# Report: Convenience Store Management System

## Introduction and Summary of Project

The "ConvenienceStore" project is a robust and user-centric application designed to cater to the operational needs of modern convenience stores. The primary objective of the project is to simplify, streamline, and enhance the day-to-day management tasks of a store, from inventory tracking to sales processing and customer relationship management. With a comprehensive suite of features and tools, the application addresses common challenges faced by store managers and employees, ensuring a seamless and efficient business operation. By integrating state-of-the-art technologies and design principles, the "ConvenienceStore" application offers a blend of functionality, aesthetics, and usability, making it an indispensable tool for contemporary retail environments.

## Development Approach

The development of the "ConvenienceStore" application followed a structured and methodological approach, ensuring optimal results and future scalability:

**Frameworks:**

.NET Core & WPF: The core of the application was built on the .NET Core framework, leveraging the capabilities of the Windows Presentation Foundation (WPF) for the user interface. This created a rich, interactive, and responsive desktop application.
Entity Framework Core: As the primary Object-Relational Mapping (ORM) solution, Entity Framework Core was adopted to facilitate database operations. This enabled a seamless interaction between the application and the underlying database, ensuring data consistency, integrity, and ease of querying.

**Libraries & Components:**

Emgu.CV: To integrate computer vision capabilities, the Emgu.CV library, a .NET wrapper for OpenCV, was employed. This could be used for features like barcode scanning or image recognition.
LiveCharts & LiveCharts.Wpf: For dynamic data visualisation and reporting, the LiveCharts library was integrated, allowing the application to display real-time sales, inventory, and other vital statistics.
Material Design: To enhance the user experience and aesthetics, Material Design libraries were employed, giving the application a modern and intuitive look.
FontAwesome: The application's icons and visual cues were powered by FontAwesome, ensuring a uniform and pleasing visual presentation.
FluentValidation: Data input validation was handled using the FluentValidation library, ensuring data accuracy and integrity throughout the application.

**External Tools:**

AForge.Video.DirectShow: For potential video capture and processing features, the AForge.Video.DirectShow library was incorporated.
ZXing.Net: To support barcode generation and scanning, the ZXing.Net library was utilised, aiding in quick product lookup and sales processing.
Throughout the development process, a strong emphasis was placed on user-centric design, performance optimisation, and modular architecture. By leveraging the aforementioned frameworks, libraries, and tools, the "ConvenienceStore" application is a testament to modern software engineering principles, ready to revolutionise the retail sector.

**Database Management: SQL Server Management Studio (SSMS)**

Our choice of DBMS was SQL Server Management Studio, given its reputation for stability and efficiency. SSMS, coupled with C# and the Entity Framework, presented a cohesive ecosystem for seamless data management. This trinity ensured our application's data integrity, responsiveness, and scalability.

**Project Structure:**

Model: This is where we store data models. We've categorised them into Admin, Staff, and three standalone classes: Account, CurrentAccount, and Customer.

- Admin: Model of an admin account with the highest permission on all actions within the system. Their role includes HR management classes (Manager, Member and Supplier with their InputInfo), product management classes (storage (Product, SmallConsignment,...) and payment (Bill)), and salary management (Salary)

- Staff: They have a smaller access pool compared to admin, but are still able to work on most of the functionality of the system, including all product management and product management classes.
- Three smaller classes outside are responsible for user profile access and modification. This includes Account - CurrentAccount (for displaying and retrieving user account information) and Customer, which is a role without any management classes, as their sole permission is to only be able to buy products from the site.

Resources: Resources required for the UI, such as Fonts, Icons, Images, ResourceXAML, Styles, and UserControlCustom, are housed here.

Utils: Utility functions and classes are stored here. Categories include Converters, DataLayerAccess, Helpers, and Validation. There are also two utility classes Constant.cs and RegexUtilities.cs.

- Converter: An image converter class that transforms an image into a binary file for easy transmission and vice versa.
- Data Access Layers (DALs): Including different types of data conversion methods, these classes are for the data formatting and transmission between the SQL server and the controller.
- Helpers: Include a lot of useful methods that are not necessarily categorisable, with different conversion methods, getters and setters, and some validation methods as well. DatabaseHelper is for admin users with higher permission, including HR database CRUD methods and data fetchers, whereas StaffDatabaseHelpers are for users with at least staff permission, with different product and customer database CRUD methods and data fetchers. The Helpers class is for miscellaneous functions that are useful to the system.
- Validators: Used to validate inputs received from product data inputs from staff or suppliers data input from suppliers and admins. Including ProductValidator class for products and SupplierValidator for suppliers
- 2 outside classes: Constant class is used to set values onto different constants throughout the login session, e.g., role or server status; RegexUtilities are for mail validation.

ViewModel: The MVVM pattern is implemented, and this is where you've placed your ViewModels. They're categorised into Admin, Login, StaffVM, and SubViewModels.

- Admin: Consists of everything needed to display and receive information from all admin-related interfaces
    - AdminVM (Admin view models): Include all the base models in which the admin has access to
    - Command: Handle all events that can occur during an admin login session

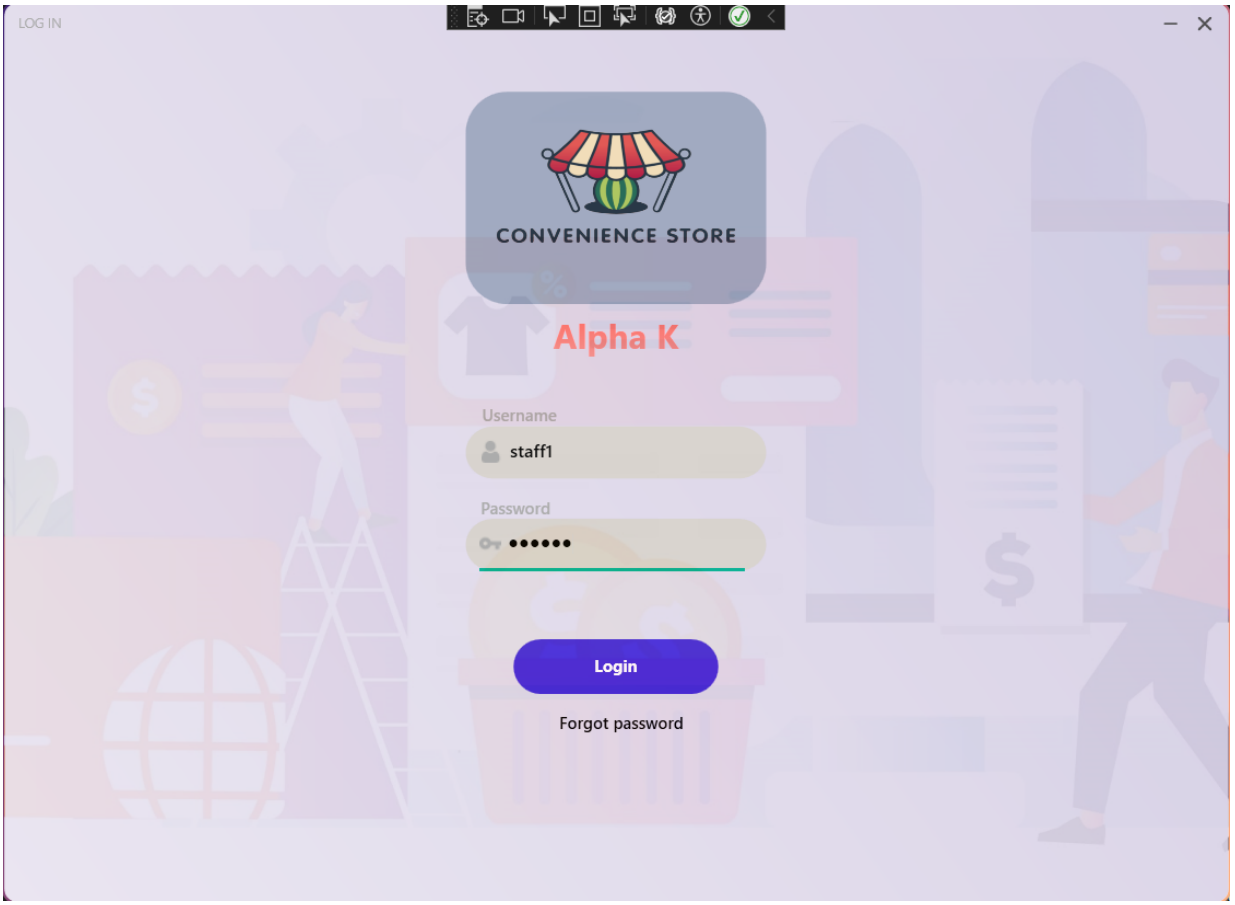
- Converter: Handle all conversions from different types of value (indicating a change in the interface) into visibility (allowing the user to view/modify something), followed by some unit converters.
- AdminMainViewModel: for the main menu after you log in as the admin
- BaseViewModel: Parent class to different functional views, with shared methods among all inherited classes.
- RelayCommand: a command permission checker and relayer.

Views: This folder contains the user interface (UI) XAML files. It includes Admin, Login, and Staff. Each folder includes all the code for generating different role's graphical user interfaces.

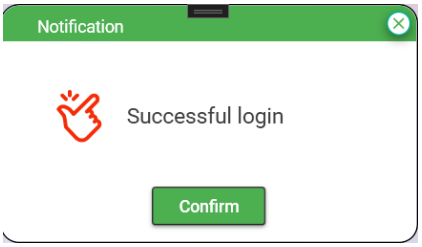
There are also a few standalone configuration and utility files such as .gitattributes, .gitignore, App.config, App.xaml, AssemblyInfo.cs, and script.sql.

**User Interface:**

It's a dashboard-style interface for a shop management system.

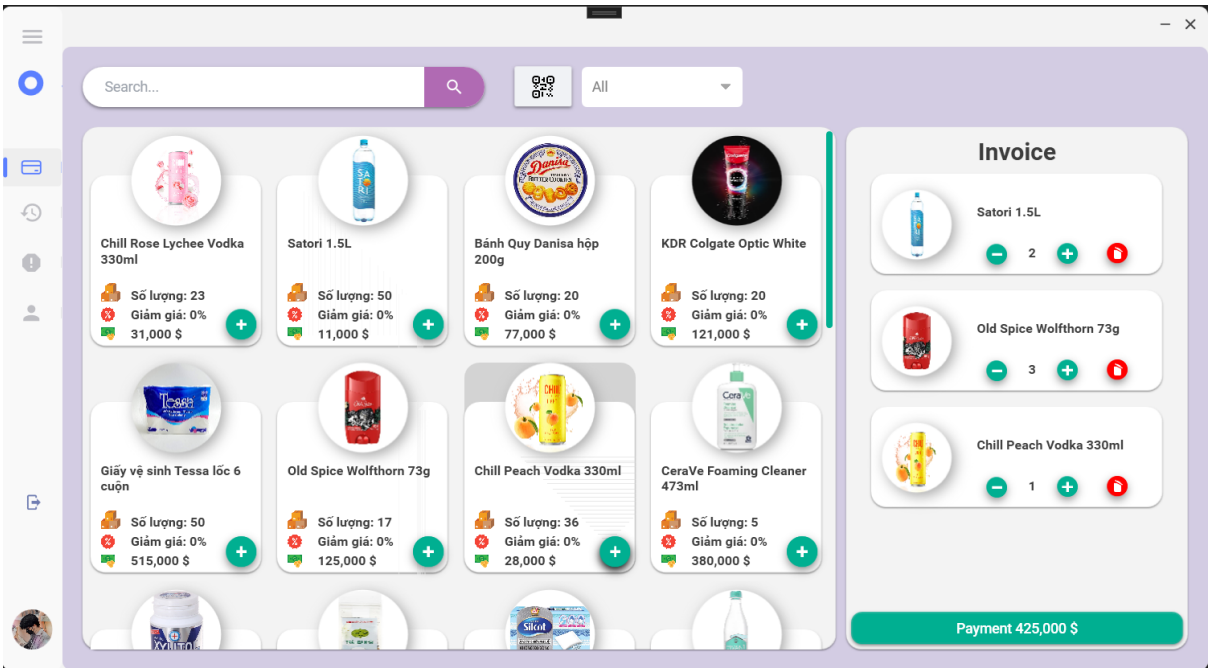



Login interface

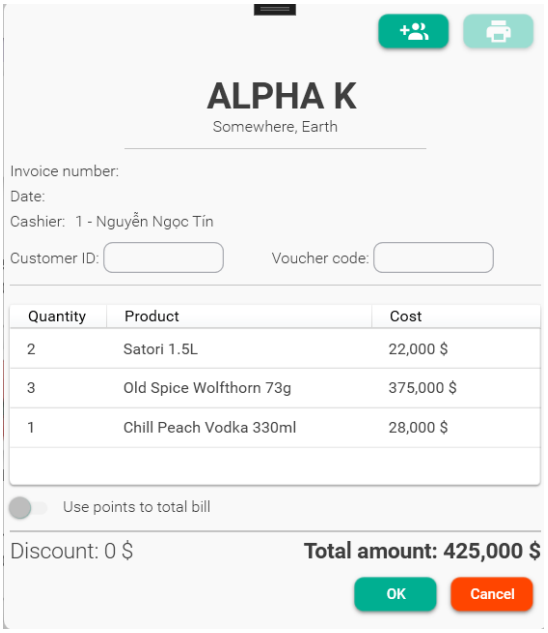



Login confirmed notification window (after valid credential was inputted)

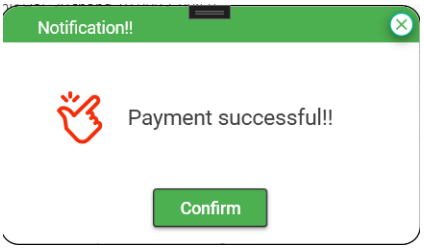
**Customer/Staff interfaces:**



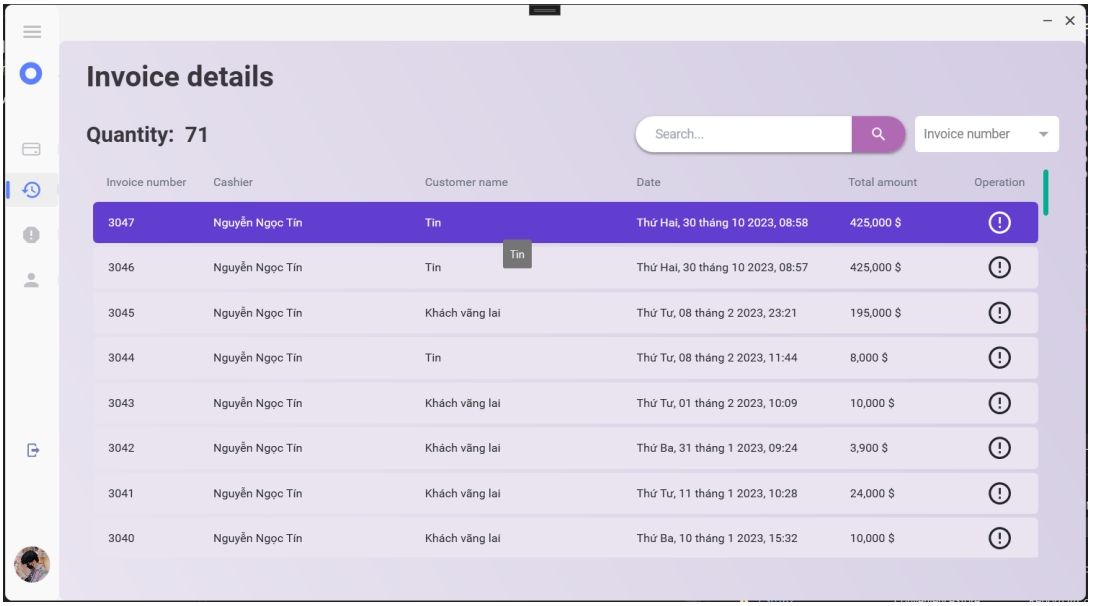

Selecting product & invoice (shopping cart) interface



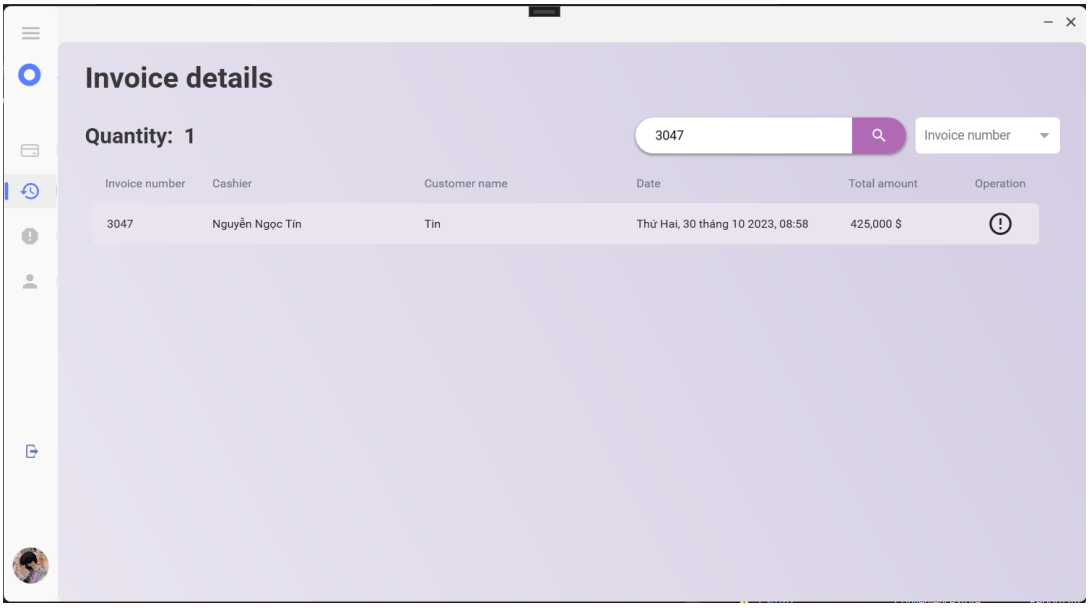

Invoice interface (After clicking the payment button)



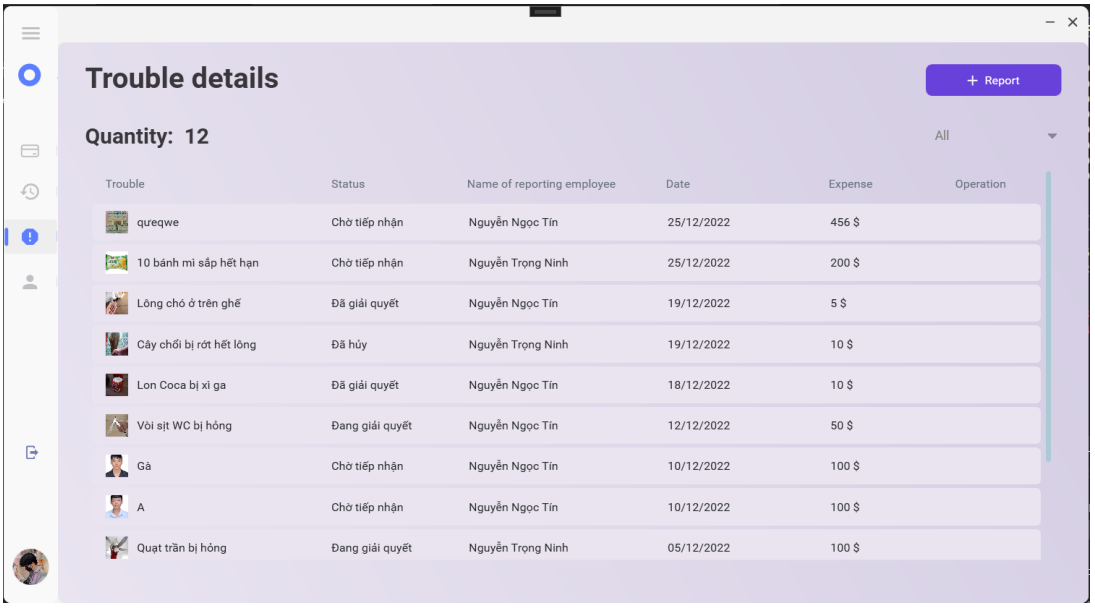

Confirmed payment notification window



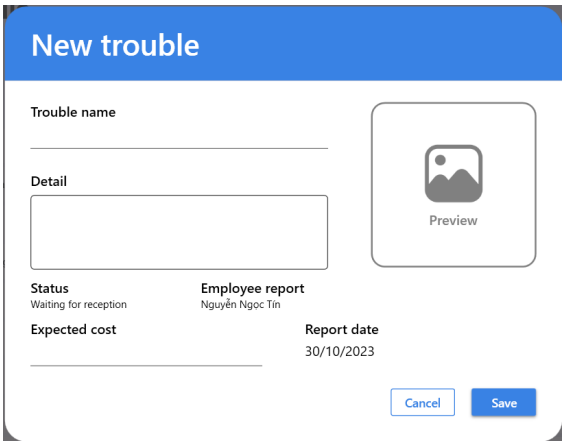

Invoice Interface



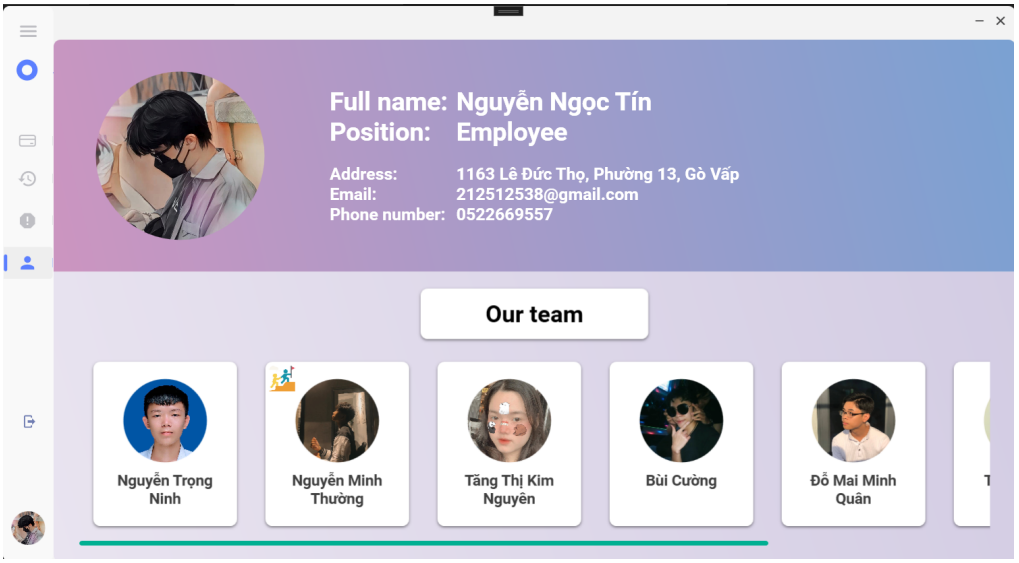

Invoice search using the invoice number



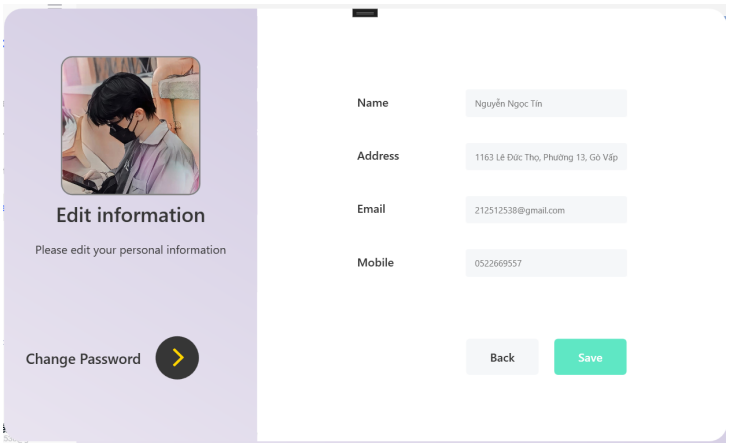

Trouble list interface




New trouble submission form

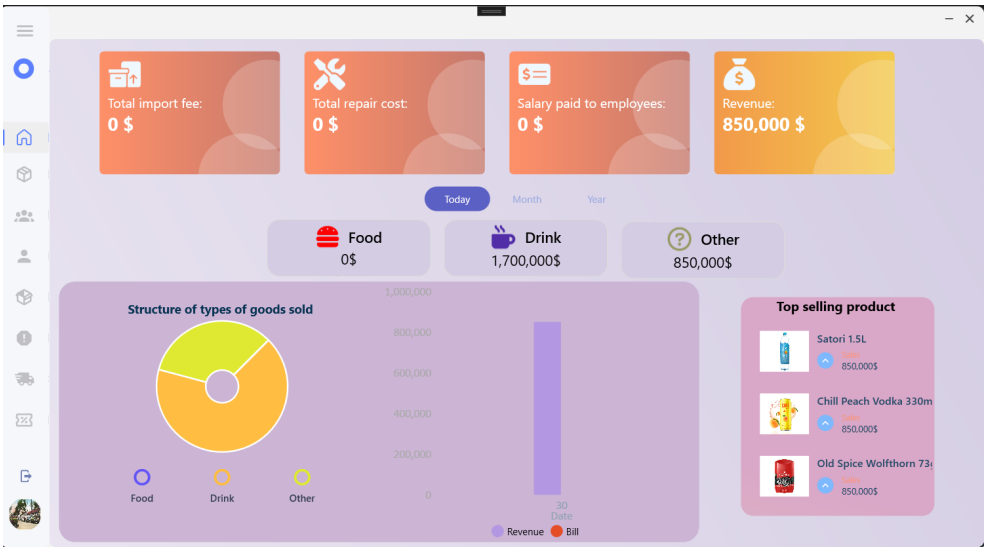



Profile Interface

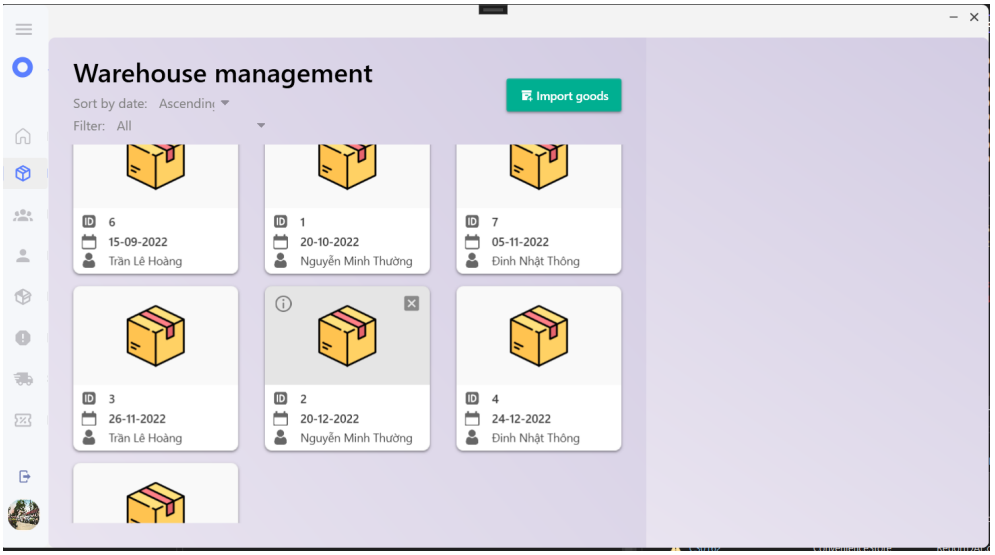



Edit Profile Information Interface

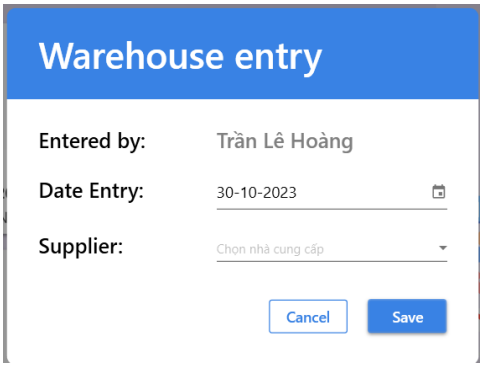
**Admin interfaces:**



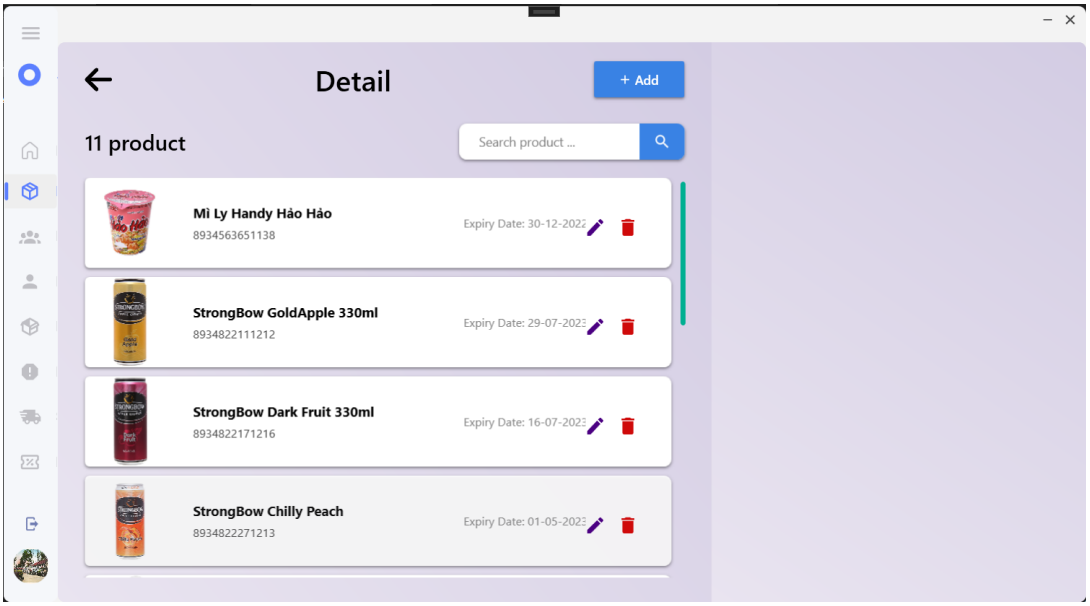

Financial report interface



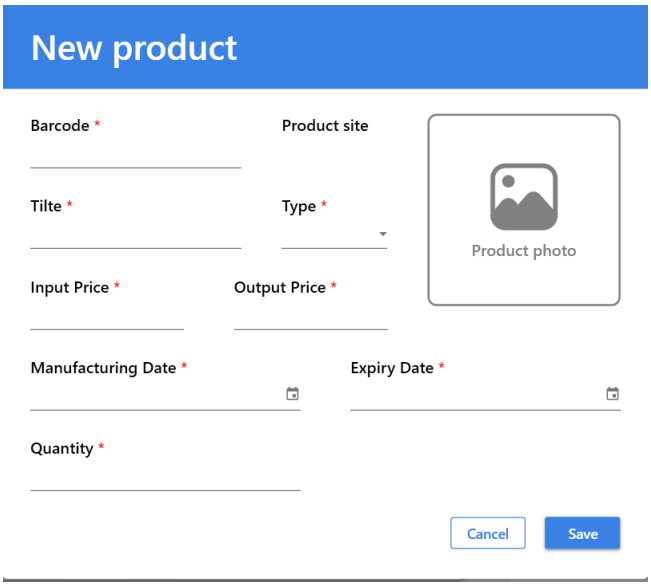

Warehouse management Interface (show the list of shipment)



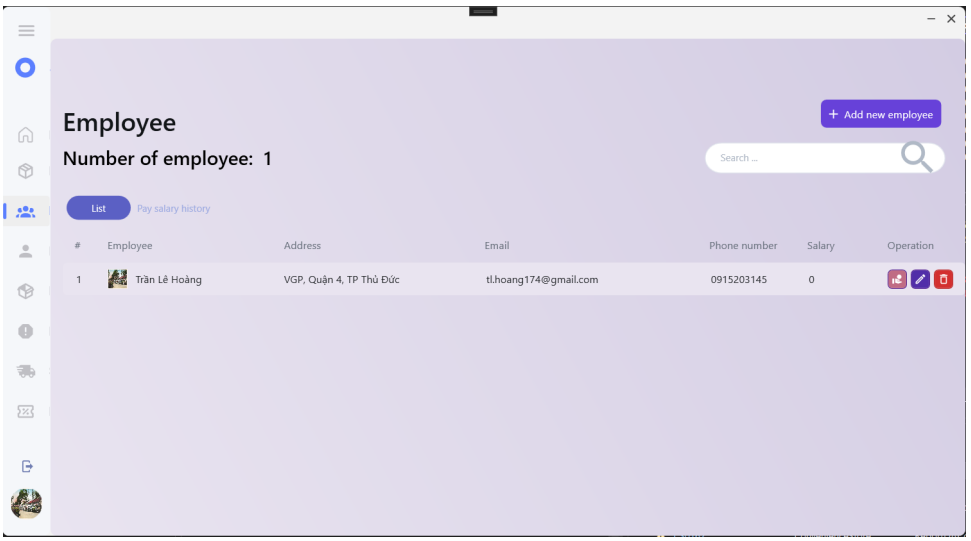

Create a new imported package interface (after clicking on “Import good”)



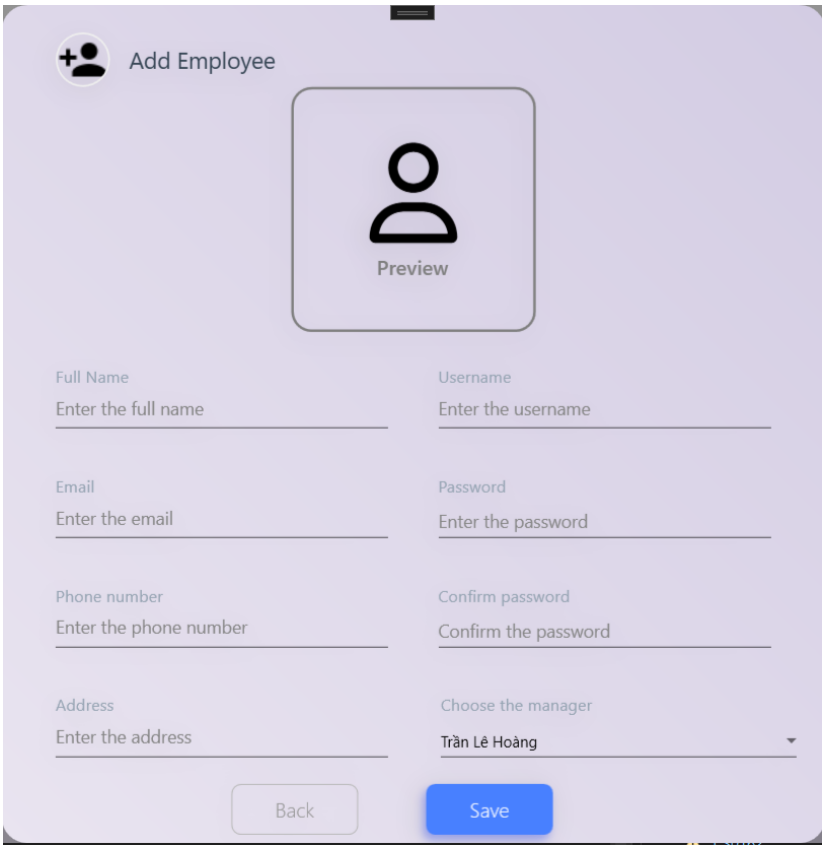

Details about the selected imported shipment (after clicking on any package in the warehouse interface)



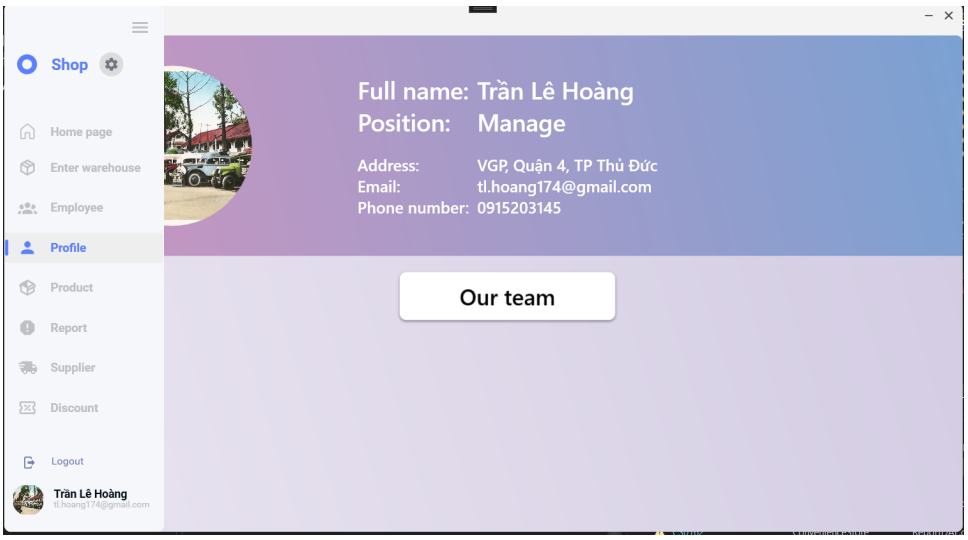

New product submission form (after clicking on “Add” in the previous interface)



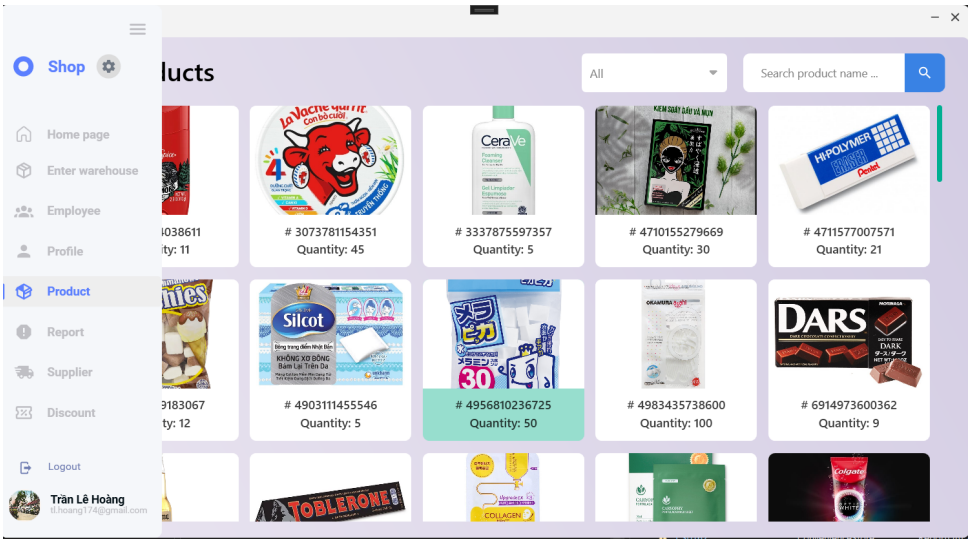

Employee list interface




New employee submission form (after clicking on “Add new employee” in the previous interface)

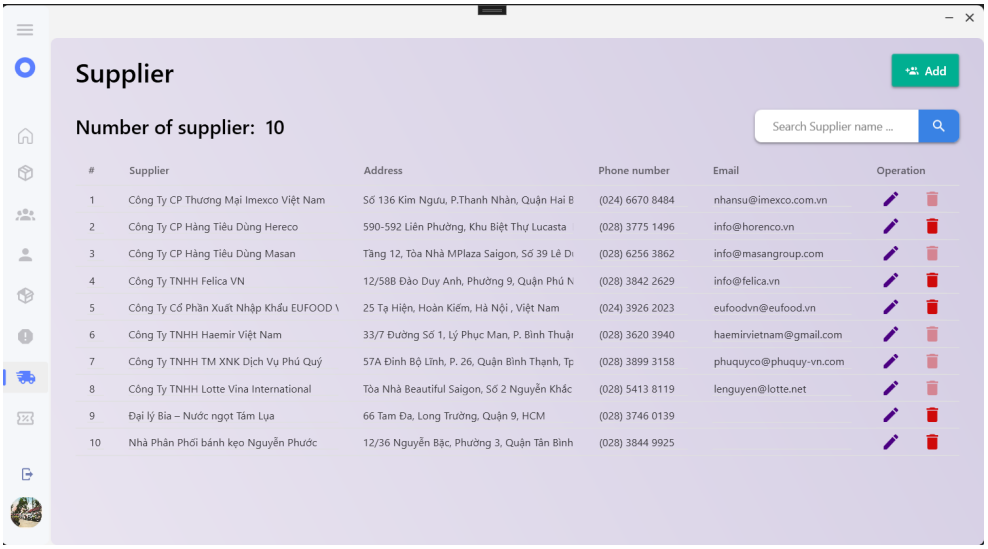



Admin profile

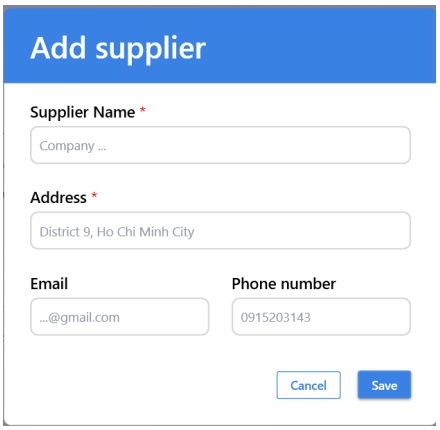



Product interface, listing of all the product and their respective amount in the warehouse

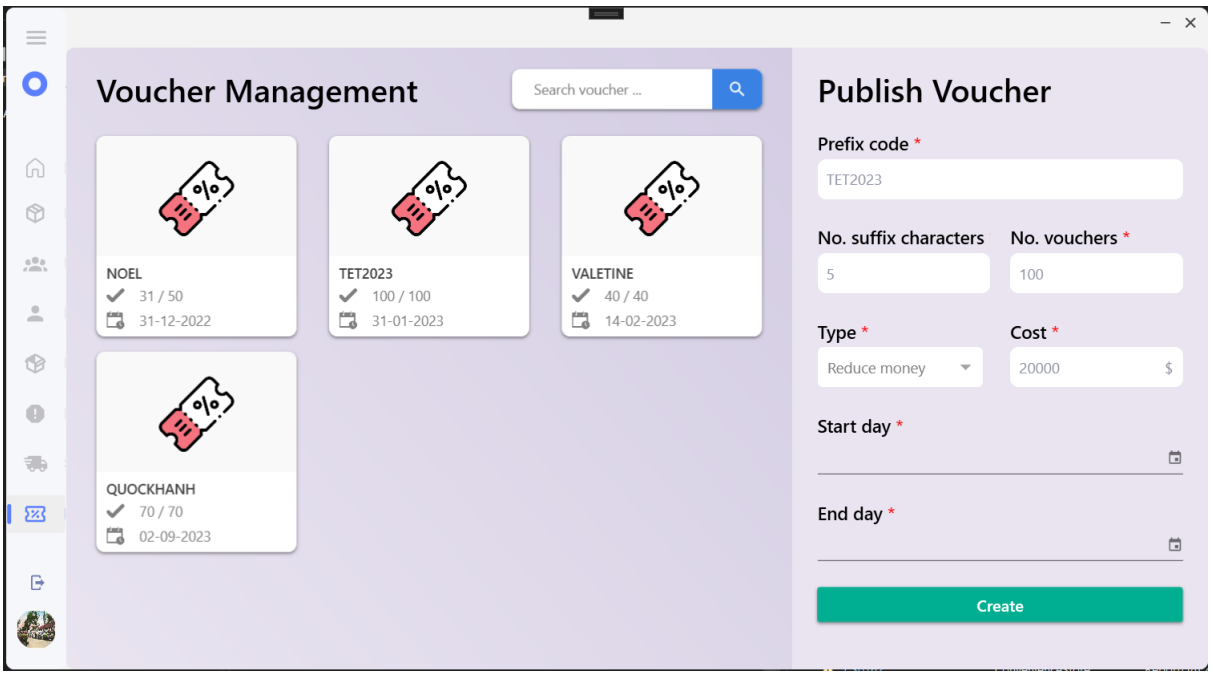

The detail of each product in the warehouse (after clicking on any product in the previous interface). On the left side, the interface shows information about the product, and on the right side, it shows the correlated imported package that it belongs to



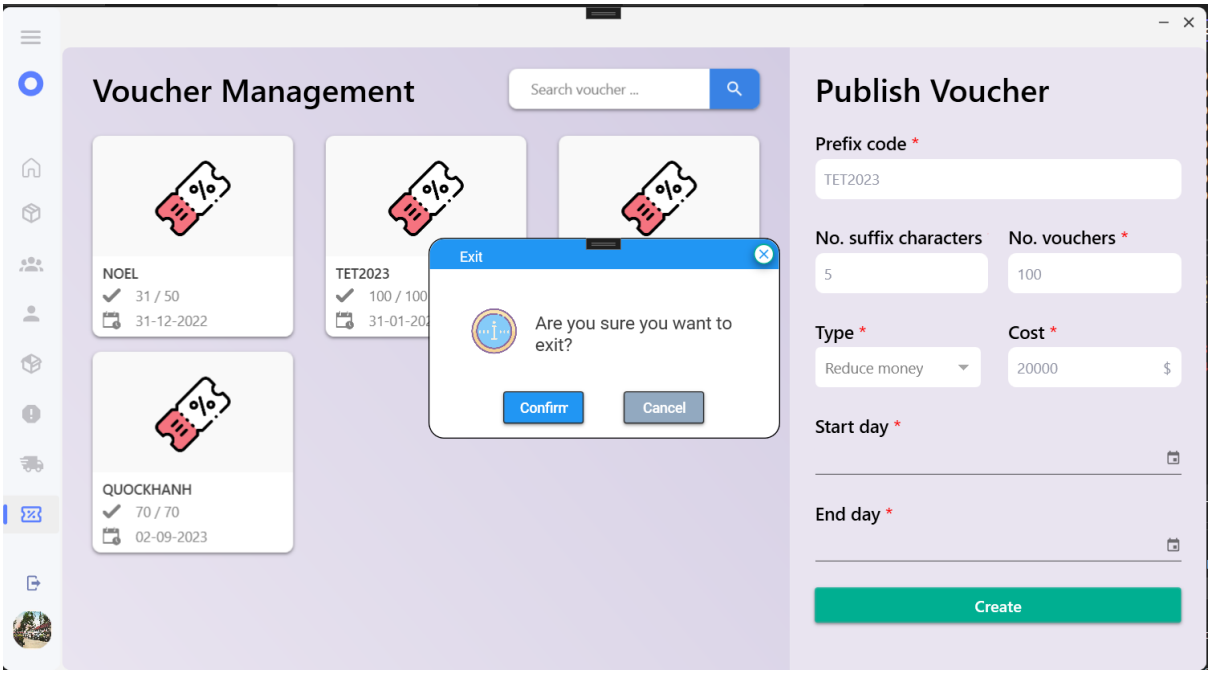

Suppliers list interface




New supplier submission form (after clicking “Add” in the previous interface)



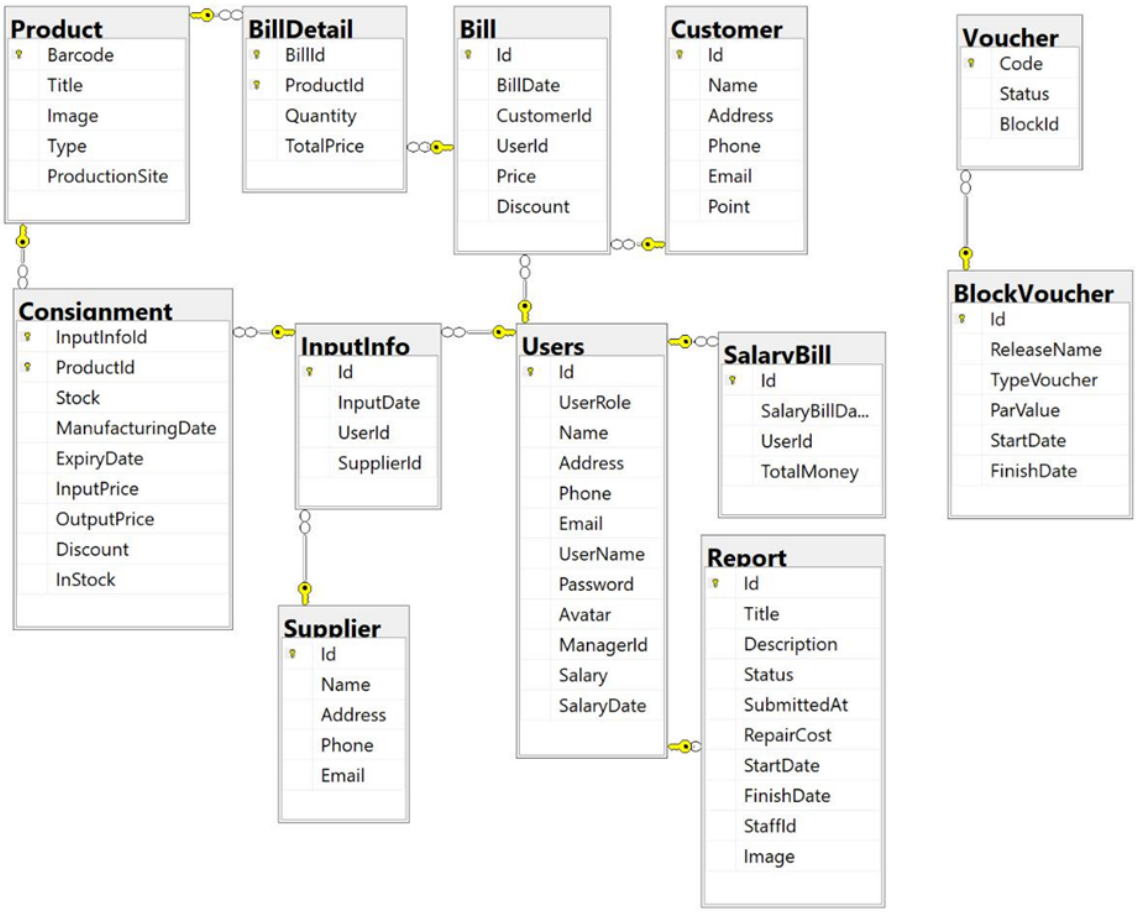

Voucher management interface




Exit confirmation interface **(for all users)**

# Flowchart

Here is a flowchart of the system, indicating the attributes of different classes, their respective attributes and the relationship between them in the form of an entity relationship diagram




# Role of Team Members

Developing the Supermarket Management System was a collaborative effort, with each team member playing a vital role in its creation. The following table summarises the roles of each team member:

| Team Member | Role |
|---|---|
| Phuong Nam Dam | GUI developer |
| Minh Khanh Pham | Report writer, Flowchart designer |
| Sanghyeon Park | Report writer, Testing and QA |