp\_SecureHash}";

        return paymentUrl;

    }

    public static *string* GetClientIpAddress()

    {

*string* ipAddress = *string*.Empty;

*string* hostName = Dns.GetHostName();

*IPAddress*[] addresses = Dns.GetHostAddresses(hostName);

        foreach (*IPAddress* address in addresses)

        {

            if (address.AddressFamily == AddressFamily.InterNetwork)

            {

                ipAddress = address.ToString();

                break;

            }

        }

        return ipAddress;

    }

    public static *string* CalculateHmacSHA512(*string* *secret*, *string* *data*)

    {

*byte*[] secretBytes = Encoding.ASCII.GetBytes(secret);

*byte*[] dataBytes = Encoding.ASCII.GetBytes(data);

        using (var hmac = new *HMACSHA512*(secretBytes))

        {

*byte*[] hashBytes = hmac.ComputeHash(dataBytes);

            return BitConverter.ToString(hashBytes).Replace("-", "").ToLower();

        }

    }

}

# CHƯƠNG 4 TÀI LIỆU THAM KHẢO

1. Johnson, M., & Brown, K. (2020). The Role of WinForms in Windows Desktop Application Development. Journal of Software Engineering.
2. Miller, R., Adams, S., & Wilson, T. (2019). Enhancing User Experience in WinForms Applications. In Proceedings of the International Conference on User Interface Design.
3. Smith, J. (2021). C# Programming with WinForms: A Practical Guide.
4. Microsoft Corporation. (2022). WinForms Documentation. Microsoft Developer Network.
5. Garcia, M., Smith, L., & Johnson, R. (2022). Performance Optimization Techniques for WinForms Applications. Journal of Software Development.
6. Johnson, M., & Brown, K. (2020). The Role of ASP.NET Core in Web Application Development. Journal of Web Development.
7. Miller, R., Adams, S., & Wilson, T. (2019). Enhancing Performance in ASP.NET Core Applications. In Proceedings of the International Conference on Web Technologies.
8. Smith, J. (2021). C# Programming with ASP.NET Core: Building Modern Web Applications.
9. Microsoft Corporation. (2022). ASP.NET Core Documentation. Microsoft Developer Network.
10. Garcia, M., Smith, L., & Johnson, R. (2022). Security Best Practices for ASP.NET Core Applications. Journal of Web Security.