Course Submission Cover Sheet

Module: Application Development

Assignment no: 001

Weighting: 100% of the module mark

Deadline: 12PM sunday the 1st of February 2026

Module Leader: Migara Alawatta Student ID:E280041

Please note that there are specific regulations concerning **the use of AI and Academic Misconduct**. Below are extracts from these regulations. By signing, you acknowledge that you have read and understood these extracts.

(signature:) yasaspasindufernando@gmail.com Date: 01/02/2026

This header sheet should be attached to the work you submit.

Academic Integrity means being honest in your academic work and your studies and making sure that you acknowledge the work of others and giving credit where you have used other people's ideas as part of presenting your arguments. Your assessment submissions must therefore always be entirely your own work, based on your own learning and appropriately referenced including how you have used Generative AI. The University regards the use of Generative AI applications by students to deceive to gain unfair advantage as **academic misconduct**. This usage includes:

- **Plagiarism**, where AI tools are used to generate output and ideas that are presented or submitted as if they were the student's own work, without proper citation or references.
- Where a complete assignment is created using Generative AI and represented as a student's own work, this will be regarded as contract cheating in the same way as commissioning an 'Essay Mill' or other third party to complete your work. Further information can be found on : Guidance on the use of Artificial Intelligence.

Academic misconduct: The University takes academic misconduct very seriously and seeks at all times to rigorously protect its academic standards. Plagiarism, collusion and other forms of cheating constitute academic misconduct, for which there is an explicit range of graduated penalties depending on the particular type of academic misconduct. The penalties that can be applied if academic misconduct is substantiated range from a reprimand to expulsion in very serious cases and for repeated instances of misconduct. You are also responsible for ensuring that all work submitted is your own and that it is appropriately referenced. The University does not tolerate cheating of any kind. You are strongly advised to familiarise yourself with the Academic Misconduct Policy and Procedure (Academic Misconduct), which list a range of categories of academic misconduct and associated penalties, covering instances of academic misconduct (plagiarism, collusion, exam cheating).

Acknowledgement

The GreenLife Organic Store project functioned as my complete personal assignment for Application Development (AD) module which Mr. Migara Alawatta supervised. The project demonstrates how software engineering principles together with development techniques learned during the module work together to create practical software solutions.

The application was designed as a desktop-based system to support the management of a small organic grocery store, providing functionality for both administrative users and customers. The development team built the application using C# programming language together with Windows Forms for user interface design and MySQL as their database management system. I used essential software engineering principles during development to create layered architecture and implement the repository pattern and transaction management and input validation and secure authentication systems which included password hashing.

The project helped me gain better knowledge about systematic development methods together with effective problem-solving techniques and complete system architectural design. I need to express my gratitude for the module guidance, which enabled me to successfully plan and execute system testing and documentation, for my system development work..

Abstract

The GreenLife Organic Store project demonstrates its development through a retail management system which operates on desktop computers. The application creates a virtual representation of daily activities in a small organic grocery store while offering complete software solutions that assist both administrative staff members and customers. The system enables administrators to handle product, category, user, order, review, and discount management functions while customers can explore products, search for items, manage their shopping carts, complete purchases, track their orders, and submit reviews.

The system was developed using C# Windows Forms as the front-end technology and MySQL as the relational database management system. The system utilizes a layered software architecture design to establish distinct separation between user interface components and business logic elements and data access technologies, which results in greater system maintainability and readability and scalability. The repository pattern was used to create a centralized database management system, while the utility services handled functions such as password storage and email delivery and image processing and PDF document creation. The application implemented input validation together with exception handling systems to improve system reliability and enhance user interaction.

The system design process included security elements as part of its development. Passwords are stored using hashing techniques rather than plain text, and parameterized SQL queries are used to prevent SQL injection. The checkout system uses transaction management to guarantee

that both order creation and stock updates happen together, which protects database integrity during system breakdowns.

The project used an incremental development method which introduced new features through continuous manual testing and ongoing system improvements. The testing team performed cross-device tests to confirm that the system worked properly on various display configurations and different operational environments. The coursework enabled students to develop essential software engineering skills through their part of the project which involved designing systems and integrating databases and creating user interfaces and executing tests and producing documentation.

Introduction

Business operations now rely on software systems to bring efficiency and accuracy to their daily processes which apply even to small retail stores. The process of handling products and customers and orders through manual methods results in operational faults and creates delays while generating inconsistent data. The project presents GreenLife Organic Store which functions as a desktop retail management software developed through the Application Development module.

The primary goal of this system is to recreate the authentic operational activities of a small organic grocery store. The application enables users to perform both administrative and customer functions through its complete system which includes product management and customer service and order processing and report creation. The system was created through C# Windows Forms which built the user interface and MySQL served as the backend database system.

The development process used software engineering principles which included layered architecture and database integration and input validation and secure authentication and transaction management. The project used a phased implementation method which involved developing and testing and enhancing features in distinct stages. The method enabled early problem detection which helped maintain system reliability until its final delivery.

Functions Provided

The system provides the following main functions:

The admin role enables users to handle all aspects of product management which includes handling categories and user management and customer management and order processing and review handling and discount creation and low-stock alert system and PDF sales report generation.

The customer system enables users to register or log in while they can search through products and handle their shopping cart and complete their purchases and monitor their order progress and access their prior purchases and write product evaluations.

The system services provide password hashing functions for user authentication purposes and input validation through established rules and the system allows users to create orders which automatically updates inventory and users receive email notifications for system events which include welcome messages and password resets and order status changes.

Scope

This project focuses on the core features required to operate a small organic grocery store in a desktop environment. The system enables users to manage their accounts while they can handle inventory and product categories and processing of shopping carts and checkout and order tracking and review writing and discount management and reporting and MySQL database storage. The system is intended for academic demonstration and small-scale use, therefore advanced enterprise features such as multi-branch support, real online payment gateways, and large-scale performance optimisation are outside the current scope. The current system architecture supports future developments because of its established layered design and repository structure which enables developers to implement new functionalities during upcoming development cycles.

System Overview

The GreenLife Organic Store functions as a retail management software which operates on desktop computers and was created using C# Windows Forms together with MySQL. The system replicates daily operations of a small organic grocery store through its dual functionality which provides distinct features for both administrators and customers on one unified system.

The dashboard system provides administrators with secure access to handle all aspects of product management product category management customer management order handling discount management and report generation. Customers can register, log in, browse products, search by keywords or categories, add items to a shopping cart, place orders, and track order status. The system also supports additional features such as product discounts, customer reviews, email notifications, and PDF report generation.

The system implements a technical framework that uses a three-dimensional architectural approach which separates its user interface components from its business functionality and database connection parts. The structure of the system enhances its ability to be maintained and read by users while it can be expanded to handle more demands. The MySQL relational database system stores data through its design which uses normalized table structures that establish distinct entity relationships between Users and Products and Orders and OrderItems and Reviews and CartItems and Discounts.

The application serves an educational purpose which shows how software engineering principles organize development work through object-oriented programming repository pattern transaction handling validation and secure authentication. The system simulates a real-world retail system which has been designed for optimal operation in educational projects that require small-to-medium scale usage.

Table of Contents

1. Detailed Instructions to Run the Program

1.1 System Requirements

- **Operating System:** Windows 10 / Windows 11 (64-bit)
- **Framework:** .NET 8 (Windows Forms)
- **IDE:** Visual Studio 2015+ (Recommended: Visual Studio 2022)
- **Database:** MySQL Server 5.7+ / 8.0+
- **RAM:** Minimum 4 GB (Recommended: 8 GB)
- **Disk Space:** Minimum 500 MB

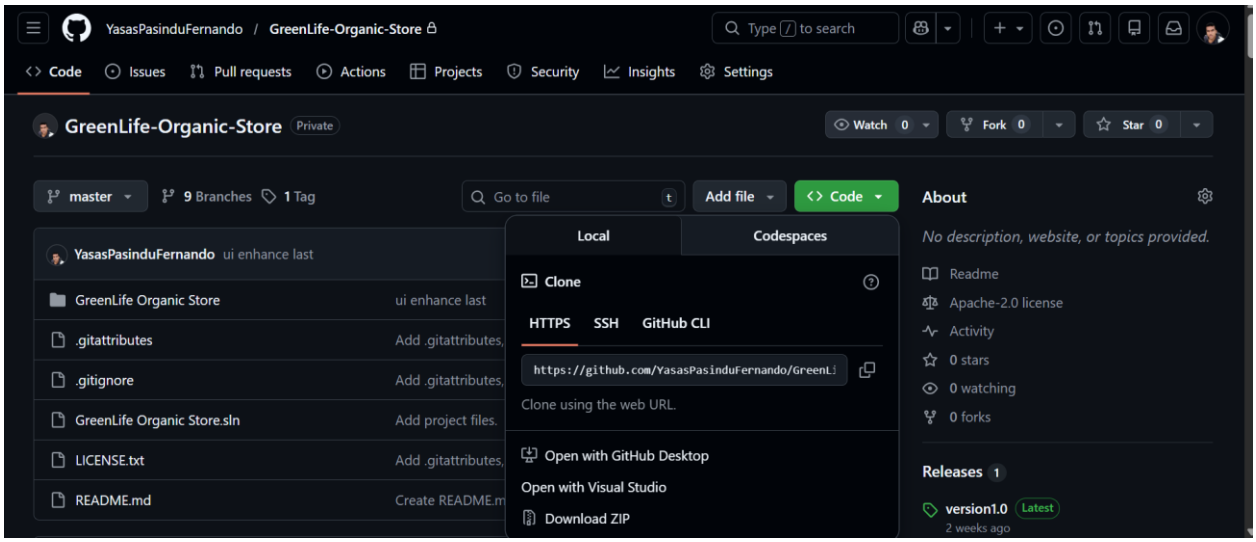
1.2 Installation Steps

Step 1: Get the Project

- Download the ZIP **or** clone the repository.

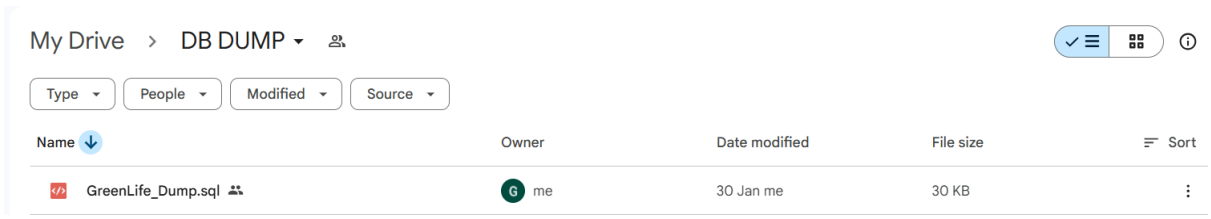
git clone https://github.com/YasasPasinduFernando/GreenLife-Organic-Store.git

cd "GreenLife Organic Store"



Step 2: Setup the MySQL Database

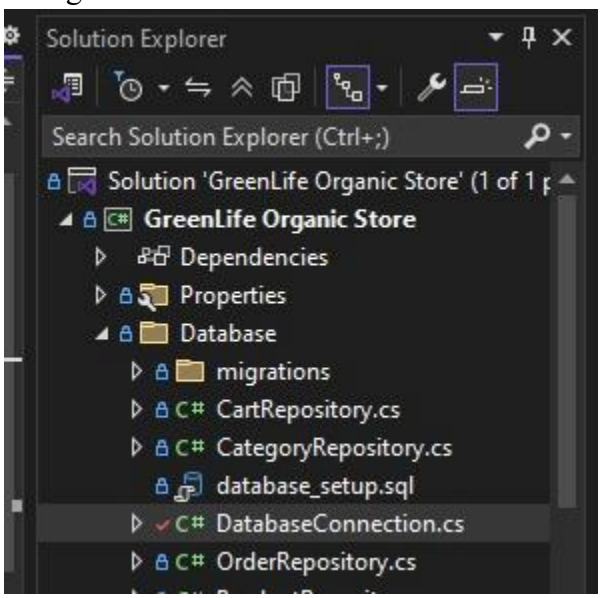
- Download the GreenLife_Dump.sql from this **link** https://drive.google.com/drive/folders/1vzYOeD7xiYvzLhnE6GZzCbIeKawjDy9e



1. Open **MySQL Workbench** (or MySQL CLI)
2. Import the **GreenLife_Dump.sql** script to create the database schema (tables and relationships) and populate it with core data.

Step 3: Configure Database Connection

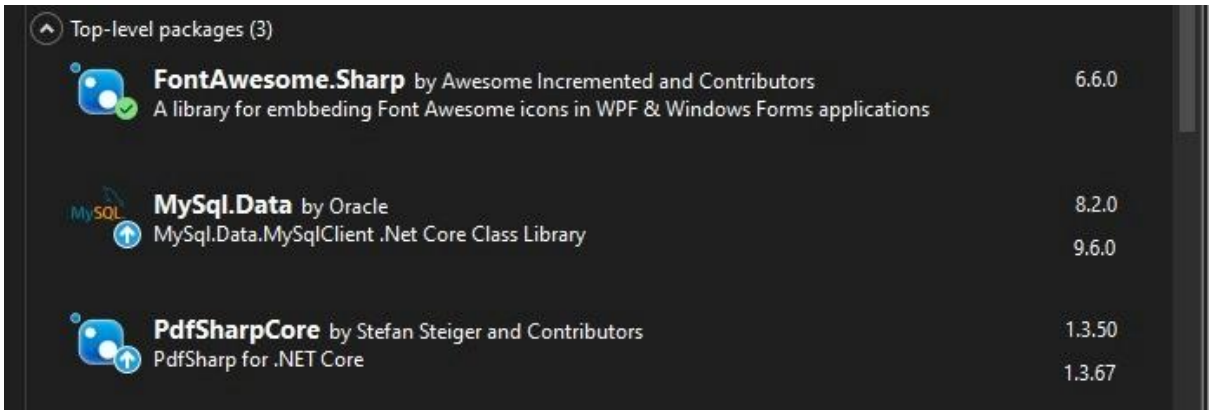
1. Open the solution in Visual Studio.
2. Navigate to: Database/DatabaseConnection.cs



3. Replace the connection string with your own.

```
Private static string connectionString="Server=localhost;Port=3306;
Database=greenlife_store;Uid=root; Pwd=YOUR_PASSWORD;";
```

Step 4: Restore NuGet Packages



- Visual Studio → **Tools → NuGet Package Manager → Restore**

This step ensures that all external libraries (such as **MySql.Data**, **FontAwesome.Sharp**, and **PdfSharpCore**) are downloaded and installed automatically. If this step is skipped, the project may fail to build due to missing dependencies.

- Build once: **Build → Build Solution**

Step 4: **(Alternative):** Install NuGet Packages Manually

If the packages are not **restored automatically**, install them manually:

- Open **Visual Studio**
- Go to **Tools → NuGet Package Manager → Manage NuGet Packages for Solution**
- Select the **Browse** tab
- Search and install the following packages:

Package Name	Version	Purpose
MySql.Data	8.2.0	MySQL database connectivity
FontAwesome.Sharp	6.6.0	Icons for Windows Forms UI
PdfSharpCore	1.3.50	PDF report generation

- After installation, click **Build → Build Solution**

Step 5: Run the Application

- Press **F5** (Start Debugging)
- The application opens with the **Login Screen**

1.3 Login Credentials

For coursework demonstration, the system includes sample accounts:

- **Admin:** admin@greenlife.com / admin@123
- **Customer:** customer@gmail.com / customer@greenlife.com

Important: For security, real personal passwords should not be included in reports. These are **demo credentials** created only for testing and presentation.

To test email and notification features, examiners may create an additional test user account using their own temporary email address if they wish to verify email functionality. The system also includes a **mock email mode**, which allows email features to be demonstrated without sending real emails.

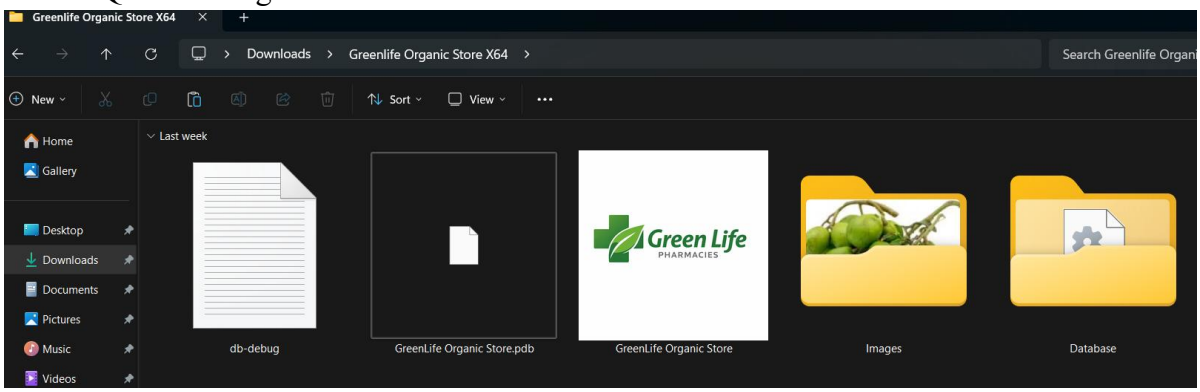
1.4 Running as an Executable

1. Build the solution (Release recommended)
2. Go to output folder: bin/Release/net8.0-windows/
3. Run: GreenLife Organic Store.exe

1.5 Portable Version (SQLite Embedded)

For portability and learning purposes, I also prepared a **self-contained portable build** with an **embedded SQLite database**. This version is useful for:

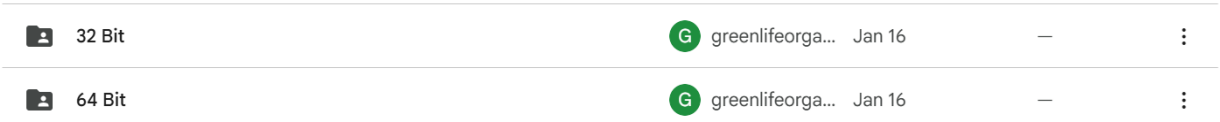
- Running the system on other computers without installing MySQL
- Demonstrating the application in labs/classrooms
- Quick testing on Windows 7 and above



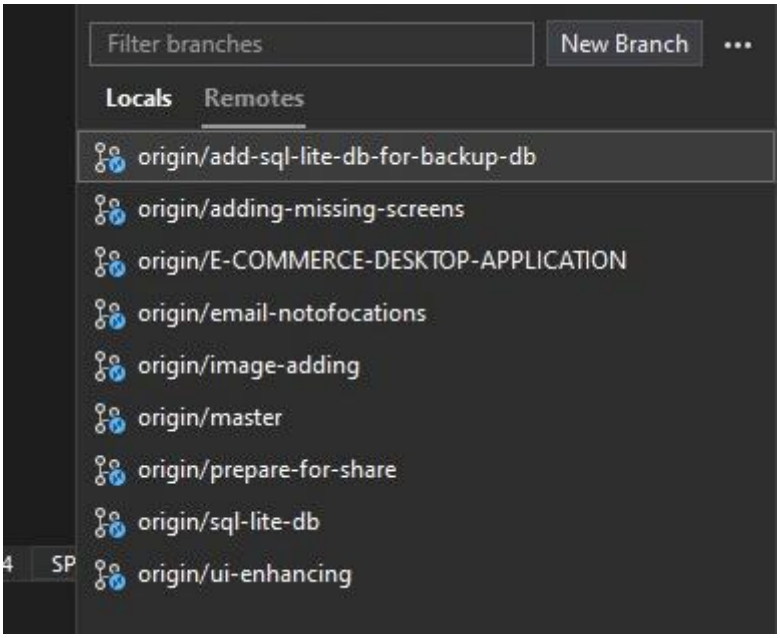
Portable build notes - Works like a normal application (double-click and run) - No separate DB installation required - Provided as an additional deliverable (separate package)

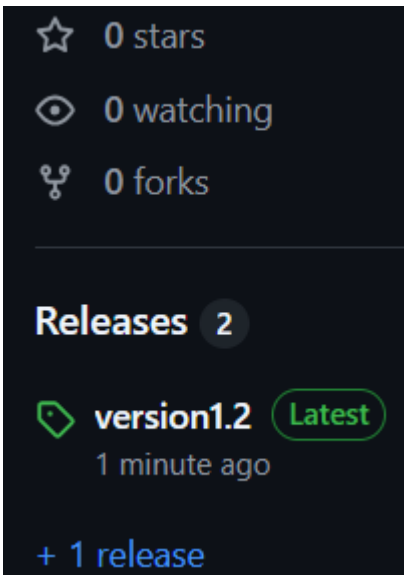
Download location (portable build / MySQL build packages): - Google Drive folder:

https://drive.google.com/drive/folders/1z57JX-LEv5VVXct0lQqeniai2r6QSBSO



For academic review of code, the primary submission is the **MySQL version** (master branch). The SQLite portable build is provided as an extra deployment approach.

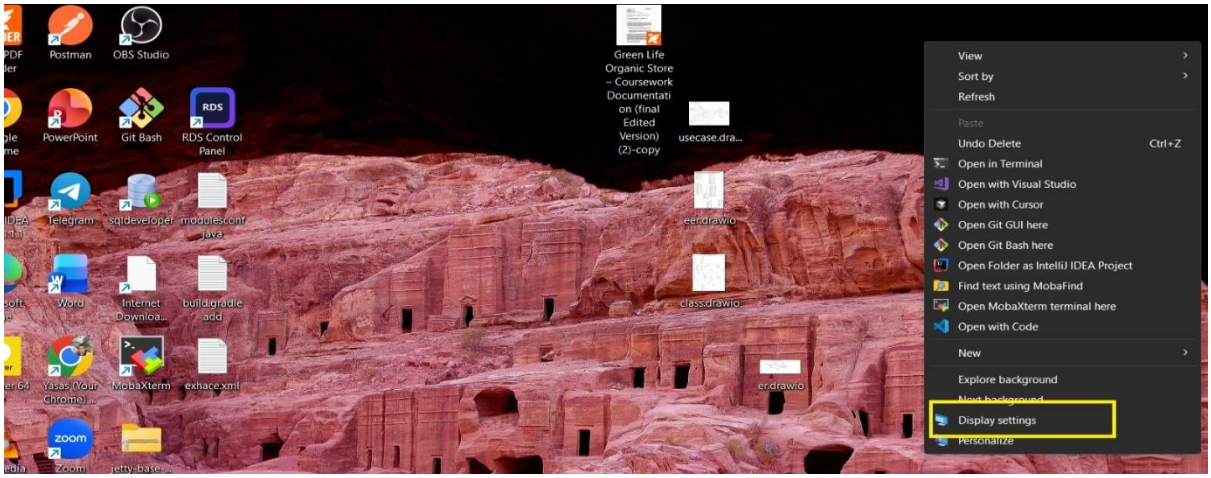


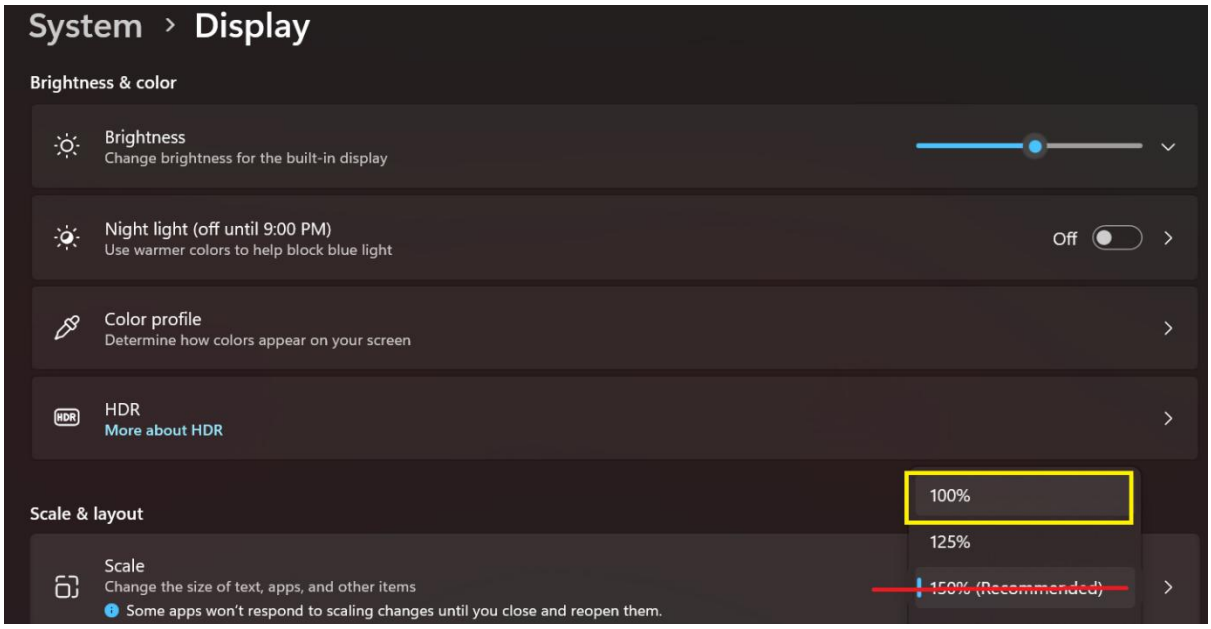


1.6 Troubleshooting

Issue	Solution
Cannot connect to database	Ensure MySQL service is running + verify connection string
Tables missing	Re-run SQL schema script
MySql.Data missing	Restore NuGet packages / install MySql.Data
App crashes on startup	Clean Solution + Rebuild Solution
Images not loading	Ensure Images folder exists + stored paths are valid

Display Scaling Note: - In certain environments, Windows display scaling beyond **100%** (e.g., 125%, 150%, etc.) may cause minor UI layout issues. **For best visual layout** while **demonstratin**g the application, it **is suggested to set display scaling to 100%.** The application has DPI-aware settings for compatibility with different screen resolutions.





Provide Appendix E: user guidance for the users to be aware of how to use the application

2. Software Architecture

2.1 System Overview

GreenLife Organic Store is a desktop-based e-commerce management system built using **C# Windows Forms** and **MySQL**. The solution follows a **Layered Architecture** approach to keep UI, business logic, and database operations separated for better maintainability.

Architecture Layers

- **Presentation Layer:** Windows Forms (Login, Dashboards, Management Screens)
- **Domain Layer:** Models (User, Product, Category, Order, OrderItem, CartItem, Review)
- **Data Access Layer:** Repositories (UserRepository, ProductRepository, OrderRepository, CategoryRepository, CartRepository, ReviewRepository)
- **Utility Layer:** Password hashing, email service, image handling, report generation
- **Database Layer:** MySQL tables with relationships and constraints

2.2 Design Principles

- **Separation of Concerns:** UI focuses on user interactions, repositories handle SQL.
- **OOP (Object-Oriented Programming):** Entities represent real-world objects (User, Order, Product).
- **Modularity:** Features are separated by forms and repositories.
- **Maintainability:** Consistent naming, reusable utilities, centralized error handling.

2.3 Design Patterns Used

1. **Layered Architecture Pattern**: clear separation between UI, domain logic, and database access.
2. **Repository Pattern**: all SQL/database operations are centralized in repository classes.
3. **Utility Service Pattern**: shared services for hashing, email, images, and reports.
4. **Transaction Pattern (in OrderRepository)**: ensures Orders + OrderItems + stock updates are consistent.

If a user wants to do something on the system, such as ordering a product or changing product information, the request passes through a number of layers in a particular order. In the Presentation Layer, which uses Windows Forms, the system gets information from the user and performs a number of tests. The request then passes to the Domain Layer, where the business logic is carried out, such as calculating the total cost of the product, checking product availability, and calculating the discount price of the product. The request then passes to the Data Access Layer, where work is carried out on the database. This includes talking to the MySQL database through repository classes and making safe queries. Along the way, Utility services are used to carry out tasks such as hashing passwords, sending emails, storing images, and making reports. Each of the layers performs a particular role, making it easier to reuse code, making it easier to debug, making it easier to test, and making it easier to extend the system in the future.

3. Project Folder Structure

The GreenLife Organic Store solution uses a system folder structure that establishes multiple levels of system architecture. The system organization was created with the goal of distributing application tasks to different components which makes it simpler to comprehend and manage and expand the system. The project structure organizes related components which helps developers follow clean code standards while making their work more understandable to others who review their work.

```
GreenLife Organic Store/
│
│
├── Forms/          (UI screens)
│
├── Models/         (Entity classes + enums)
│
├── Database/       (Repository classes + DatabaseConnection)
│
├── Utilities/      (Password hashing, email, images, validation)
│
├── Reports/        (PDF report generation)
│
└── Images/         (Stored product images)
```

Why this structure matters:

- Separates UI and database code
- Simplifies marking and maintenance
- Follows good software engineering practices

Forms:

This directory holds all Windows Forms user interface screens which the application uses including the login screen, admin dashboard, product management screens, shopping cart, checkout process, and reporting tools. User interface elements control user interaction and display user interface elements while other system components handle business operations and database functions. The separation process maintains UI elements in a clean state which displays only design elements..

Models:

The Models folder contains the core domain entities that represent real-world objects in the system, such as User, Product, Category, Order, OrderItem, CartItem, Review, and Discount. These classes define the data structure and include small pieces of business logic (e.g., price calculation or discount handling). The system maintains a precise data representation through model separation which prevents any UI or database code from entering the application.

Database:

The Repository Layer uses this folder to manage all MySQL database communications through its various database access classes. The system contains UserRepository and ProductRepository and OrderRepository and DiscountRepository and DatabaseConnection as its database access classes. The system stores all SQL queries together with database operations in this centralized location. The system design implements the Repository Pattern which keeps SQL code from spreading across user interface components thus enhancing application maintenance and debugging processes.

Utilities:

The system uses shared helper services from the Utilities folder which provide essential functions for various processes throughout the system. Examples include:

- PasswordHasher for secure password storage
- EmailService for sending notifications
- ImageStore for managing product images
- Input validation helpers

The system components allow reusable functions which work independently from both user interfaces and database systems thus enabling developers to create separate modules while cutting down on necessary program code.

Reports:

The folder contains all the classes needed to create PDF reports which include sales reports and order summaries. The system uses PdfSharpCore to create formatted reports that administrators can access. Document generation works independently of core application functions since the system separates its reporting logic from other system components.

Images:

The folder contains product image files which the application uses. The system saves relative paths in the database instead of absolute file paths which enables the project to work on different computers and environments as well helps for the first application run.

Why This Structure Matters:-

- The organized folder system provides multiple essential advantages.
- Separation of concerns - UI, business logic, and data access are clearly separated
- Improved readability - lecturers and developers can quickly understand the system layout
- Simplified maintenance -changes in one layer do not heavily affect others
- Scalability -new features (like Discounts) can be added without restructuring the whole project
- Professional practice — reflects real-world software engineering standards

The structure enables the creation of a clean application design which maintains maintainability and extendability while following layered architecture design rules.

4. Database Design (ER) Summary

The GreenLife Organic Store database system uses a relational database system which has been developed according to normalized standards to meet the requirements of retail operations found in small to medium sized businesses. The design focuses on reducing data duplication while achieving consistent data results and enabling actual e-commerce business activities that include product administration and order processing and shopping cart management and customer review handling and discount administration.

The structure follows third normal form (3NF) principles, where each table represents a single entity and relationships between entities are maintained using primary keys (PK) and foreign keys (FK). The system maintains data integrity because all operations for updating and insertingand deleting records execute correctly to prevent any data inconsistencies from entering the system.

4.1 Main Tables

- **Users** (Admin/Customer)
  The table keeps track of all system users who include both customers and administrators. The system contains user information which includes their login credentials and contact data and their specific role (UserType) in the system. This table serves as the system's main component for user authentication and access control.
- **Categories**
  The system uses categories to classify products into their appropriate product groups. The system allows customers to browse products through multiple product categories, which enable them to filter their search results.
- **Products**
  The system database contains product information through various attributes. The CategoryID foreign key links each product to a specific category.
- **Orders**
  The system handles customer purchases through this module. The module contains all order information, which includes details about the customer and the total order value and the current order state and the delivery location for the order. The system connects each order to a specific user through their CustomerID relationship.
- **OrderItems**
  The table holds all items that make up an entire order. The system uses product records to connect products with orders while keeping information about purchased products, which includes their quantity and unit price and subtotal at the time of purchase. The system maintains pricing information from the past, which remains unchanged despite future product price changes.
- **CartItems**
  The system maintains CartItems As temporary cart data which users create until they complete their checkout process. The system uses this connection to direct users toward their intended products while it monitors their requested product quantities.
- **OrderReviews**
  The system enables customers to provide ratings and feedback through their completed order evaluations. Each review links to both the customer and the order which enables review moderation and customer feedback tracking.
- **Discounts**

The system keeps track of all active promotional discount information which businesses apply to their products. The discount system requires three elements which include percentage value and start date and end date and active status. This system enables businesses to set up promotional pricing which depends on specific time periods.

4.2 Key Relationships

The relationships between the tables model real-world store behavior:

- Each user can place multiple orders, identified by **CustomerID**.
- Each order may contain multiple order items, linked to **OrderID**.
- Each category can include multiple products, associated via **CategoryID**.
- Each user may have several cart items, specified by **UserID**.
- Each product can appear in multiple cart items, referenced by **ProductID**.
- Each order may receive multiple reviews, connected through **OrderID**.
- Each product can have zero or many discounts, linked using Discounts. Productid → Products.ID (one-to-many).
- Only one discount is treated as "active" for a product at a time based on Is Active + date range (StartDate–End Date).

  The system maintains multiple historical discounts for a product but only one discount reaches active status. The system controls this process through the following methods: The active state remains true The current date exists within the period between the StartDate and EndDate.

The ER and EER diagrams are included in the submission Appendix C (Diagram Section).

5.Detailed Description of Classes(Core Models, Repositories, Utilities)

The section describes the essential system classes which distribute their duties among three components which are models and repositories and utility services. The application achieves better maintainability and readability and scalability through this method of separation.

5.1 Core Domain Models

Domain models represent the real-world business entities of the system. The classes hold both entity data and minor business logic components which belong to those entities.

5.1.1 User

All system users are represented by this model which includes **admin** and **customers**. The User Type enumeration is used to control role-based access within the system.

Key Properties:

- ID, Email, Name, Phone, Age, Address
- Gender (enum), UserType (Admin/Customer)
- HashedPassword, CreatedDate, UpdatedDate, IsActive

Purpose: This model handles user authentication and role separation together with user data security.

5.1.2 Product

The organic store sells products which this system represents through information about pricing and stock and promotional details.

Key Properties:

- ID, ProductName, CategoryID, Description
- Price, DiscountPrice, Stock, Supplier
- ImagePath, IsFeatured, IsActive

Key Methods:

- IsInStock(),
- GetFinalPrice(),
- HasDiscount(),
- GetDiscountPercent(),
- GetStockStatus()

Purpose: The model contains all pricing and stock management functions which control product inventory throughout the application.

5.1.3 Category

Organises products into categories.

Key Properties:

- ID, CategoryName, Description, ImagePath, IsActive

Purpose: The system enables users to navigate through products while maintaining an organized product display.

5.1.4 Order & OrderItem

Records purchase transactions and associated items.

Order Properties:

- ID, OrderNumber, CustomerID
- CustomerName, Phone, Email
- OrderDate, TotalAmount, Status
- ShippingAddress, Notes, Items (List)

Purpose: The models work together to enable order processing and financial tracking through their combined functionalities.

OrderItem Properties:

- ID, OrderID, ProductID, ProductName
- Quantity, UnitPrice, Subtotal

5.1.5 ShoppingCart (Static)

The system stores items temporarily in memory which will be removed during the checkout process.

- Items (List)
- AddItem(), RemoveItem(), UpdateQuantity(), Clear()
- GetTotal(), GetItemCount(), HasItems()

Purpose: The system enables customers to control their selected items until they finalize their purchase because it restricts database updates until that point.

5.1.6 OrderReview

The system records customer feedback together with product ratings which customers submit after their purchase.

Key Properties:

- ID,OrderID, CustomerID
- Rating (1 to 5 stars)
- Comment,
- ReviewDate,IsVisible (for moderation)

Purpose: The system collects customer feedback while giving administrators tools to control review content.

Key Methods (Business Logic Examples):

- IsValidRating(),GetShortPreview(),CanCustomerEditReview()

5.1.7 Discounts

A promotional price reduction offered on products for a fixed time period.

Key Properties (example list):

- ID, DiscountName, DiscountPercent, ProductID
- StartDate, EndDate, IsActive, CreatedDate, UpdatedDate

Purpose: The system separates promotional pricing from standard product pricing while it allows automatic promotion activation which depends on time intervals.

5.2 Repository Layer (Data Access)

Repositories encapsulate all database operations using **ADO.NET (MySql.Data)** with parameterized queries.

- **UserRepository:** Authenticate, create/update users, password reset workflow

- **ProductRepository:** CRUD products, search, featured products, low-stock list
- **CategoryRepository:** CRUD categories
- **OrderRepository:** Create orders with transaction + status updates
- **CartRepository:** Optional cart persistence
- **ReviewRepository:** Create/update/delete reviews
- **DiscountRepository**: CRUD discounts, find active discount by product, validate date range, sync active discount status

Benefits of Repository Pattern: Using the repository pattern, SQL queries were avoided in the form classes. The code is now easier to manage. The database error handling is now centralized instead of being duplicated throughout the UI. This has made the structure of the application better. It will be easier to maintain in the future.

5.3 Utility Layer

- **PasswordHasher:** HashPassword(), VerifyPassword()
- **EmailService:** Sends order confirmation / password reset emails (SMTP)
- **ImageStore:** SaveImage(), GetImagePath(), DeleteImage()
- **PdfReportGenerator:** GenerateSalesReport(), GenerateOrderReport()

The Class Diagram is included in the submission Appendix C (Diagram Section).

5.4 NuGet Packages & External Libraries

To enhance the user interface, database connectivity, and reporting capabilities, the project uses a minimal set of direct NuGet dependencies:

Package	Purpose in GreenLife	Notes
FontAwesome.Sharp	Modern UI icons (buttons, menus, actions)	Improves UI clarity and consistency
MySql.Data	MySQL database connectivity (ADO.NET provider)	Used across repositories with parameterized queries
PdfSharpCore	PDF report generation (Sales / Order summaries)	Generates formatted multi-page PDF reports

The email alert feature uses the **.NET library** that is available in the system, which is **System.Net.Mail.** A **mock mode** is also added in the **system** so that the emails are not sent but the content is displayed within the interface of the application. This is done to make the testing process easier.

For security purposes, the actual database credentials, email passwords, and API keys are not provided in this report. They are replaced with dummy data.

6. Requirements Specification

The GreenLife Organic Store system needs this section to specify its essential functions and its performance characteristics. The requirements establish the system's necessary functions and its required performance standards.

6.1 Functional Requirements

Functional requirements describe the specific features and operations the system must perform.

User Account Management

- The system shall allow customers to **register** for an account.
- The system shall allow users to **log in and log out** securely.
- The system shall allow users to **reset passwords** via email.
- The system shall allow customers to **edit their profile details**.
- The system shall allow admins to **view and manage customer accounts**.

Product Management

- The system shall allow admins to **add new products**.
- The system shall allow admins to **edit product details** (price, stock, description, image).
- The system shall allow admins to **delete or deactivate products**.
- The system shall display product availability based on stock level.

Category Management

- The system shall allow admins to **create, update, and delete product categories**.
- The system shall display products grouped by category.

Shopping and Cart

- The system shall allow customers to **browse products**.
- The system shall allow customers to **search products by keyword**.
- The system shall allow customers to **add items to a shopping cart**.
- The system shall allow customers to **update or remove items** in the cart.

Order Processing

- The system shall allow customers to **place orders**.
- The system shall automatically **calculate order totals**.
- The system shall reduce product stock when an order is placed.
- The system shall allow admins to **view and update order status**.
- The system shall allow customers to **view order history**.

Discount Management

- The system shall allow admins to **create discounts** for products.
- The system shall allow admins to define **discount percentage and valid date range**.
- The system shall display the **discounted price** when a discount is active.
- The system shall apply discounts automatically during checkout.

Review System

- The system shall allow customers to **submit product or order reviews**.
- The system shall store a **rating (1–5 stars) and comment**.
- The system shall allow admins to **view or moderate reviews**.

Reporting

- The system shall allow admins to **generate sales reports**.
- The system shall export reports in **PDF format**.

Notifications

- The system shall send **email notifications** for order confirmation.
- The system shall send **password reset emails**.

6.2 Non-Functional Requirements

Non-functional requirements describe how well the system performs rather than what it does.

Performance

- The system shall load product lists within **a few seconds**.
- The system shall support a **small-to-medium product catalog** efficiently.

Usability

- The system shall provide a **simple and user-friendly interface**.
- The system shall use **clear icons and buttons** for navigation.
- The system shall support **standard screen resolutions** and DPI scaling.

Security

- The system shall store passwords using **hashing**.
- The system shall use **parameterized SQL queries** to prevent SQL injection.
- The system shall restrict access to admin features based on **user roles**.

Reliability

- The system shall use **database transactions** during checkout to prevent data inconsistency.
- The system shall handle errors using **exception handling** without crashing.

Maintainability

- The system shall follow a **layered architecture**.
- The system shall separate UI, business logic, and database access.
- The system shall use the **repository pattern** for data operations.

Portability

- The system shall run on **Windows 10/11**.
- A **portable SQLite version** shall be available for environments without MySQL.

Scalability

- The system design shall allow adding **more products, categories, and users** without major changes.

Compatibility

- The system shall be compatible with **MySQL database**.
- The system shall run using **.NET Windows Forms environment**.

7. Implemented Features

7.1 Admin Features

The admin module enables full control over store operations and product management and customer activities. The module was created to give administrators complete system knowledge while they handle business operations.

- Secure admin login and dashboard navigation
- Manage **Products** (add/edit/delete, image upload, discount price, active/inactive)
- Manage **Users** (add/edit/delete, update)
- Manage **Categories** (CRUD)
- Manage **Customers** (view, update, activate/deactivate)
- Manage **Orders** (view all, status updates, cancellation handling)
- **Low-stock** monitoring (inventory awareness)
- **Sales report** generation and export to **PDF**
- **Review monitoring** / moderation (where applicable)
- **Email notifications** for important operations (optional feature set)
- **Manage Discounts** (add/edit/delete discounts, set percentage and date range, activate/deactivate)
- **Discount validation** (prevent invalid date ranges and negative values)

The admin uses these features to maintain complete and correct records of product information and customer activities and promotional campaigns and order management. The admin dashboard serves as the system's primary control center

7.2 Customer Features

The customer module provides an easy-to-use shopping experience which enables customers to access all store functions. Customers can browse products, place orders, and interact with the store through reviews and profile management features. The customer module provides an easy-to-use shopping experience which enables customers to access all store functions.

- Customer **registration** and **login**
- **Browse categories** and **product** catalog
- **Keyword search** for products
- Shopping **cart management** (add/update/remove)
- **Checkout** and **order** placement
- Order **history** and **status** tracking
- Profile **edits** and **password** change
- Submit **reviews**/**ratings** after purchase (where applicable)
- View discounted **price** and **discount badge** when a valid discount is active
- Checkout totals reflect **discounted** prices where applicable

The store enables customers to search for products and handle their orders while interacting with the store through features that operate like actual e-commerce websites. The discount display and automatic price calculation improve transparency and enhance the overall

shopping experience.The store enables customers to search for products and handle their orders while interacting with the store through features that operate like actual e-commerce websites. The discount display and automatic price calculation improve transparency and enhance the overall shopping experience.

7.3 Forms & UI Highlights

- The UI uses **FontAwesome.Sharp** icons to improve usability and visual clarity.
- The system includes multiple screens such as login, dashboards, product management, order management, cart, checkout, reports, and profile management.
- Cross-device testing helped reduce UI overlap issues; best results are achieved at **100% scaling** (no overlap), while DPI-aware settings improve behavior at 125%/150%.

UI & Layered Design Note:
The aim is to have a clean and responsive UI that integrates well with a layered design. The UI is purely for display and interaction purposes, while the business logic is contained in the domain and repository layers. This makes the UI lightweight and easier to maintain. The use of visual elements such as icons, clean layouts, and grouping of controls enables the user to understand the features quickly without clutter. When combined with DPI-aware modifications, the UI design appears professional and scales well on various screen sizes and resolutions.

8. Search Algorithms Implementation

The system provides essential search and filtering and sorting functions which work effectively with desktop-based store software. The techniques enable users to find products in a fast manner while the system maintains its operational efficiency and simple maintenance.

8.1 Linear Search (Keyword Search)

Approach: filter products by name/description using case-insensitive matching.

- Suitable for small/medium catalog size
- Simple and reliable

Description:
The coursework uses linear search to show how basic algorithms work in real-world applications. The system checks each product one by one to find matching items which it adds to the results. Although linear search is not the most efficient method for very large datasets, it is appropriate for the scope of this project and provides clear evidence of algorithm implementation.

8.2 Category Filtering (Database-Level)

Approach:- To filter the products by category, we used a SQL query with a **WHERE** clause. The SQL query used is: "**CategoryID = ?".** This way, the system will retrieve only the products

in the specified category, as opposed to retrieving all the products first and then filtering them. This is faster because it is done on the database.

- Faster than filtering all products in memory
- Can be optimized using indexes

Description:

The system uses database-level filtering to enhance both performance and scalability. The method uses database engine optimization which provides better efficiency compared to processing data in memory.

8.3 Sorting

- Sort by product name
- Sort by price (ascending/descending)

Description:

Users can use sorting to compare products directly which displays items that meet their needs through two options. The system executes sorting operations through SQL ORDER BY clauses and in-memory sorting based on the specific situation.

8.4 Rationale

The selected search methods together with their filtering techniques provide appropriate matching solutions for a retail application which operates at coursework level and handles moderate datasets. The system provides:

- Simple implementation which users can easily comprehend
- System capabilities that ensure dependable operation and easy process maintenance
- System performance which meets requirements because of expected data processing needs

Database indexing enables faster product retrieval through three criteria which include category selection and price range and stock level. The implementation system shows good performance results because it uses basic techniques which provide educational benefits while maintaining system transparency.

9. Validation, Security & Exception Handling

The application implements multiple validation layers together with security measures and structured error handling systems to maintain system reliability and correct data processing and secure system operations

9.1 Input Validation

The system applies input validation throughout all its components to block any attempt to store invalid data or inconsistent information into the database.

- **Required fields validation:** Users must complete all essential fields which include Email, Password, Product Name, Price, and Stock before they can save a record. This prevents incomplete records. Numeric validation (Price, Stock, Quantity)
- **Email format validation:** Users can only enter valid numeric values into Price, Stock, and Quantity fields. This avoids calculation errors and data corruption.
- **Range validation (Stock must be non-negative):** The system checks email inputs to ensure they match a standard format before users can register or log into their accounts.
- **Discount percentage validation (0–100):** The system prohibits users from entering negative stock values. This prevents organizations from showing impossible inventory situations which would disrupt their operational procedures.
- **StartDate must be before End Date: The** system verifies that discount percentages remain within permissible boundaries when users create or modify discounts. This prevents incorrect pricing behavior.
- **Prevent overlapping active discounts for the same product:** The system prevents multiple active discounts from being applied to the same product at the same time which maintains clear and consistent pricing.

Purpose: The validation rules protect data integrity while they stop users from entering invalid information which would result in runtime errors.

9.2 Security

The system establishes security protocols which safeguard user information and block standard software vulnerabilities.

- **Password hashing**: Password hashing system stores user passwords in hashed format instead of using plain text. The system protects passwords against database exposure because they remain unreadable.
- **Parameterized SQL**: Parameterized SQL queries system requires all database queries to use parameters for their execution. This system protects against SQL injection attacks because it prevents user input from changing SQL command structure.
- **Role-based access**: Role-based access control system divides Admin permissions from Customer permissions. The system restricts administrative features to Admin accounts which include product management and order management and user management and discount management.

Purpose: The system uses these mechanisms to maintain user data confidentiality while blocking unauthorized access and common security threats.

9.3 Exception Handling

Exception handling system enables the application to function properly when unexpected errors occur.

- **try/catch blocks in repositories:** The system uses try/catch blocks in repositories to protect database operations because it enables error recovery from connection failures and query issues.
- **friendly UI error messages:** User-friendly error messages The system displays clear and simple messages to users which show what went wrong instead of showing technical error details.

- **rollback on checkout transaction failures:** transaction rollback system automatically restores database state to its previous condition when order processing fails at any point before inserting order items and updating stock. The system prevents partial data updates while preserving database consistency.

Purpose: Proper exception handling protects system stability because it stops data corruption while improving user experience through controlled failure handling.

10. Testing & Quality Assurance

This section describes the testing process which verified the application accuracy together with its user experience and system performance stability. The testing process included both functional tests and user-interface tests which assessed the main features of the system.

Testing was carried out using **manual test cases** to validate the main user flows, database accuracy, and error handling. The following table summarizes **20 key test cases** used during development.

10.1 Manual Test Cases

TC ID	Test Scenario	Preconditions	Steps	Expected Result	Status
TC-01	Admin Login - Valid	Admin account exists and active	Enter admin email & password, click Login	Admin Dashboard loads, no errors	Pass
TC-02	Admin Login - Invalid Password	Admin email exists	Enter correct email + wrong password	Error message shown, no navigation	Pass
TC-03	Customer Registration - New Account	Email not used before	Fill valid details and submit	Customer account created successfully	Pass
TC-04	Customer Registration - Duplicate Email	Email already exists	Register using existing email	Validation error shown (email already exists)	Pass
TC-05	Add Product (Admin)	Admin logged in	Manage Products -> Add -> Save valid product	Product saved to DB and appears in grid	Pass
TC-06	Update Product & Stock	Product exists	Edit price/stock, click Update	DB updated and UI refreshes with changes	Pass
TC-07	Delete Product (Admin)	Product exists	Select product -> Delete -> Confirm	Product removed (or marked inactive) and list updates	Pass
TC-08	Search Products (Keyword)	Products exist	Enter keyword in search box	Matching products only displayed	Pass

TC-09	Filter by Category	Categories and products exist	Select category filter	Only category products displayed	Pass
TC-10	Add to Cart	Customer logged in, product in stock	Open product -> Add to cart	Item added and cart count updates	Pass
TC-11	Update Cart Quantity	Cart has an item	Increase/decrease quantity	Subtotals and total recalculated correctly	Pass
TC-12	Checkout - Sufficient Stock	Cart has items and stock available	Proceed to checkout -> Confirm	Order + items inserted, stock reduced, confirmation shown	Pass
TC-13	Checkout - Insufficient Stock	Requested qty exceeds stock	Checkout with too many items	Validation error, no DB changes	Pass
TC-14	Generate Sales Report (PDF)	Admin logged in, orders exist	Sales Report -> Export PDF	PDF generated successfully with correct totals	Pass
TC-15	Database Offline Handling	MySQL service stopped	Launch app / perform DB action	Friendly error message, app does not crash	Pass
TC-16	Add Discount (Admin)	Admin logged in, product exists	Manage Discounts → Add → set % + date range → Save	Discounts were saved; product shows discounted price where applicable	Pass
TC-17	Discount Validation - Invalid Date Range	Admin logged in	Manage Discounts where StartDate > EndDate	Validation errors will be shown, discount not saved	Pass
TC-18	Customer View Discounted Price	Active discount exists for a product	Customer Dashboard → Browse products	Discount badge/price displayed, final price uses discount	Pass
TC-19	Checkout With Discount Applied	Customer logged in and cart contains discounted product	Add discounted product to cart → Checkout → Confirm	Order totals reflect discounted final price; stock reduces correctly	Pass

TC-20	Reviews - Add & View Review	Customer has a delivered/completed order (or allowed review state)	Order History → Select order → Add rating + comment → Submit	Review saved; appears under order/product review list; rating visible	Pass

10.2 Regression Strategy

- Re-run **TC-01** to **TC-12** after each major feature merge.
- Re-test checkout **(TC-12, TC-13)** after any schema or stock logic changes.
- Verify report accuracy **(TC-14)** after any pricing/discount updates.
- Re-test **discount functionality (TC-16 to TC-19)** after changes to product pricing or promotion
- Re-test **review system (TC-20)** after updates to order status logic or customer permissions

10.3 Cross-Device Testing (10+ Computers) & DPI/Scaling Fix

To validate portability and real-world compatibility, I tested the application on **around 10 different computers**. On a few devices the application did not display correctly at first due to **Windows DPI / scaling settings**.

Observed issue: On some PCs with 125% / 150% scaling, a few screens showed alignment issues (content overlap or misalignment).

Fix applied: I updated UI layout and scaling-related settings to reduce overlap issues. - When the screen scaling is **100%**, the UI does **not** overlap and displays correctly.

This testing process improved UI robustness and ensured the system can be demonstrated reliably on multiple machines.

11. Personal Reflection & Experience

11.1 Project Background & Motivation

I developed GreenLife Organic Store as my individual coursework to demonstrate practical knowledge of software engineering concepts taught in the module conducted by **Mr. Migara Alawatta**. The goal was to build a complete desktop system that resembles a real-world organic store environment, including user roles, inventory management, ordering, reporting, and secure login.

11.2 Development Approach

I built the system step-by-step using a structured workflow:

1. Designed the database entities and relationships (Users, Products, Categories, Orders, OrderItems, CartItems, Reviews).
2. Implemented core models in the Models layer.

3. Implemented repository classes to keep all database operations consistent.
4. Developed forms (UI) for Admin and Customer flows.
5. Added validations, security, and exception handling.
6. Added reporting and email notifications.
7. Performed manual testing and iterative improvements.

I used Git branching to isolate features and keep the main branch stable during development.

11.3 Challenges Faced and How I Solved Them

- **Database connectivity and query errors:** I moved all SQL logic into repositories and used parameterized queries to prevent injection and reduce mistakes.
- **Windows Forms UI complexity:** I organized forms by feature and reused common methods for validation and UI updates.
- **Transaction handling:** I learned how to use MySQL transactions so that order creation and stock updates happen safely.
- **SMTP email configuration:** I tested SMTP settings separately before integrating into the application
- DPI / Screen Scaling Compatibility Issue (Real-world testing)
  - Some screens displayed incorrectly with Windows DPI scaling at 125% or 150%.
  - I resolved this by updating form layout, anchoring/docking, and UI resizing.
  - **100% scaling**

Portability Requirement: To support demonstrations and learning, I prepared an additional **SQLite embedded portable build** that runs without MySQL installation.

11.4 What I Learned

This project strengthened my skills in:

- Layered architecture and repository design
- MySQL database design and transactions
- Windows Forms UI development
- Secure authentication and hashing
- Validation and exception handling
- System documentation and software engineering reporting

Overall, this coursework helped me understand how real applications are planned, built, tested, and documented in a professional way.

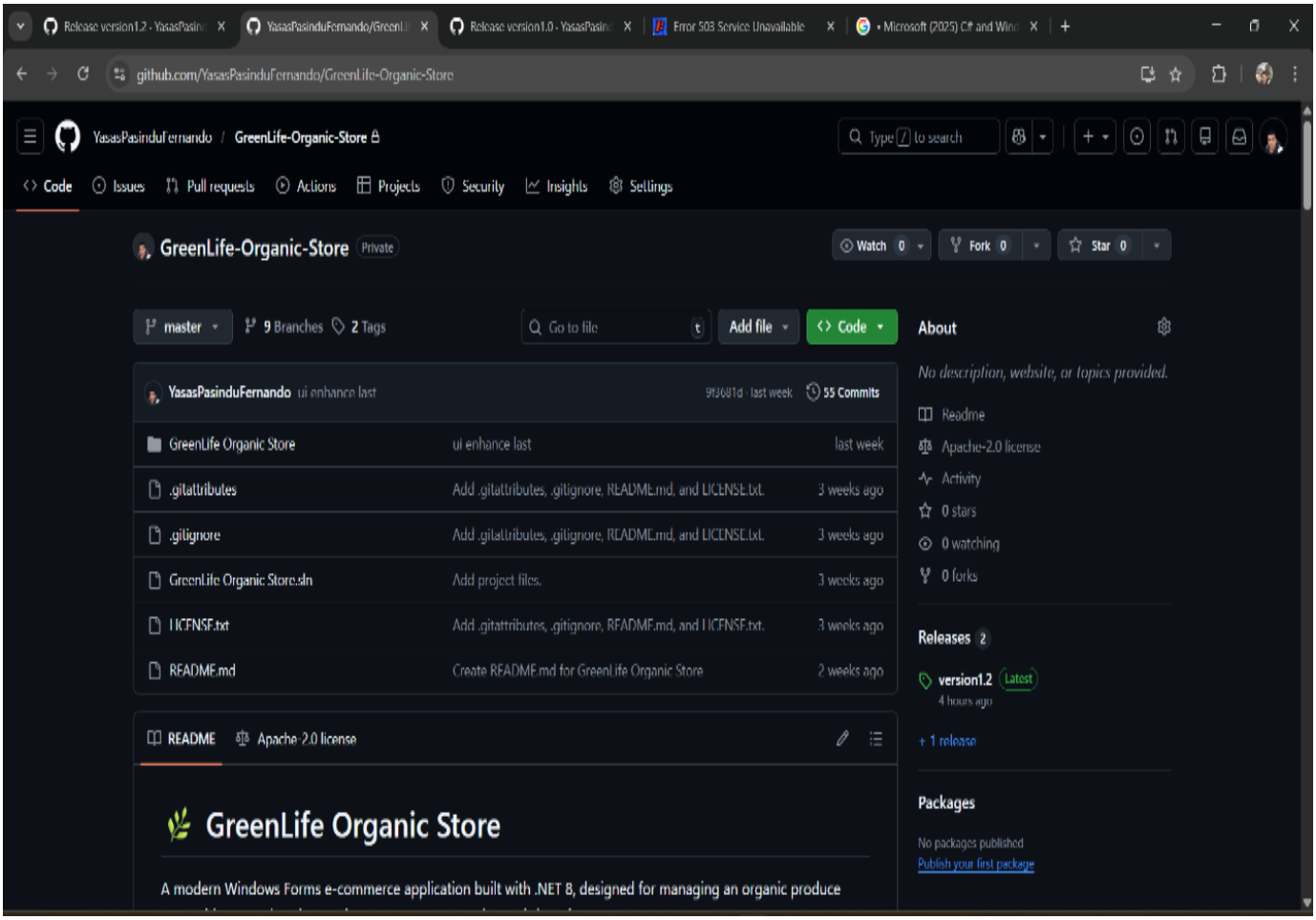
12. Version Control & Branching Strategy

I used Git as the primary version control system and developed the project in a structured way:

- Each major feature was implemented **step-by-step** and committed with meaningful messages.
- Development work was organized into **separate branches** and then merged into **master** after testing.
- The SQLite portable version was maintained in a separate branch:
  sql-lite-db branch (portable / embedded database)
- The **master branch** represents the main MySQL academic review version.

The project source code is maintained using Git and hosted on GitHub for **version tracking** and backup. The repository includes the full Visual Studio solution, **database script**, and a **README** file with setup instructions.

Repository: https://github.com/YasasPasinduFernando/GreenLife-Organic-Store.git



13. References

Microsoft (n.d.) *Microsoft .NET documentation*. Available at: https://learn.microsoft.com/ (Accessed: 05 January 2026).

MySQL (n.d.) *MySQL documentation: SQL syntax, constraints and indexing*. Available at: https://dev.mysql.com/doc/ (Accessed: 10 January 2026).

NuGet (n.d.) *NuGet package documentation for MySql.Data, FontAwesome.Sharp and PdfSharpCore*. Available at: https://www.nuget.org/ (Accessed: 12 January 2026).

Stack Overflow (n.d.) *Community discussions and solutions related to C# and MySQL development*. Available at: https://stackoverflow.com/ (Accessed: 10 January 2026).

W3Schools (n.d.) *SQL and C# tutorials and reference materials*. Available at: https://www.w3schools.com/ (Accessed: 16 January 2026).

GeeksforGeeks (n.d.) *Programming and computer science concepts related to search algorithms and data handling*. Available at: https://www.geeksforgeeks.org/ (Accessed: 23 January 2026).

YouTube (n.d.) *Video tutorials on Windows Forms UI design and MySQL connectivity*. Available at: https://www.youtube.com/ (Accessed: 09 January 2026).

14. Declaration of Originality

All the code and documentation in this project have been created by me, based on my knowledge of C#, .NET, and software engineering. Libraries used are official NuGet packages.

15. Academic Supervision & Module Information

This project was developed as part of the Application Development academic module under the supervision of **Mr Migara Alawatta**. The system design, implementation, testing, and documentation were completed in accordance with the module requirements and academic assessment guidelines.

16. Conclusion

The GreenLife Organic Store system successfully demonstrates the design and implementation of a complete desktop-based e-commerce solution using modern software engineering practices. The application provides comprehensive functionality for administrators and customers, including secure authentication, product catalog management, shopping cart and checkout, order processing with transaction support, stock management, review and rating system, reporting, and email notifications.

The system was tested on multiple computers to ensure portability, DPI compatibility, and runtime stability. Additional engineering effort was invested to support a portable executable version with an embedded SQLite database for environments where MySQL is not available.

This project enhanced my understanding of:

- Layered architecture and repository design
- Secure authentication and password hashing
- Database transactions and consistency handling
- Windows Forms UI development
- Validation and exception handling
- Software documentation and reporting standards

In conclusion, I would say that this coursework gave me hands-on experience in creating a full desktop application, and I think I improved my knowledge of C#, working with a database, creating a user interface, and software testing. I also learned that classroom ideas about software engineering could be applied to a real project.

The project helped me to improve my software engineering skills because I learned to use its principles in actual work projects. The project helped me to improve my software engineering skills because I learned to use its principles in actual work projects.

Appendix A: Key Implementation Examples (Code-Level Evidence)

This section will highlight some of the important code snippets from the application that were created to implement some of the main features of the application.

A.1: High-DPI Application Startup (Program.cs)

In the startup file (Program.cs), DPI awareness is enabled to prevent issues with the layout of the application when Windows display scaling is enabled. The startup code also ensures that the necessary directories, such as the directory containing the images, are created. Optional services, such as email configuration, are also checked but in a way that will not prevent the application from running if they are not available.

```
[STAThread]
static void Main()
{
    // DPI awareness for multi-monitor scaling
    Application.SetHighDpiMode(HighDpiMode.PerMonitorV2);

    Application.EnableVisualStyles();
    Application.SetCompatibleTextRenderingDefault(false);

    // Ensure image directory exists (non-blocking)
    try
    {
        ImageStore.EnsureImagesDirectoryExists();
    }
    catch
    {
```

```
        // Non-fatal: application can still run
    }

    // Optional email service validation (non-blocking)
    try
    {
        EmailConfigValidator.LogConfigurationStatus();
        EmailService.TestConnection();   }
    catch
    {
        // Non-fatal: continue without email capability
    }

    Application.Run(new LoginForm());
}
```

Rationale:

This start-up strategy was selected in order to make the application more stable and user-friendly on different computers. The DPI setting helps eliminate problems with the layout when the Windows display scaling is set to 125% or 150%. It also checks optional services, such as email, without the application failing if the services are not available.

A.2: Transactional Order Creation (OrderRepository.cs)

Order processing is implemented using a database transaction to ensure **atomicity**: the order header, order items, and stock updates are committed together or rolled back together if an error occurs.

```
public static int CreateOrder(Order order)
{
    MySqlConnection? connection = null;
    MySqlTransaction? transaction = null;

    try
    {
        connection = DatabaseConnection.GetConnection();
        connection.Open();
        transaction = connection.BeginTransaction();

        // 1) Insert order header and obtain ID
        string orderQuery = @"INSERT INTO Orders
        (OrderNumber, CustomerID, CustomerName, CustomerPhone, CustomerEmail,
              OrderDate, TotalAmount, Status, ShippingAddress, Notes)
```

```
    VALUES
        (@OrderNumber, @CustomerID, @CustomerName, @CustomerPhone, @CustomerEmail,
    @OrderDate, @TotalAmount, @Status, @ShippingAddress, @Notes);
    SELECT LAST_INSERT_ID();";

    int orderId;
    using (var cmd = new MySqlCommand(orderQuery, connection, transaction))
    {
        cmd.Parameters.AddWithValue("@OrderNumber", order.OrderNumber);
        cmd.Parameters.AddWithValue("@CustomerID", order.CustomerID);
        cmd.Parameters.AddWithValue("@CustomerName", order.CustomerName);
        cmd.Parameters.AddWithValue("@CustomerPhone", order.CustomerPhone);
        cmd.Parameters.AddWithValue("@CustomerEmail", order.CustomerEmail);
        cmd.Parameters.AddWithValue("@OrderDate", order.OrderDate);
        cmd.Parameters.AddWithValue("@TotalAmount", order.TotalAmount);
        cmd.Parameters.AddWithValue("@Status", order.Status.ToString());
        cmd.Parameters.AddWithValue("@ShippingAddress", order.ShippingAddress);
        cmd.Parameters.AddWithValue("@Notes", (object?)order.Notes ?? DBNull.Value);
        orderId = Convert.ToInt32(cmd.ExecuteScalar());
    }

    // 2) Insert order items and reduce stock
    string itemQuery = @"INSERT INTO OrderItems
    (OrderID, ProductID, ProductName, Quantity, UnitPrice, Subtotal)
    VALUES
        (@OrderID, @ProductID, @ProductName, @Quantity, @UnitPrice, @Subtotal);";

    foreach (var item in order.Items)
    {
        using (var cmd = new MySqlCommand(itemQuery, connection, transaction))
        {
            cmd.Parameters.AddWithValue("@OrderID", orderId);
            cmd.Parameters.AddWithValue("@ProductID", item.ProductID);
            cmd.Parameters.AddWithValue("@ProductName", item.ProductName);
            cmd.Parameters.AddWithValue("@Quantity", item.Quantity);
            cmd.Parameters.AddWithValue("@UnitPrice", item.UnitPrice);
            cmd.Parameters.AddWithValue("@Subtotal", item.Subtotal);
            cmd.ExecuteNonQuery();
        }

        using (var cmd = new MySqlCommand(
            "UPDATE Products SET Stock = Stock - @Quantity WHERE ID = @ProductID",
            connection, transaction))
        {
            cmd.Parameters.AddWithValue("@Quantity", item.Quantity);
            cmd.Parameters.AddWithValue("@ProductID", item.ProductID);
            cmd.ExecuteNonQuery();
        }
```

```
        }

        transaction.Commit();
        return orderId;
    }
    catch
    {
        transaction?.Rollback();
        throw;
    }
    finally
    {
        connection?.Close();
        connection?.Dispose();
    }
}
```

Rationale:

Transaction control ensures the database is never left in a partial state (e.g., order created but stock not updated). This is essential for inventory accuracy and prevents overselling.

A.3: Product Search Using Linear (Sequential) Search (ProductRepository.cs)

A linear search algorithm is implemented to meet coursework requirements for fundamental algorithm application. The search is case-insensitive and checks both product name and description.

```
public static List<Product> SearchProducts(string searchTerm)
{
    var allProducts = GetAllProducts();
    var results = new List<Product>();

    foreach (var product in allProducts)
    {
        if (product.ProductName.IndexOf(searchTerm,
                StringComparison.OrdinalIgnoreCase) >= 0)
        {
            results.Add(product);
            continue;
        }

        if (product.Description != null &&
                product.Description.IndexOf(searchTerm,
```

```
                    StringComparison.OrdinalIgnoreCase) >= 0)
        {
            results.Add(product);
        }
    }

    return results;
}
```

Rationale:

Linear search is simple and effective for small-to-medium product catalogs and provides clear algorithmic evidence. For large datasets, indexed database queries or full-text search would be more scalable.

A.4: Email Service with SMTP and Mock Mode (EmailService.cs)

The system supports transactional emails (order confirmation, password reset, status updates). For academic submission and testing, mock mode can be enabled to avoid external dependency failures.

```
private const string SMTP_SERVER = "smtp.gmail.com";
private const int SMTP_PORT = 587;
private const string SENDER_EMAIL = "YOUR_SENDER_EMAIL";
private const string SENDER_PASSWORD = "YOUR_APP_PASSWORD";
private const bool USE_MOCK_MODE = true;

public static bool SendEmail(string toEmail, string subject, string body, bool isHtml = true)
{
    if (USE_MOCK_MODE)
    {
        // Mock: do not send externally (safe for demos)
        return true;
    }

    using var smtp = new SmtpClient(SMTP_SERVER, SMTP_PORT)
{
    EnableSsl = true,
    Credentials = new NetworkCredential(SENDER_EMAIL, SENDER_PASSWORD),
    DeliveryMethod = SmtpDeliveryMethod.Network,
    Timeout = 10000
};

    using var msg = new MailMessage();
```

```
    msg.From = new MailAddress(SENDER_EMAIL, "GreenLife Organic Store");
    msg.To.Add(toEmail);
    msg.Subject = subject;
    msg.Body = body;
    msg.IsBodyHtml = isHtml;

    smtp.Send(msg);
    return true;
}
```

Rationale:

Mock mode supports repeatable testing without requiring internet connectivity or live credentials. When real SMTP is enabled, TLS is used (EnableSsl = true) to protect email transmission.

A.5: Portable Image Storage Using Relative Paths (ImageStore.cs)

Images are stored in an Images/ folder and the database stores **relative paths**. This improves portability (e.g., when running as a portable application on different machines).

```
private const string ImagesFolderName = "Images";

public static void EnsureImagesDirectoryExists()
{
    Directory.CreateDirectory(GetImagesDirectory());
}

public static string SaveImageFile(string sourceFilePath)
{
    EnsureImagesDirectoryExists();

    var fileName = Path.GetFileName(sourceFilePath);
    var destDir = GetImagesDirectory();
    var destPath = Path.Combine(destDir, fileName);

    if (File.Exists(destPath))
    {
        var unique = Guid.NewGuid().ToString().Split('-')[0];
        var name = Path.GetFileNameWithoutExtension(fileName);
        var ext = Path.GetExtension(fileName);
        destPath = Path.Combine(destDir, name + "_" + unique + ext);
        fileName = Path.GetFileName(destPath);
    }
```

```
    File.Copy(sourceFilePath, destPath);

    // Store DB-friendly relative path
    return ImagesFolderName + "/" + fileName;
}

public static string GetFullPath(string storedPath)
{
    if (Path.IsPathRooted(storedPath)) return storedPath;
    var normalized = storedPath.Replace('/', Path.DirectorySeparatorChar);
    return Path.Combine(GetProjectRoot(), normalized);
}
```

Rationale:

Relative paths (e.g., Images/product_abc123.jpg) remain valid even if the application is moved to another folder or another machine, which supports the project's portable deployment goal

A.6: Secure Login with Password Hashing and Role-Based Navigation

Feature: Authentication + user role routing (Admin / Customer)

File Paths

- Database/UserRepository.cs
- Utilities/PasswordHasher.cs
- Forms/LoginForm.cs

Method Signatures

```
public static User? AuthenticateUser(string email, string password)
public static string HashPassword(string password)
public static bool VerifyPassword(string password, string hash)
private void PerformLogin()
```

Explanation:

The system uses distinct functions to manage its login operation. The LoginForm collects credentials and triggers the login action. The UserRepository.AuthenticateUser() method retrieves the matching active user record using a parameterized query and then verifies the password by comparing the stored hash with the hash of the entered password. The application uses UserType (Admin/Customer) to direct users to their appropriate dashboard screen after they complete their authentication process. The system prevents customers from using admin-exclusive functions through UI-based access restrictions.

Rationale:

The system gains better security and maintenance capabilities through its design which separates authentication functions into the repository and its hashing functions into a utility class instead of uniting them within the form. The system keeps all passwords in encrypted formats which protects against unauthorized access while parameterized queries block SQL injection attacks. The system requires SHA256 hashing for its coursework needs but production systems should implement salted adaptive hashing methods like PBKDF2 or BCrypt.

A.7: Input Validation Utilities and Form Validation

Feature: Prevent invalid input and protect database consistency

File Paths

- Forms/CustomerRegistrationForm.cs
- Forms/CheckoutForm.cs
- Forms/AddEditDiscountForm.cs

Method Signatures

```
private bool ValidateInput()
private bool IsValidEmail(string email)
private void BtnPlaceOrder_Click(object sender, EventArgs e)
private void BtnSave_Click(object sender, EventArgs e)
```

Explanation:

The system performs its validation process before it executes any database operation to minimize runtime errors while ensuring database integrity. The registration and login forms perform validation checks for all required fields including email format and age limits and password strength requirements and confirmation matching. The checkout process checks customer information by verifying phone number format and it stops customers from creating orders that are either incomplete or invalid. The discount management system checks discount percentage limits between 0 and 100 and it ensures that StartDate comes before EndDate and it prevents discounts that contain logical errors. Message boxes which are designed for user convenience show users the exact parts that they must fix.

Rationale:

The initial validation process enhances user experience while it decreases database mistakes that occur from invalid entries and complete data loss and stock inventory decrements. The system provides users with smooth operation because it delivers instant results instead of waiting for errors to occur during database operations.

A.8: Product CRUD Using Parameterized Queries

Feature: Admin product management (Add / Update / Delete / Activate)

File Path

- Database/ProductRepository.cs

Method Signatures

```
public static int CreateProduct(Product product)
public static bool UpdateProduct(Product product)
public static bool DeleteProduct(int productId)
```

Explanation:

The ProductRepository uses parameterized SQL commands to implement product management. The system creates operations to insert products into the database while using LAST_INSERT_ID() to obtain the generated product ID. The update operations enable product field changes while allowing users to update additional fields which include images and discount price information. The system supports two methods for handling product records through deletion which allows products to be either completely removed or marked as inactive. The system tracks stock thresholds to provide notifications when product inventory reaches low levels.

Rationale:

The use of parameterized queries together with repository classes enables centralized control of database logic which protects against SQL injection attacks. The project achieves better maintainability through SQL separation from Windows Forms which also facilitates easier project comprehension and evaluation.

A.9: Discount Management with Date Validation and Product Price

Feature: Admin discount CRUD + automatic pricing updates

File Paths

- Models/Discount.cs
- Database/DiscountRepository.cs
- Forms/AddEditDiscountForm.cs

Method Signatures

```
public bool IsValid()
public string GetStatusText()
public static Discount? GetActiveDiscountForProduct(int productId)
public static void SyncActiveDiscountForProduct(int productId)
```

Explanation:

The Discounts table stores discounts which include keyfields that define percent discount and date range and active status of the discount. The system uses active discount determination through IsActive flag assessment and valid date range assessment which includes StartDate and EndDate. The repository updates Products.DiscountPrice field after a discount creation or update to enable product browsing and checkout which shows correct discounted price without using excessive join queries. The UI validation system prevents users from entering invalid values which include negative discounts and discounts that exceed 100 and date ranges that do not meet requirements.

Rationale:

The system uses this approach to achieve fast and uniform pricing throughout the entire system. The product table receives discounted prices which allows customer screens to compute totals immediately while maintaining discount records in a separate table.

A.10: Price Calculation with Discount Logic (Product Model)

Feature: Consistent price display and discount calculations

File Path

- Models/Product.cs

Method Signatures

```
public bool HasDiscount()
public decimal GetFinalPrice()
public int GetDiscountPercent()
public string GetStockStatus()
```

Explanation:

The Product model includes helper methods that maintain consistent pricing and stock management operations throughout all application interfaces. The function GetFinalPrice() returns the discounted price when a discount exists but it returns the standard price when no discount exists. The function HasDiscount() verifies that a discount only becomes active when DiscountPrice exists and that value remains below the standard price. The function GetDiscountPercent() determines the discount percentage which users will see on UI badges. GetStockStatus() delivers user-friendly stock status indicators which show In Stock status and Low Stock status and Out of Stock status.

Rationale:

The implementation of business rules within the model framework eliminates the need for multiple logic implementations across different forms while it establishes identical discount and total calculation methods for browsing and cart and checkout processes.

Appendix B: Future Enhancements and Roadmap

The GreenLife Organic Store application delivers an entire e-commerce management system which operates entirely on desktop computers. The system provides multiple functions which include product management and order processing and discount handling together with reporting and customer interaction capabilities. The system fulfills essential requirements for course completion but developers discovered multiple system enhancements which would boost system performance and operational intelligence and protection measures and ability to grow in the future. The proposed system enhancements represent practical development paths which extend the existing architecture that combines Windows Forms and MySQL and the Repository Pattern. The system exists as a complete design which permits development of new systems through existing framework components and existing design elements which include current models and repositories and utility functions.

B.1: Online Payment Integration

Description:
Current system operations treat orders as offline payments. The system will enable digital payment processing through an online payment gateway which will launch in a future software version.

Benefits: - Enables real e-commerce transactions and Reduces manual payment handling Improves customer convenience.

Possible Implementation Approach: - Add a **Payment** table to store transaction details. Integrate a third-party payment API (such as Stripe or PayPal) and Update the **Checkout** process to confirm payment before marking orders as “Completed”

B.2: Mobile Application Version

Description:
The current system operates on desktop computers. The mobile application enables customers to search for products and complete their purchases while tracking their order status. Better accessibility for customers The system enables users to access its functions through devices that do not need desktop computers. The development team will create a mobile interface by using either Flutter or React Native technology The existing system will provide a basic REST API which mobile devices will use to connect with the system The same MySQL database will be utilized throughout the entire project.

Benefits: - Better accessibility for customers and Expands system usability beyond desktop users

Possible Implementation Approach: - Develop a mobile frontend using Flutter or React Native. Expose a simple REST API from the existing system for mobile communication. Reuse the same MySQL database.

B.3: Enhanced Security Features

Description:
The system protects passwords through hashing while it prevents SQL injection attacks. The system requires additional development for security improvements. The system should implement Two-Factor Authentication (2FA) which uses email codes. The system should automatically lock user accounts after several unsuccessful login attempts. The system should enable users to view their login activities. The system needs to implement a stronger password policy which requires users to create their passwords.

Possible Improvements: - The system requires two-factor authentication through email codes and it locks accounts after multiple unsuccessful login attempts. The system provides users with a record of their login activity to enforce stronger password requirements.

Benefits: - Reduces risk of unauthorized access Aligns the system with modern security practices

B.4: Advanced Business Reports

Description:
The system already generates PDF & CSV sales reports. The upcoming versions of the system will introduce higher level data analysis capabilities.

Possible Enhancements: The following elements make up the system which includes revenue comparison charts for monthly and annual periods and product sales data of top-selling items and customer buying patterns reports and ability to create export reports in How **Charted Accountant** does or manage Accounts using CSV format. And Aswell integrate **AI** features to analyse and make predictions to save money and grow the product.

Benefits: - Helps administrators make better business decisions and Provides deeper insight into store performance

B.5: Automated Database Backup System

Description:
Database backups currently require manual execution. The scheduled backup system creates a safeguard which protects data from being lost.

Possible Implementation: - The system will create a background utility which will export the MySQL database every day. The system will create timestamped backup files. The system will establish a backup log table which will monitor backup operations to determine their success rate. The system protects business data through its backup system which prevents accidental data loss. The system uses backup systems to enhance its overall reliability.

Benefits: - Protects business data from accidental loss and Improves system reliability

B.6: Stock Prediction and Demand Forecasting

Description:
The system tracks stock levels but does not predict future demand.

Future Improvement: - The system needs to develop a complete inventory management system which requires three main functions to operate effectively. The first function needs to analyze past order history in order to project future product demand. The system requires a second function which will determine appropriate reorder quantities for products that experience high sales volume. The system needs to show "Predicted Low Stock Soon" alerts to users who access the Admin Dashboard.

Benefits: - Prevents the need for stock-outs, reduces overstocking, and gives support to smarter inventory planning.

B.7: Improved Discount System

Description:
Currently, discounts are product-specific and time-limited. The system could be expanded to support:

- Category-wide discounts
- Coupon codes entered at checkout
- Bulk purchase discounts (e.g., buy 3 get 10% off)

Benefits: - More flexible marketing strategies - Encourages higher customer spending

B.8: Review Moderation System

Description:
Customers can submit reviews, but there is no moderation workflow.

Future Addition: - The system requires admin approval to show reviews. Users have the capacity to conceal content that they find unsuitable. Admins have the ability to reply to customer reviews.

Benefits: - Keep pages for products professional, to avoid misuse of the review function.

B. 9: Image Optimization and Multiple Product Images

Description:
Products currently store a single image. Future improvements could include:

- Support for multiple images per product
- Automatic resizing and compression during upload
- Thumbnail generation for faster loading

Benefits: - Improves product presentation - Reduces storage space and loading time

B.10: Audit Logging System

Description:
An audit log could record important system actions.

Examples of Logged Actions: - Product updates , Discount changes , Order status changes , Admin account actions

Benefits: - Provides traceability and Helps diagnose issues Aswell helps for Improves accountability

B.11: Stock Movement History

Description:
Currently only total stock is tracked. A future version could track detailed stock movements.

Examples: - Stock added (restock) - Stock reduced (orders) - Manual adjustments

Benefits: - Better inventory transparency and Easier to detect stock errors

B.12: Customer Loyalty Program

Description:
A points based loyalty system could reward repeat customers.

Possible Features: - Earn points for every purchase - Redeem points for discounts - Customer tiers (**Silver**, **Gold**, **Platinum**)

Benefits: - Encourages repeat purchases and Increases customer engagement

Conclusion

The GreenLife Organic Store system will develop into an advanced retail management platform through its upcoming improvements. The existing system design, which uses layered architecture and repository pattern, allows for system enhancements to proceed without needing complete system reconstruction.

The roadmap establishes its three main objectives by selecting business value, system security, data protection, and customer experience as essential components of operational software systems.

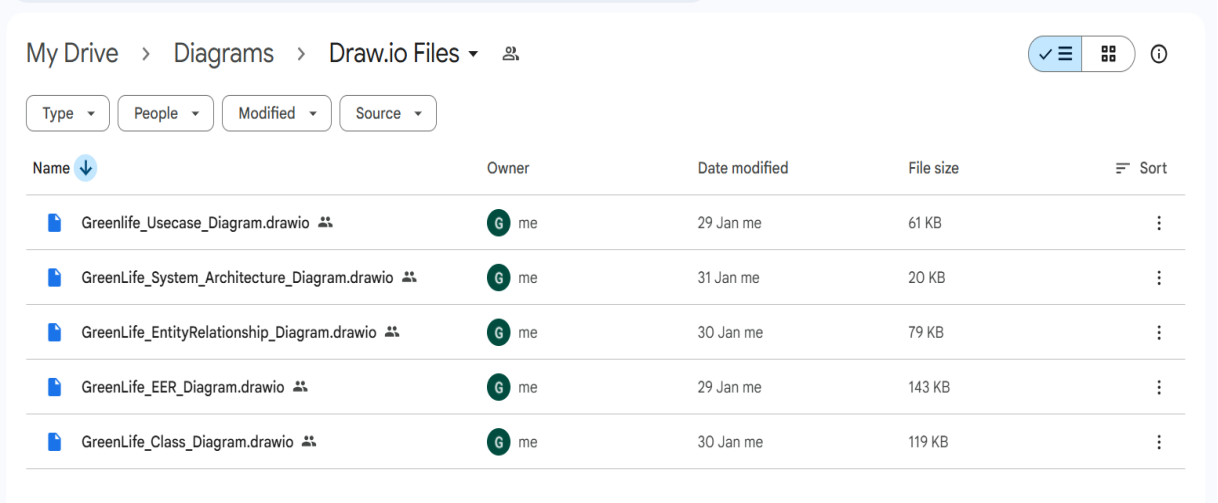
Appendix C: System Diagrams

This appendix contains the core UML and system design artefacts that support the analysis, design, and implementation of the GreenLife Organic Store application. The diagrams included here provide visual evidence of system structure, behaviour, and scope, while keeping the main body of the report focused on explanation and evaluation.

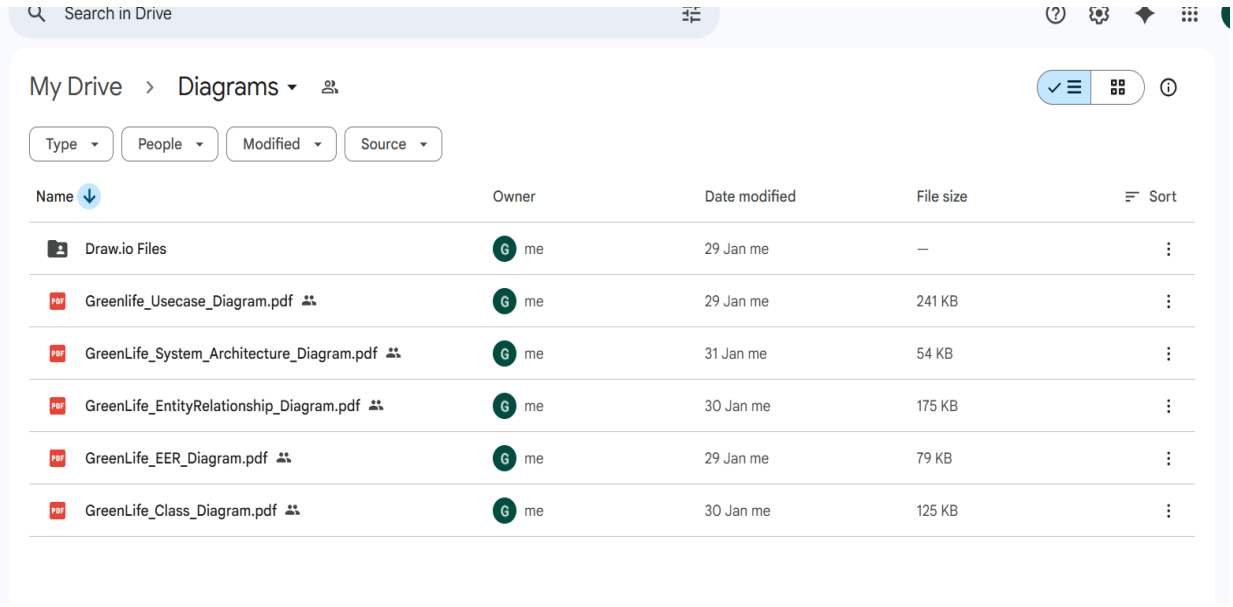
Note: Due to their level of **detail**, some diagrams are **larger** than standard **A4 page size** and may appear reduced in this document. **Full-resolution** versions of all diagrams have been **uploaded** separately and can be accessed via the provided links for **clearer** viewing and download. (pdfs & draw.io files included)

Link: - https://drive.google.com/drive/folders/1w6NCb2lghGpLLgvdA4OmszW6-aVv6KA8

Draw.io Files: -



Full Resolution Pdf Diagrams: -

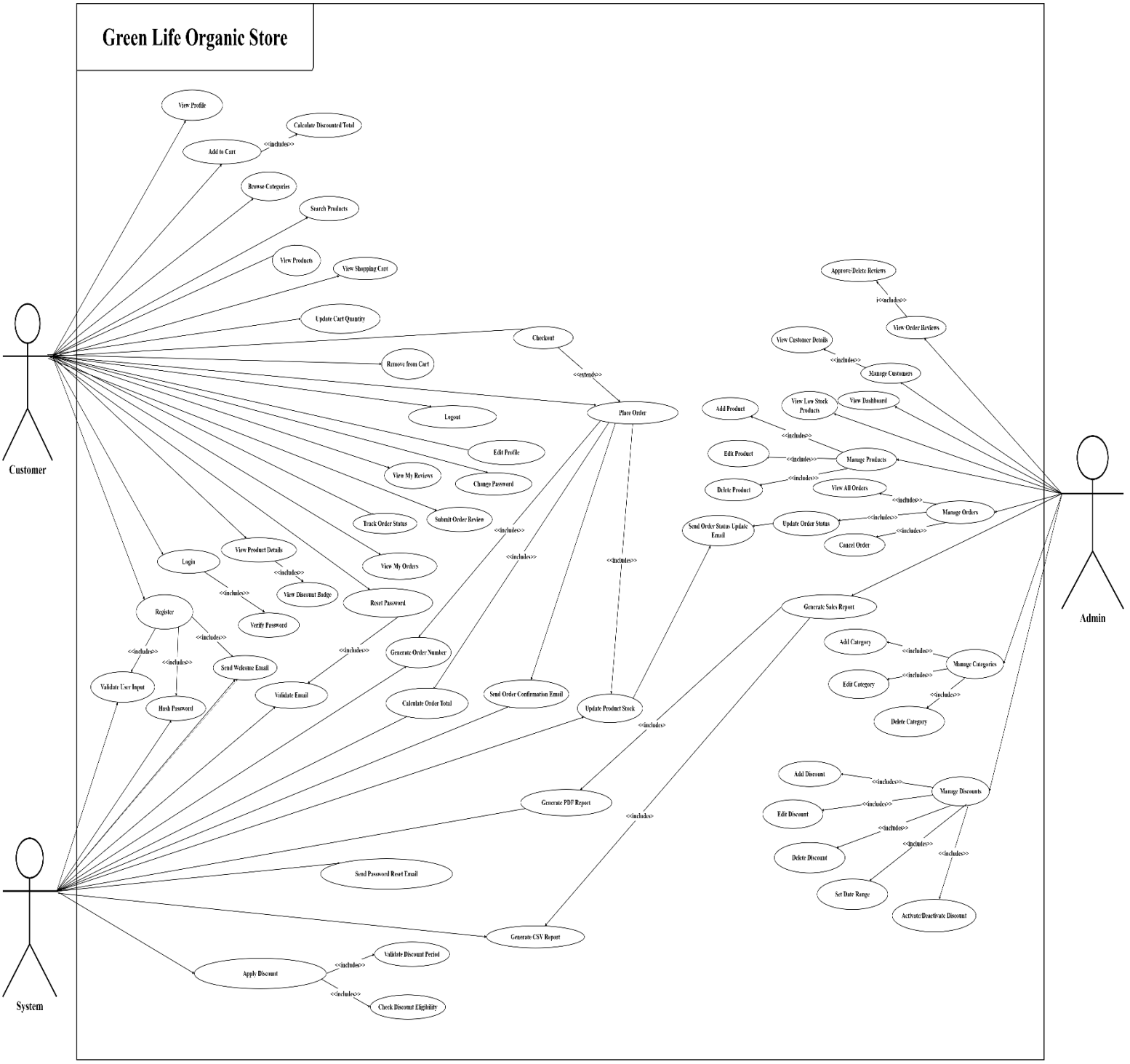


C.1 Use Case Diagram

The Use Case Diagram represents the functional scope of the system from the perspective of external actors. It identifies the main interactions between users and the system and clearly distinguishes responsibilities between different user roles.

Referenced in Section 6: Implemented Features.

Download Link: - https://drive.google.com/file/d/17vvQxuk0CUuh_STMIDSzQ9t3fF4Uj3NX/view?usp=drivesdk

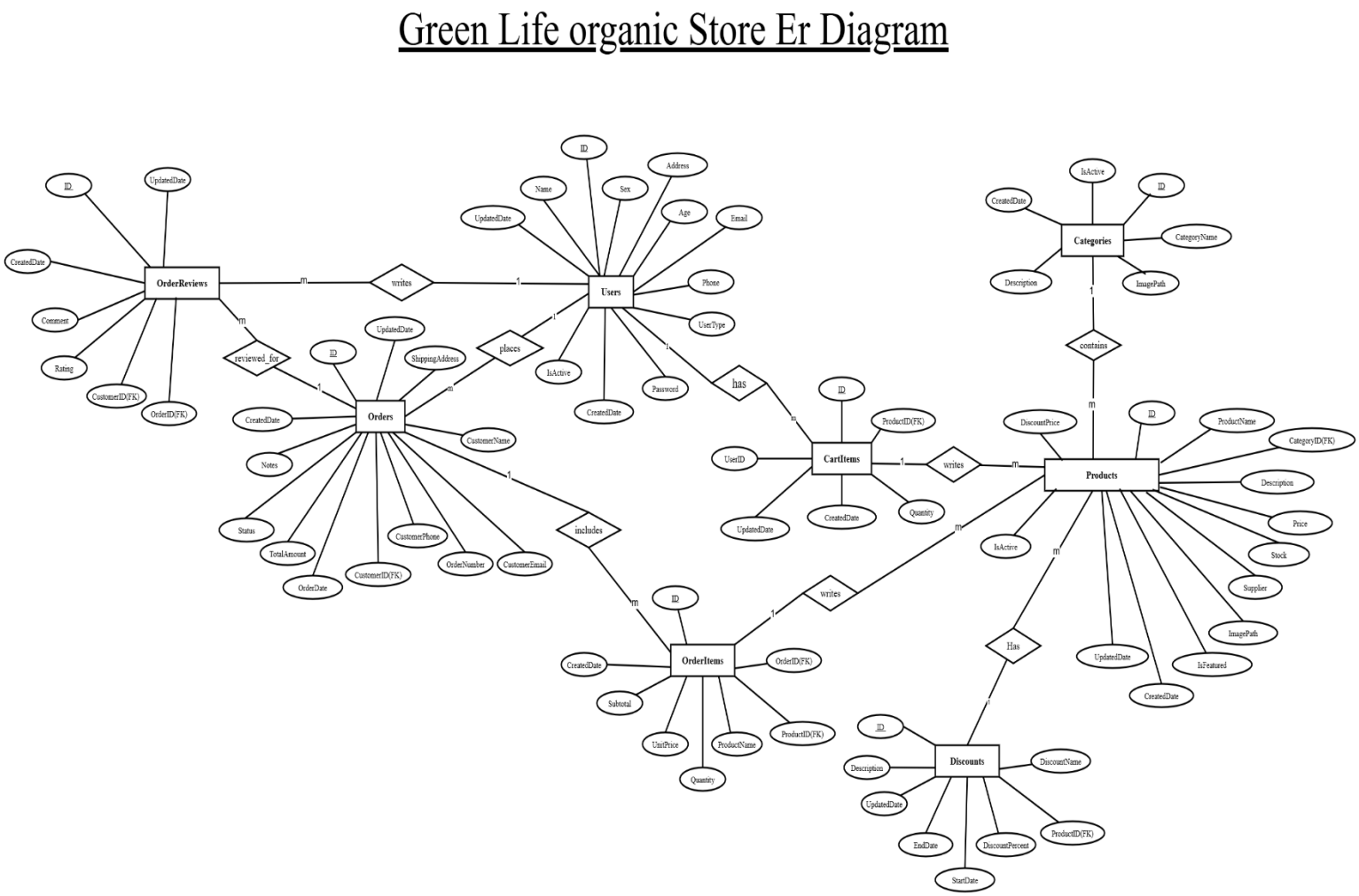


C.2.1 Entity Relationship (ER) Diagram

This diagram illustrates the complete database design of the system, including all major entities.

Referenced in Section 4: Database Design (ER Summary).

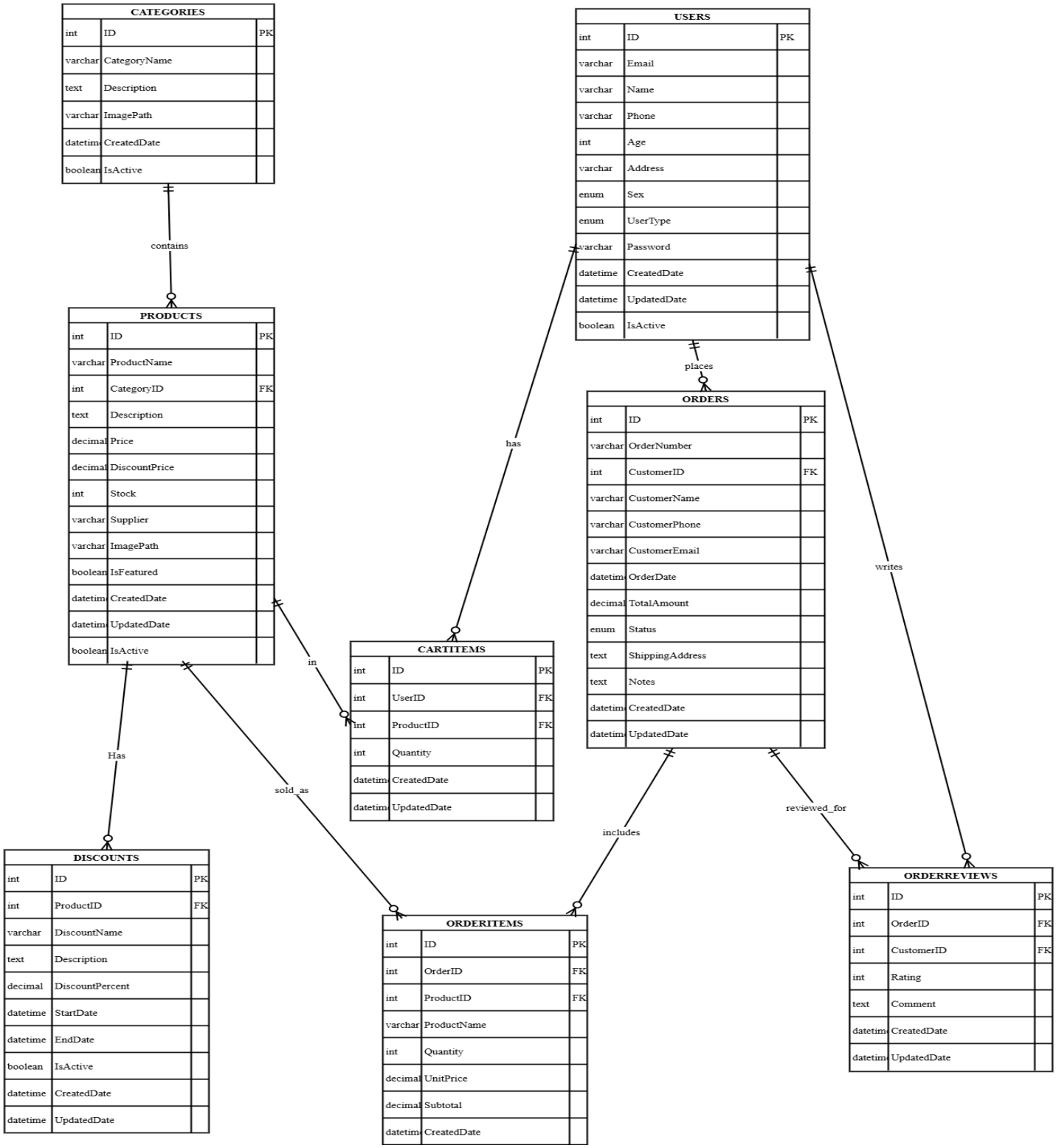
Download Link: -
https://drive.google.com/file/d/1t3spJFzt9eCH7p17y92JvQsofn6yULs3/view?usp=sharing



C.2.2 Entity Relationship (EER) Diagram

Download Link: -
https://drive.google.com/file/d/1Pgd3IoSqx0wHx8pVhs17QEbk_u3cCROS/view?usp=drives
dk

Green Life organic Store EER Diagram

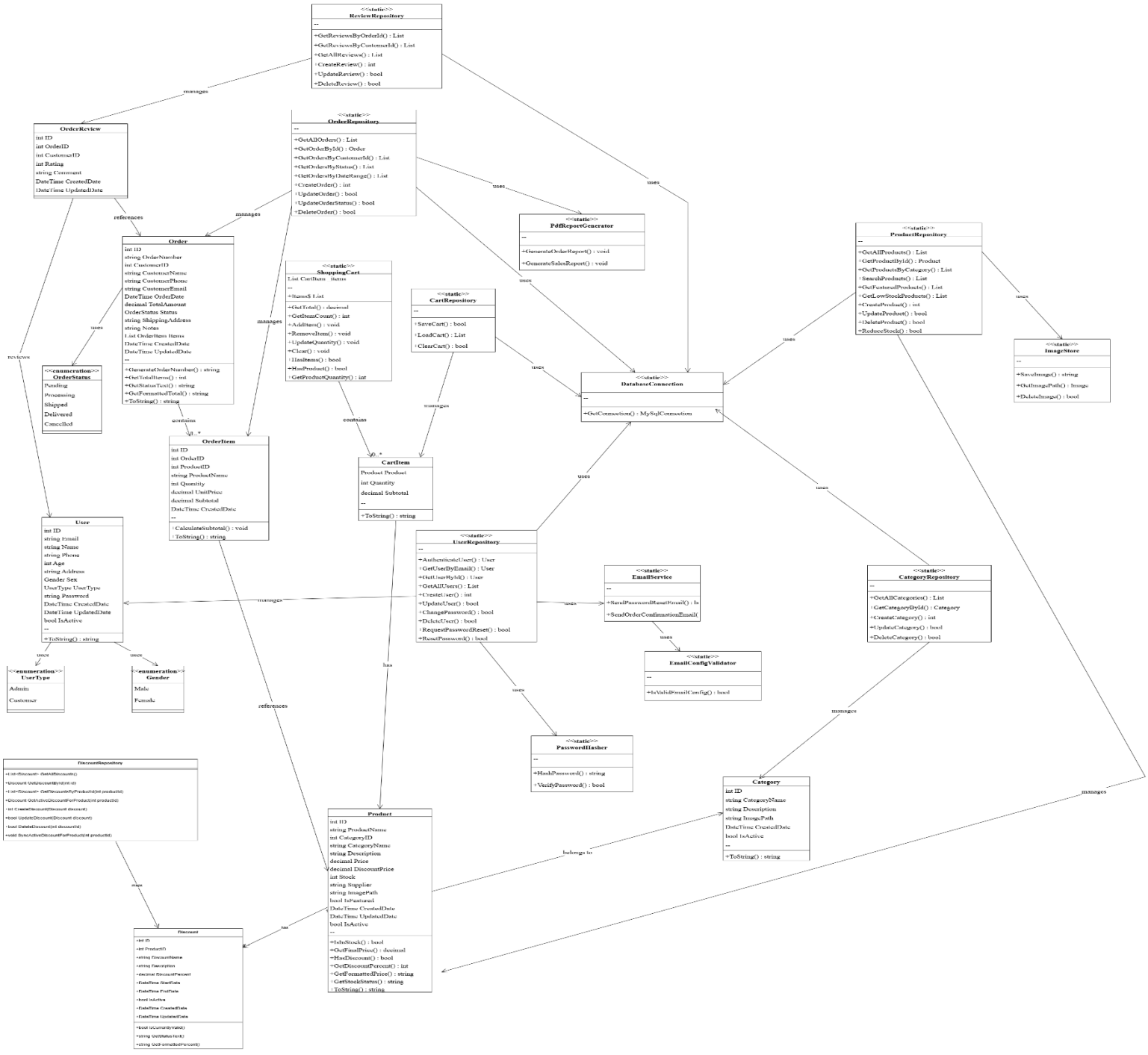


C.3 Class Diagram

The Class Diagram represents the static structure of the application and shows how the core classes are organised and related. It reflects the actual implementation used in the system.

Referenced in Sections 2 and 5: Software Architecture and Class Descriptions.

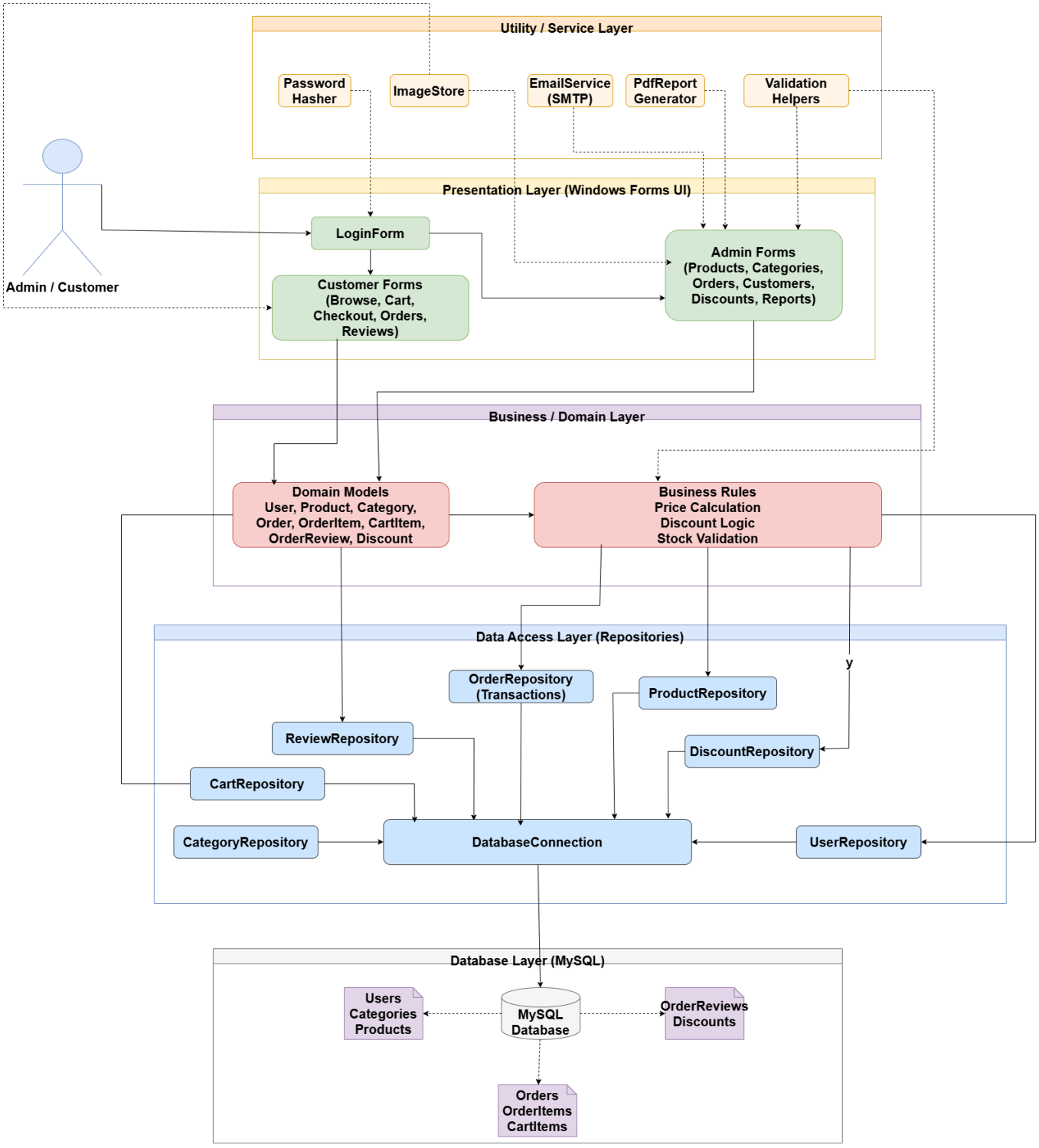
Download link: -
https://drive.google.com/file/d/19FnL4jt4tje_NijvlaGEDcao7UoLns71/view?usp=drivesdk



C.4 System Architecture Diagram

The System Architecture Diagram displays the complete application structure and demonstrates the interactions between its primary components. The system uses a real layered architecture pattern that divides its user interface and business logic and database access components for improved system maintenance and organizational structure..

Download link: - https://drive.google.com/file/d/1mN2DWL7Y3jmkd4a5i-9iGUac0lqMxwTc/view?usp=sharing

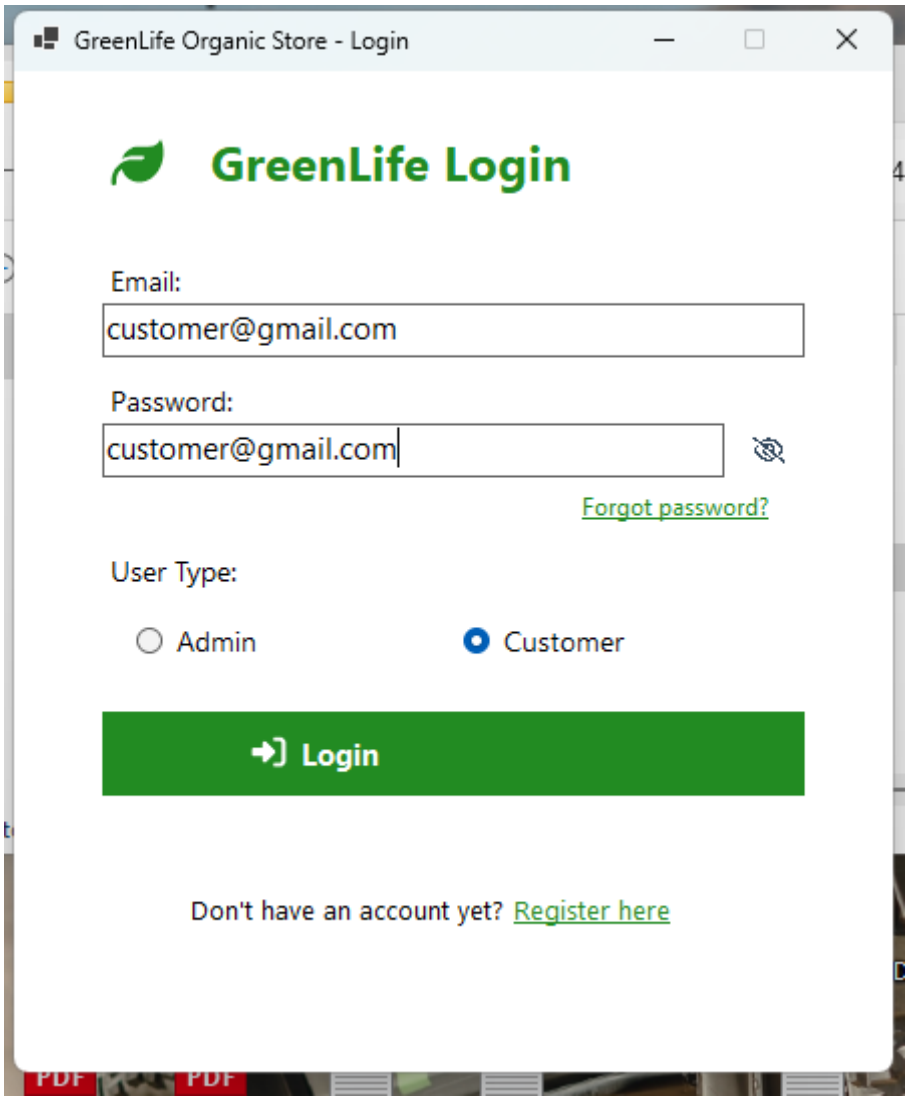


Appendix D: User Interface Screenshots

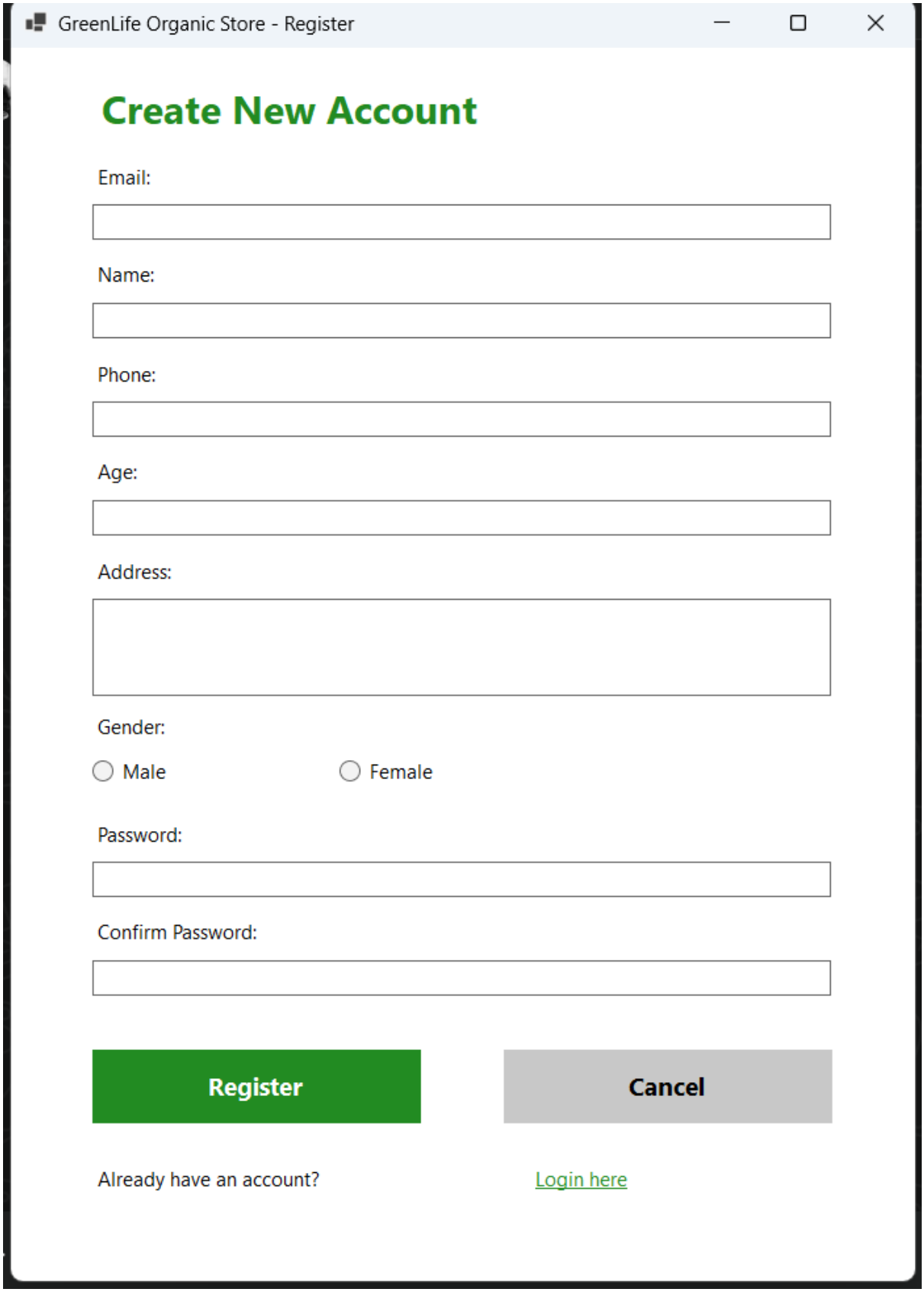
This section contains representative screenshots of the application user interface. These visuals are provided for demonstration and validation purposes only and are not intended to replace functional descriptions.

Typical screens include:

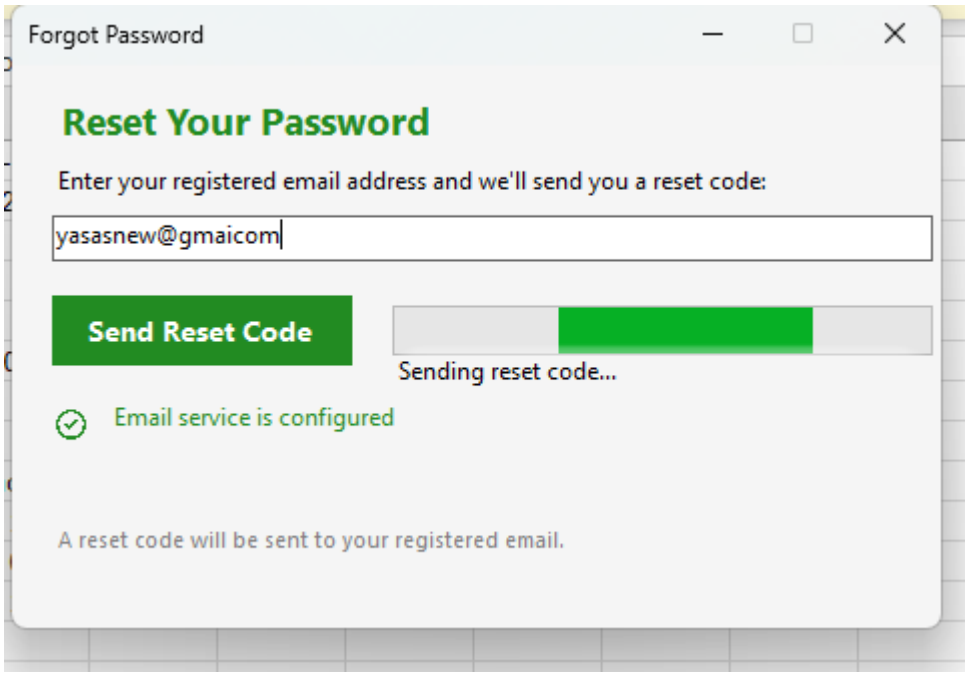
D.1 Login Screen

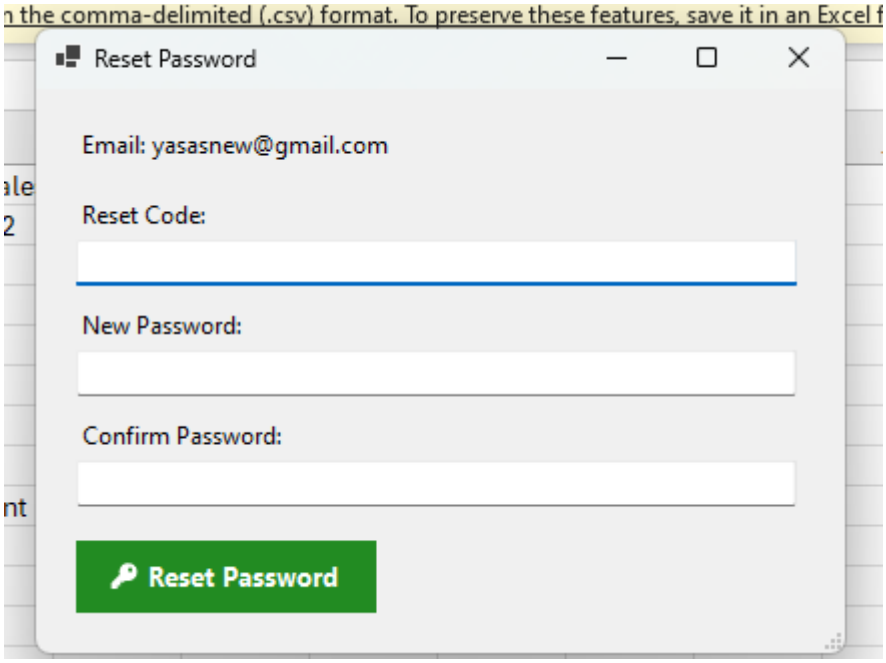


D.2 Register Screen

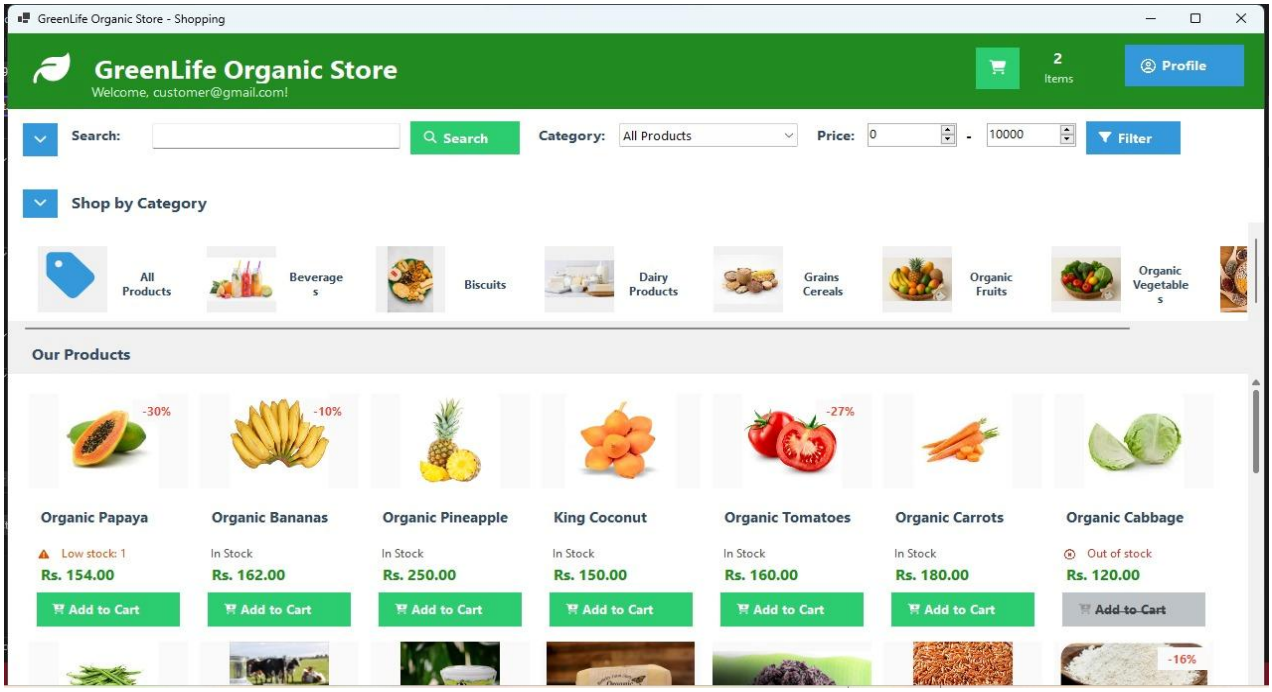


D.3 Password Reset Screen

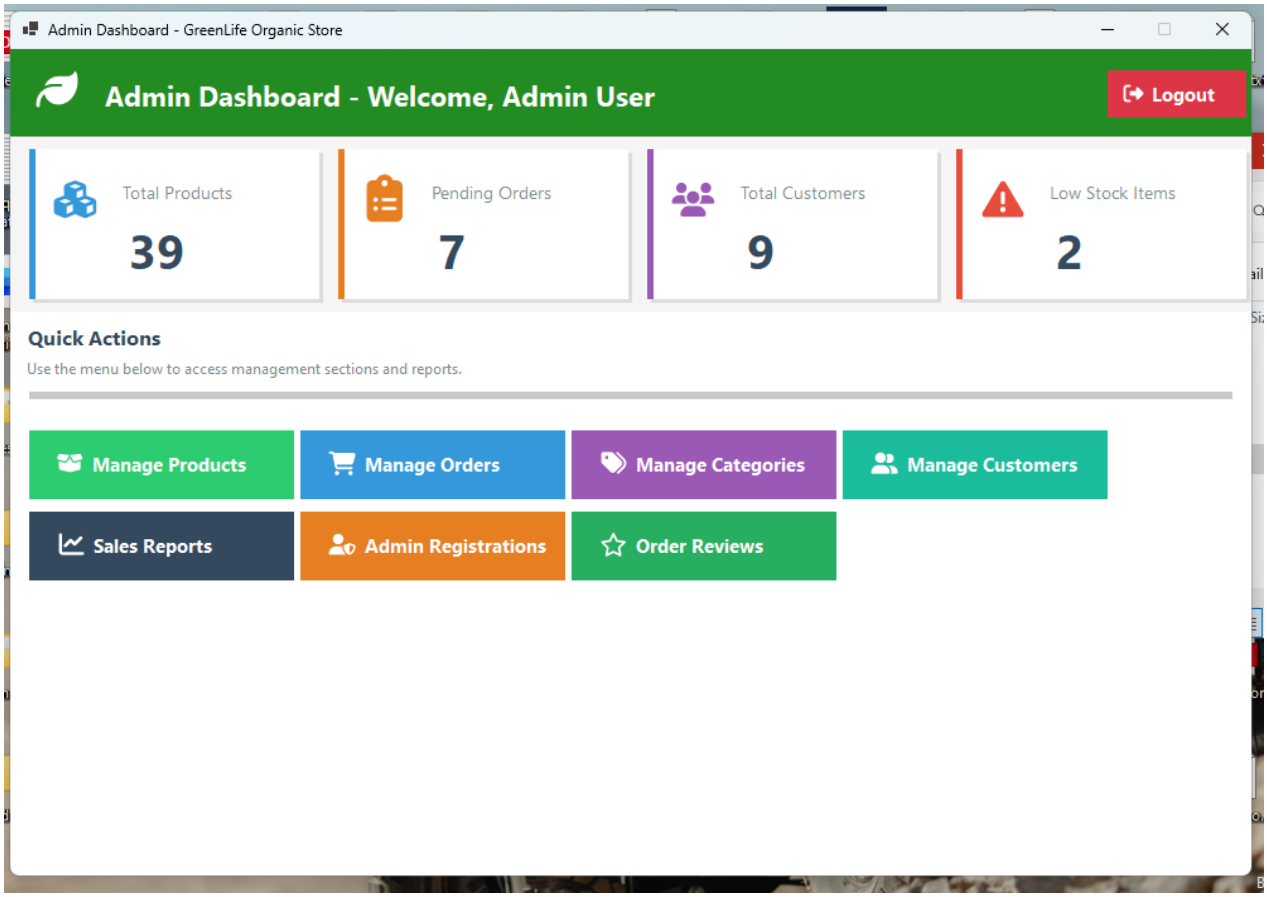


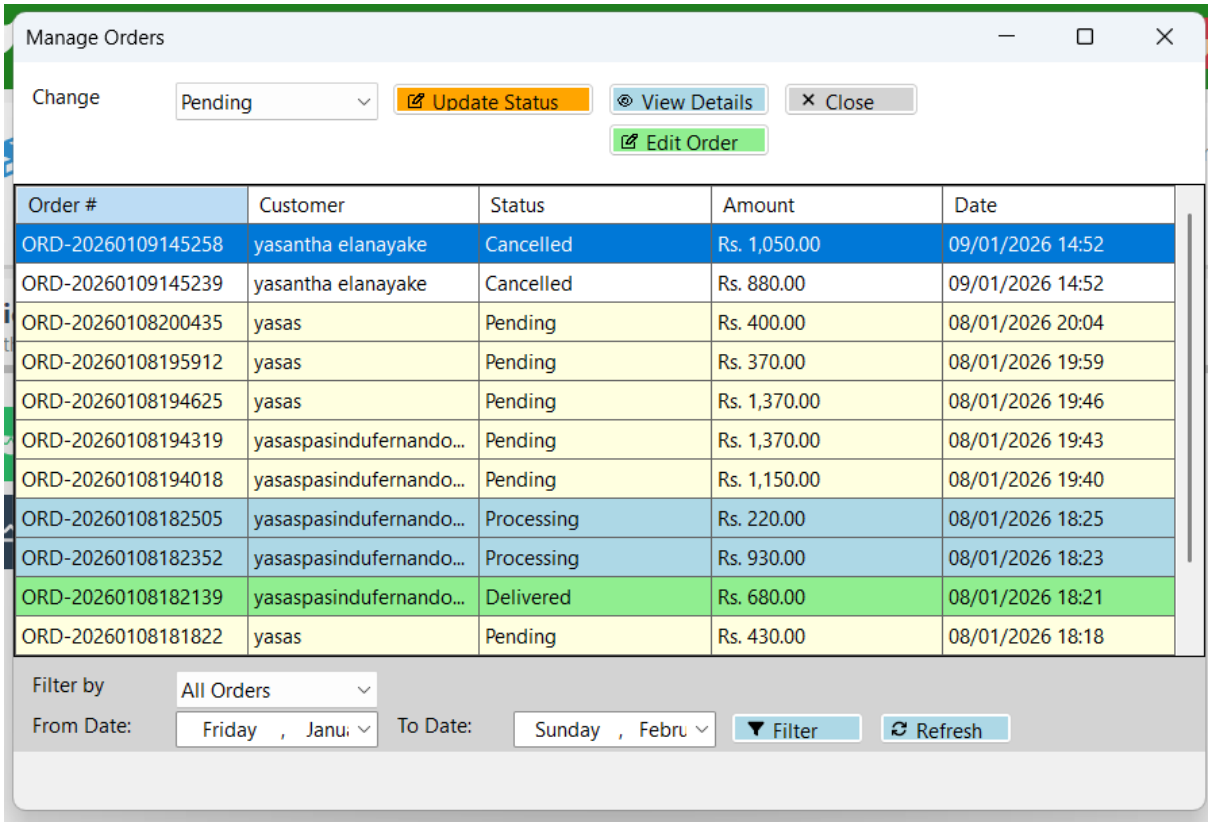


D.4 Customer dashboard

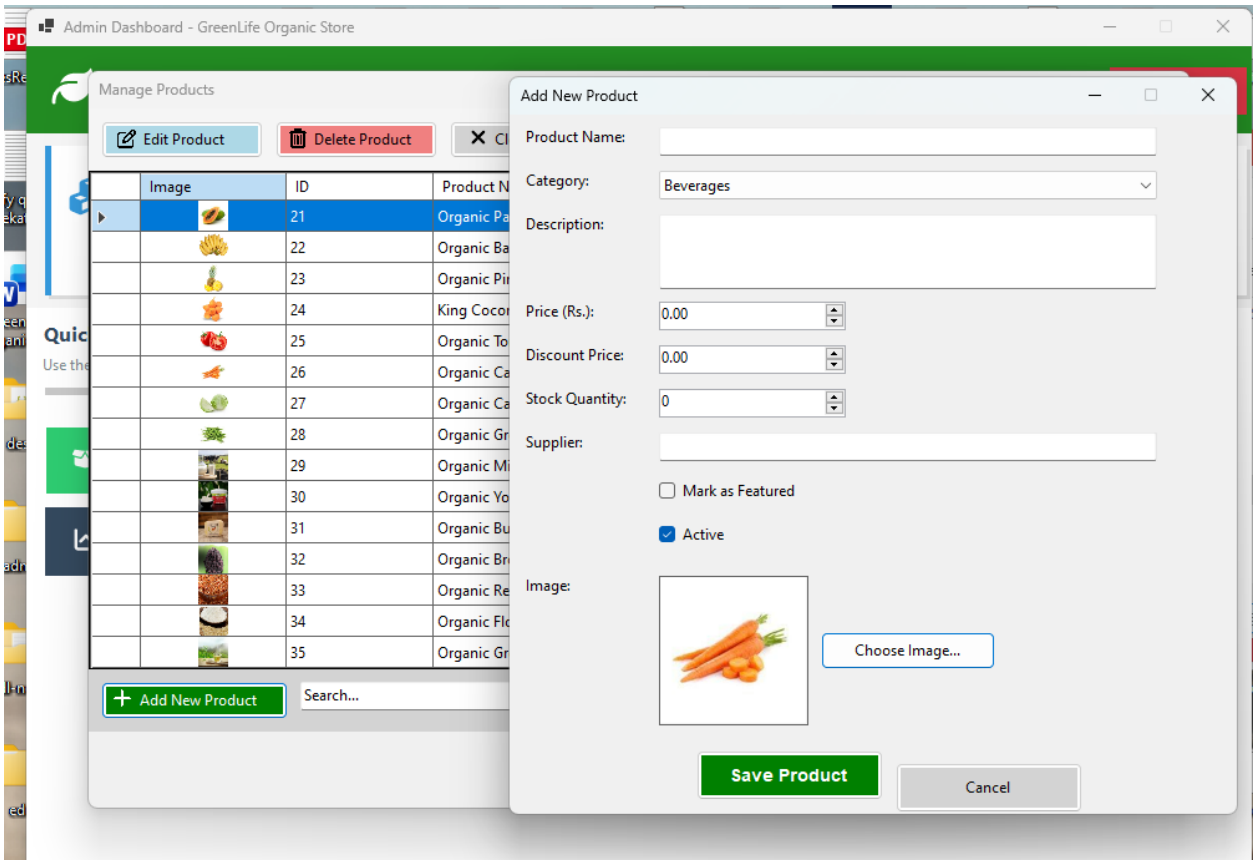


D.5 Admin dashboard

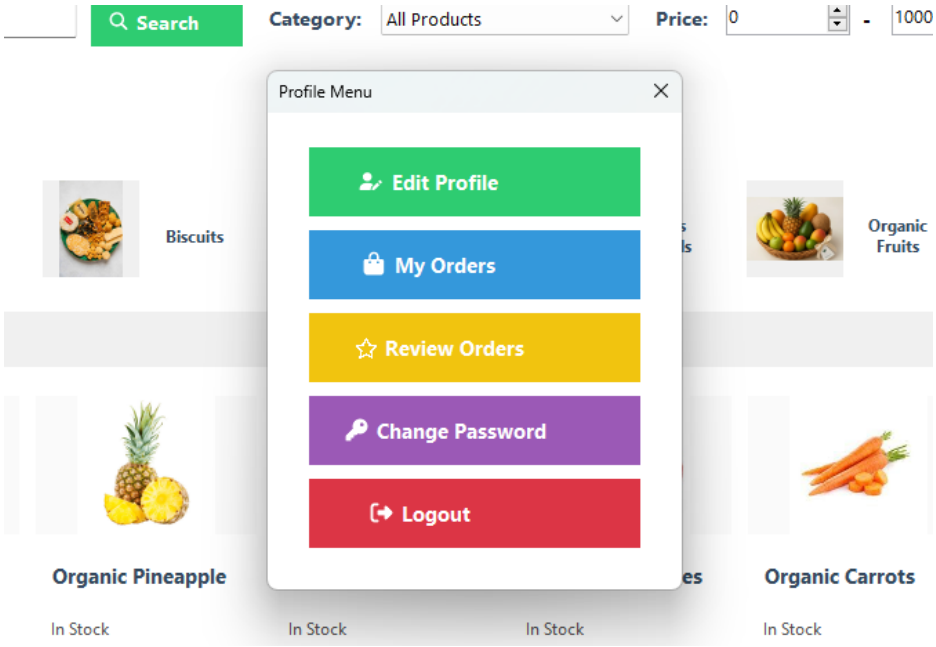




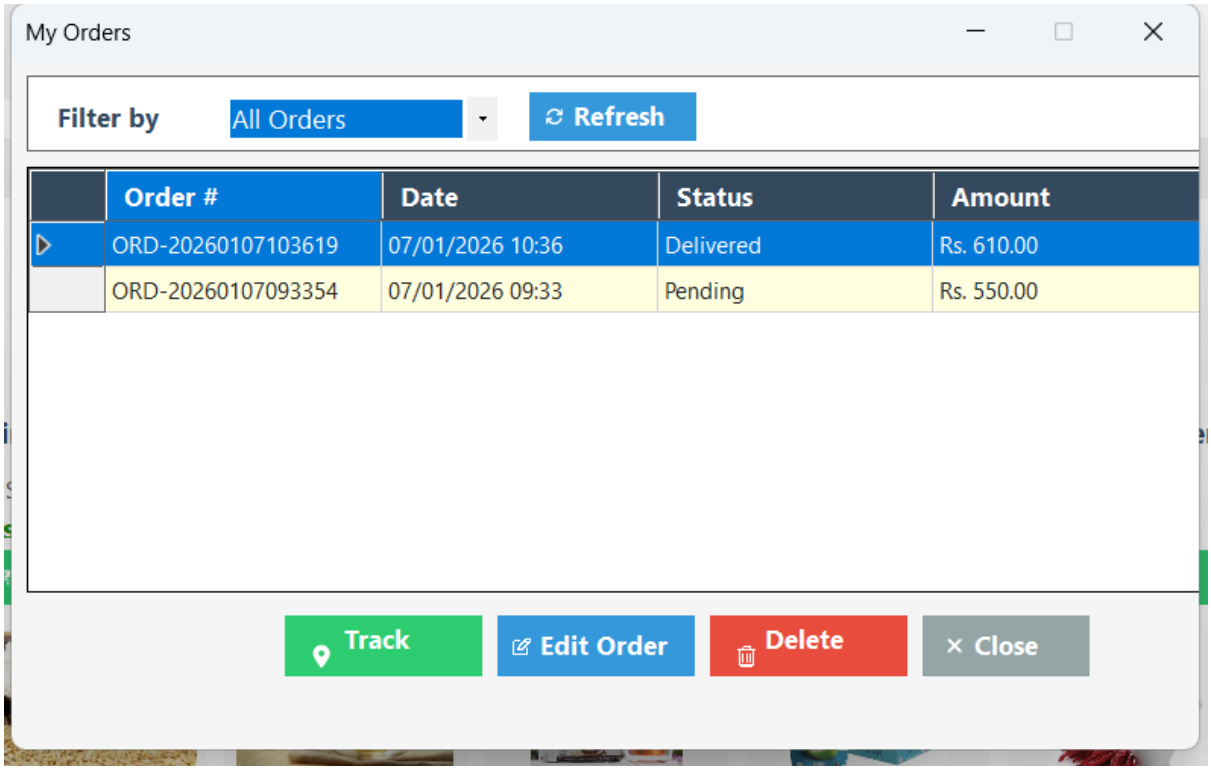
D.6 Product management screen



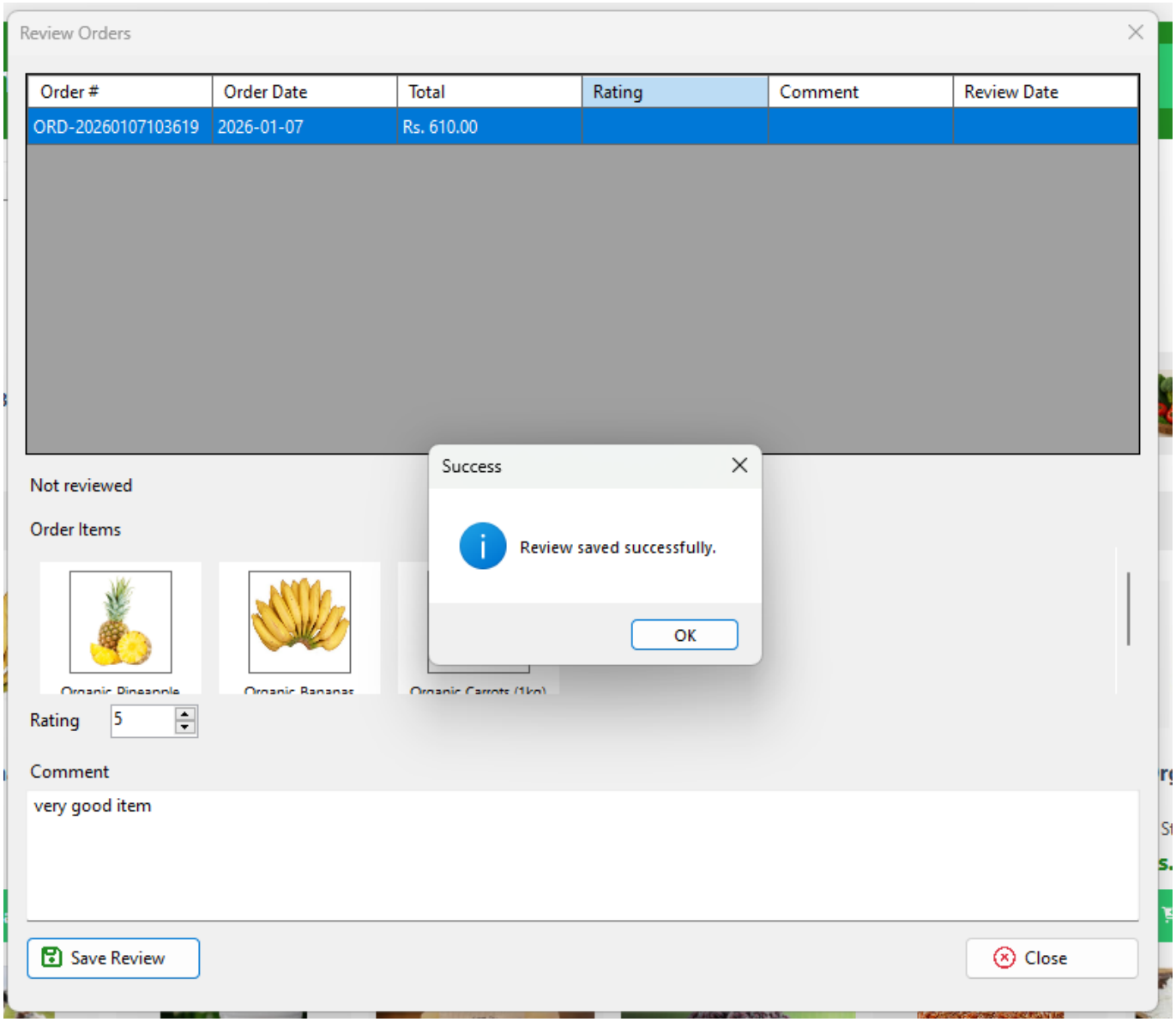
D.7 User Controllers



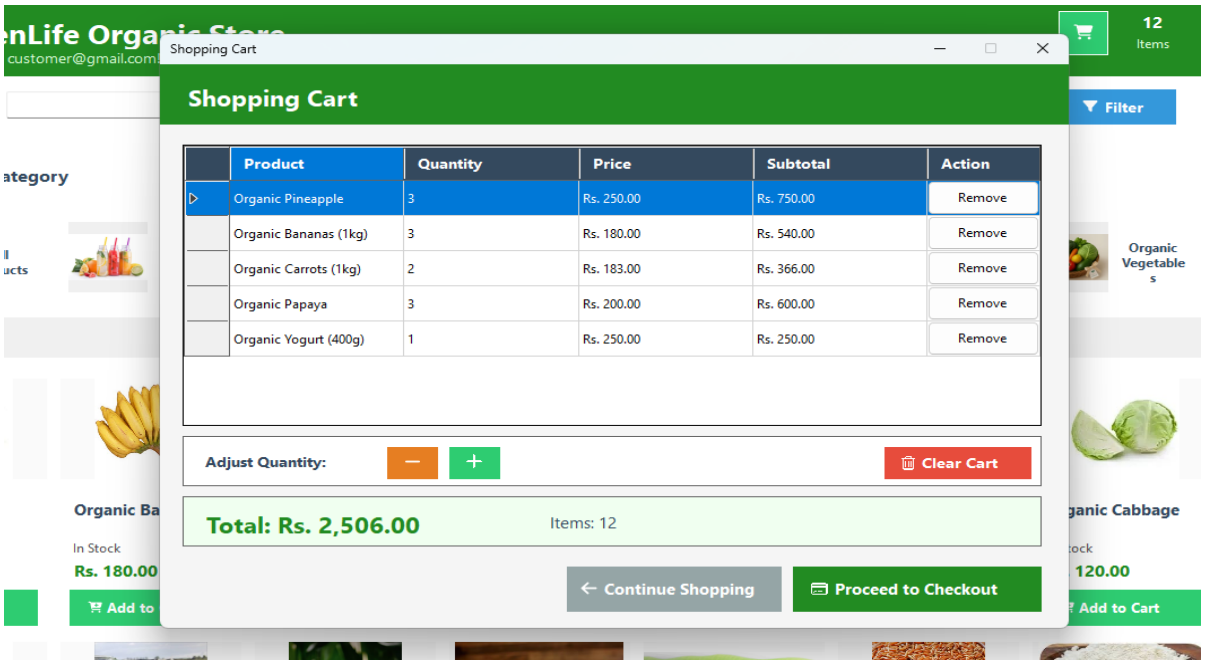
D.8 My Orders Screen

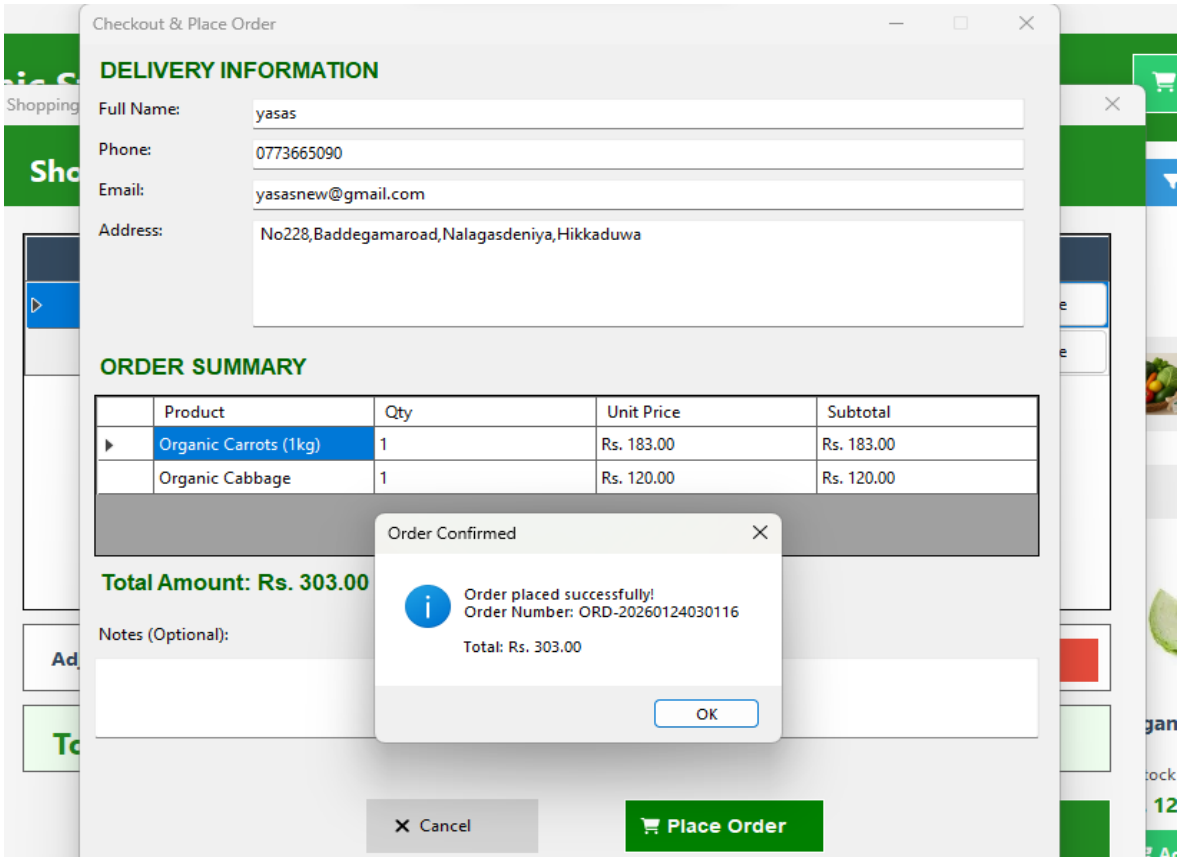


D.9 Product review Screen

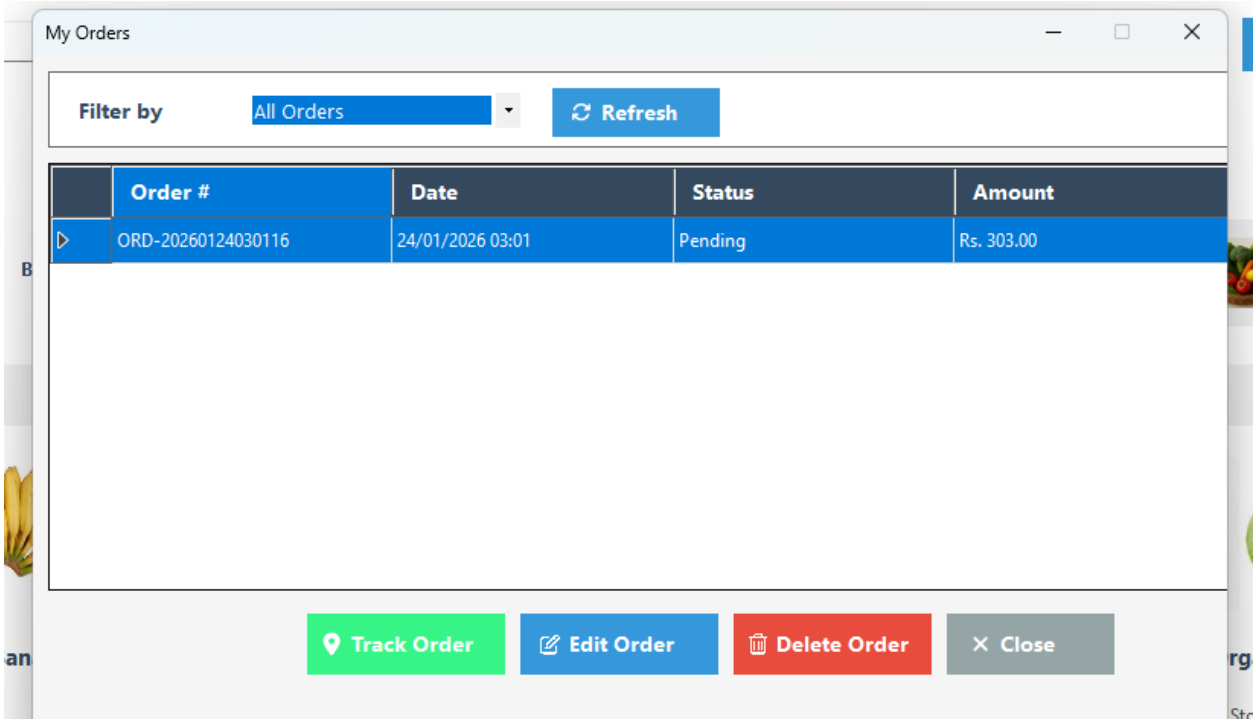


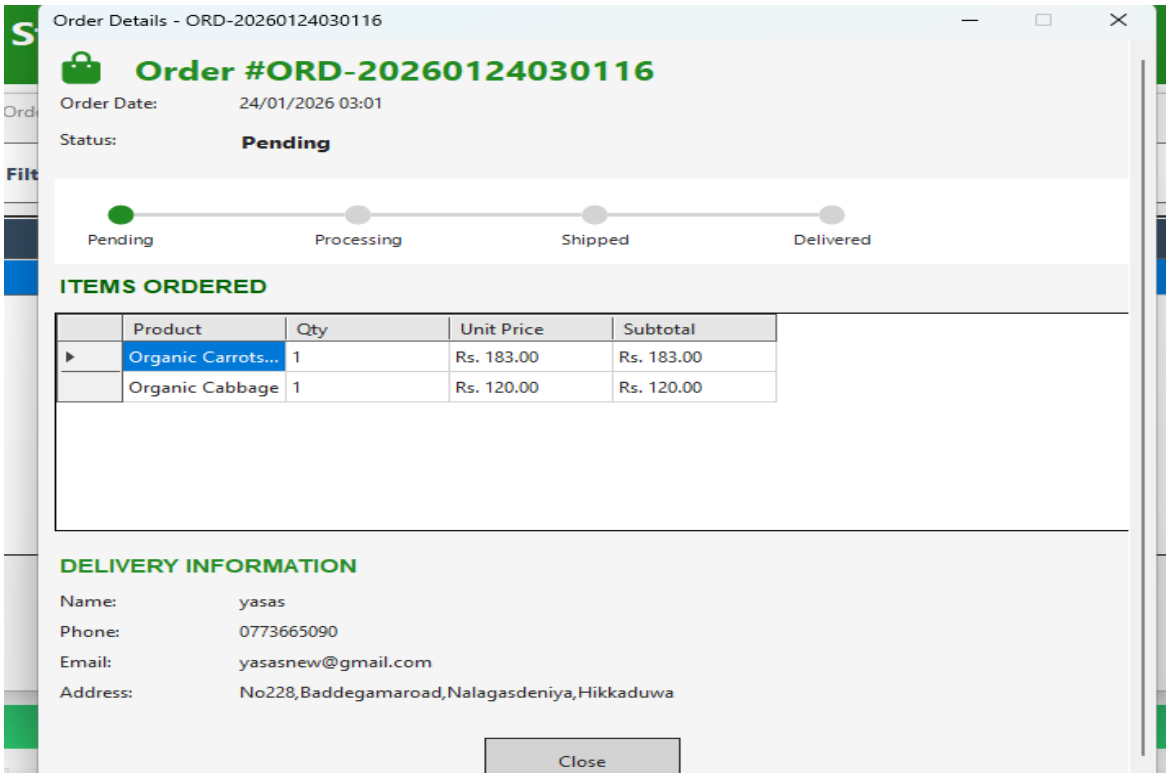
D.10 Shopping cart and checkout screen



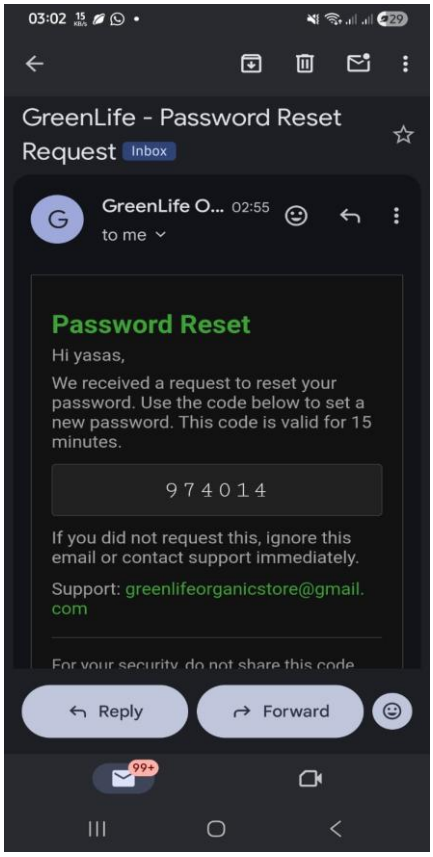


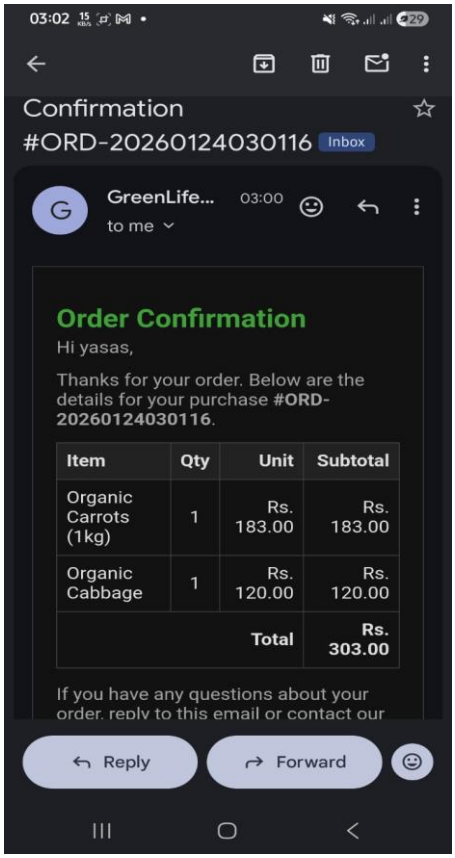
D.11 Order Tracking

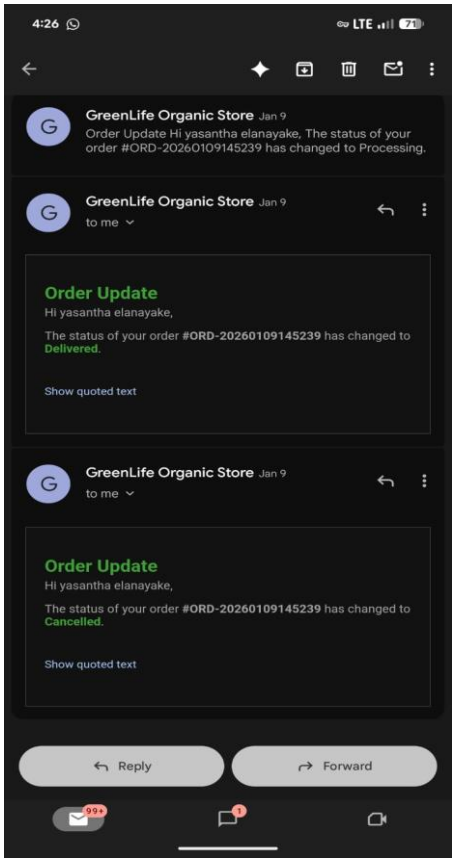


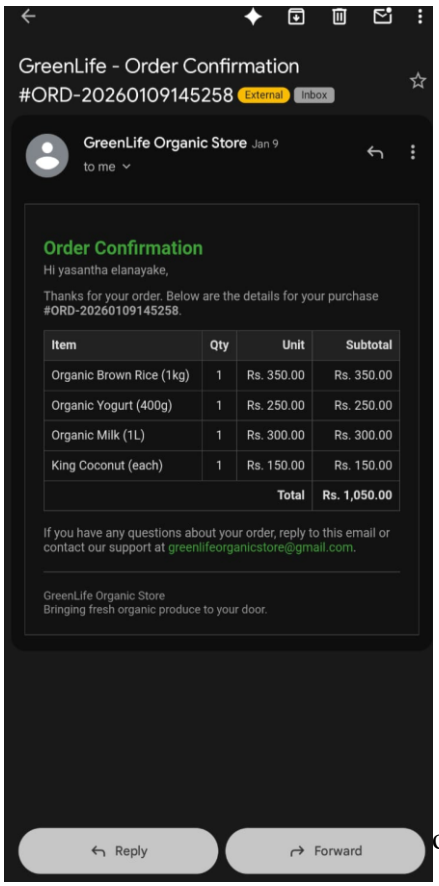


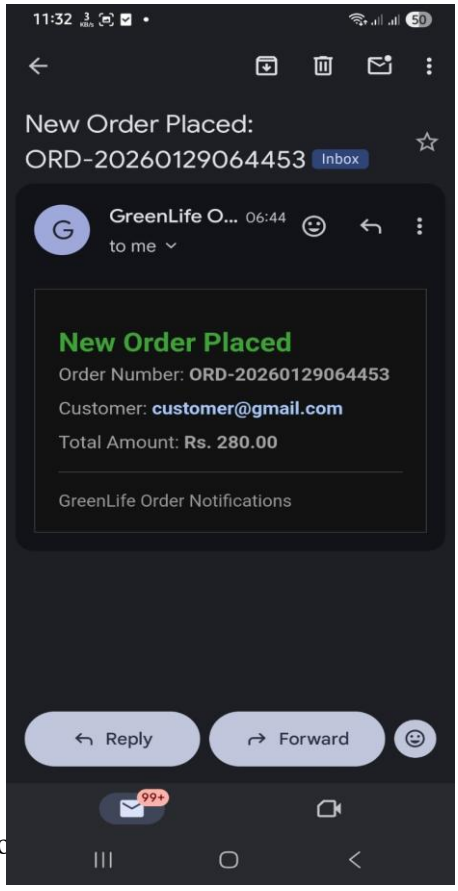
D.12 Email Bodies (Password Reset/Welcome Register /Low-stock Alert/ Order Confirmation/Delivery Update)

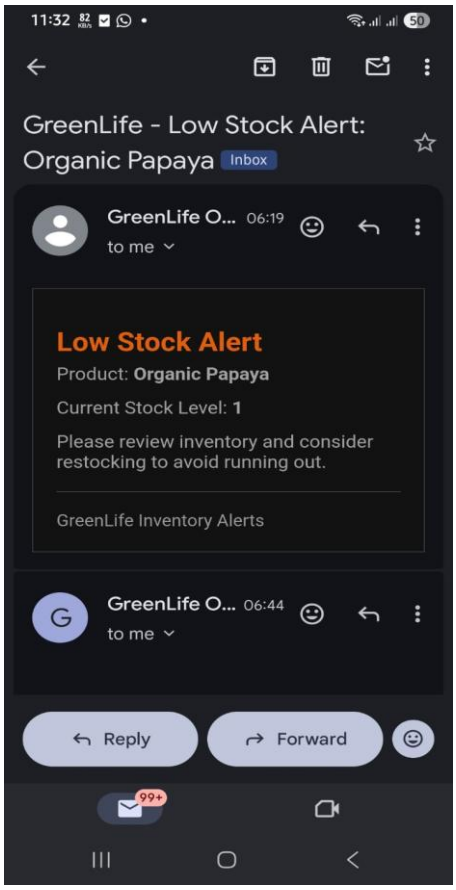




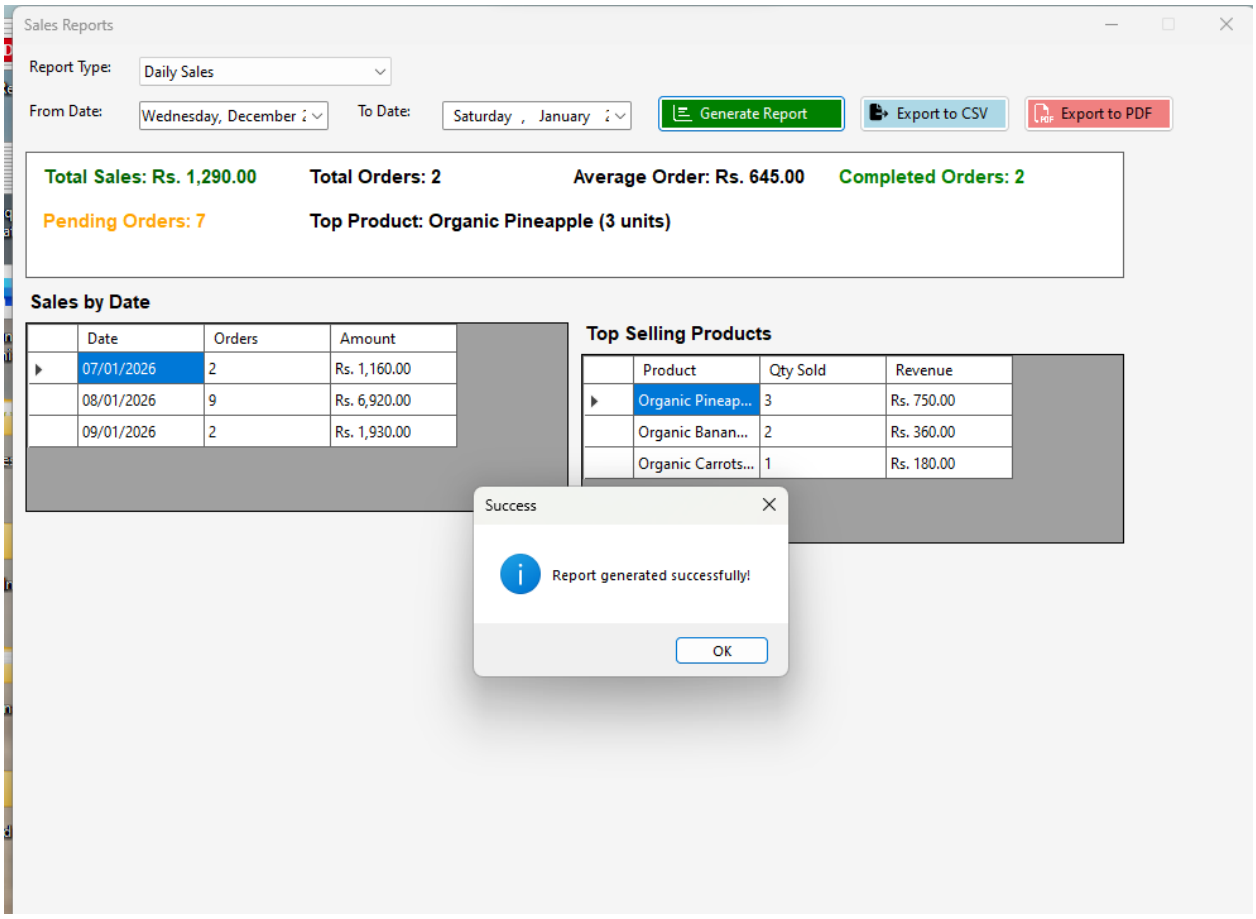


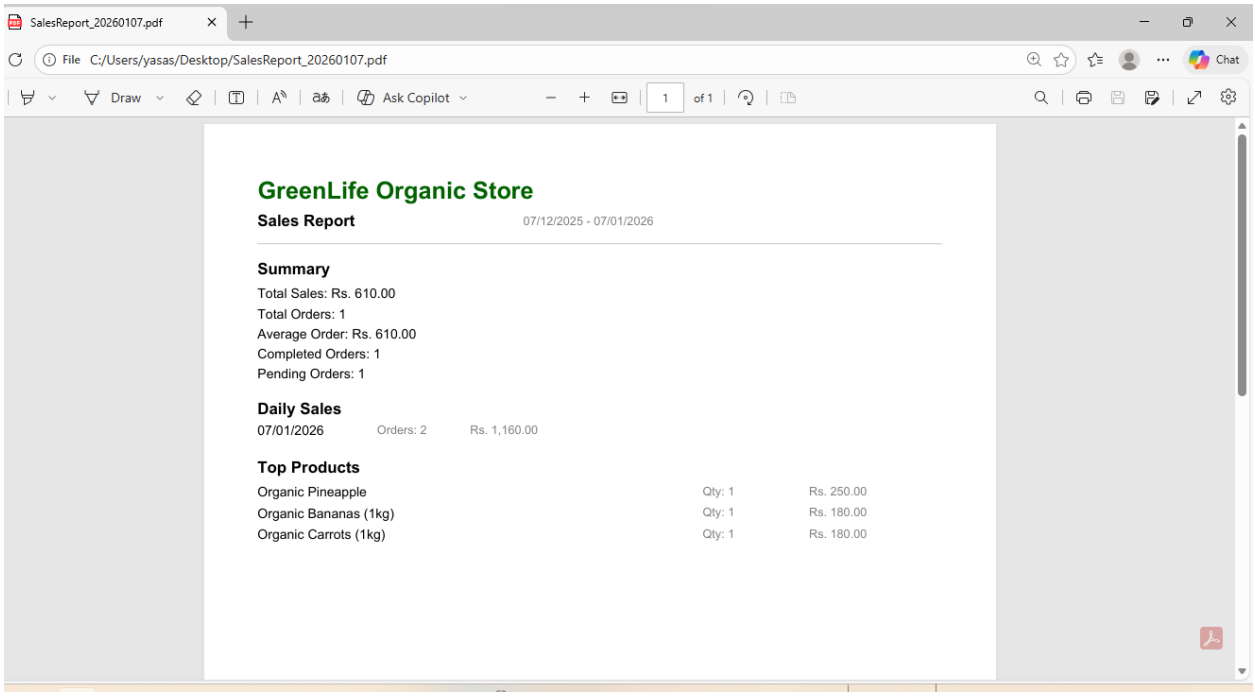


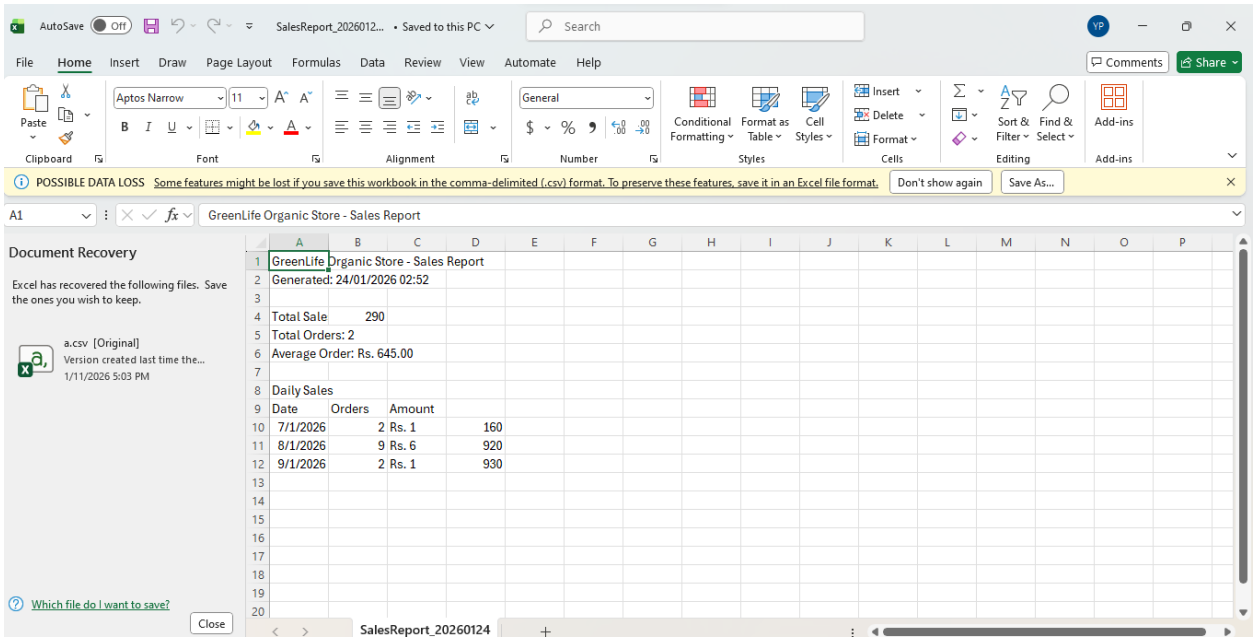




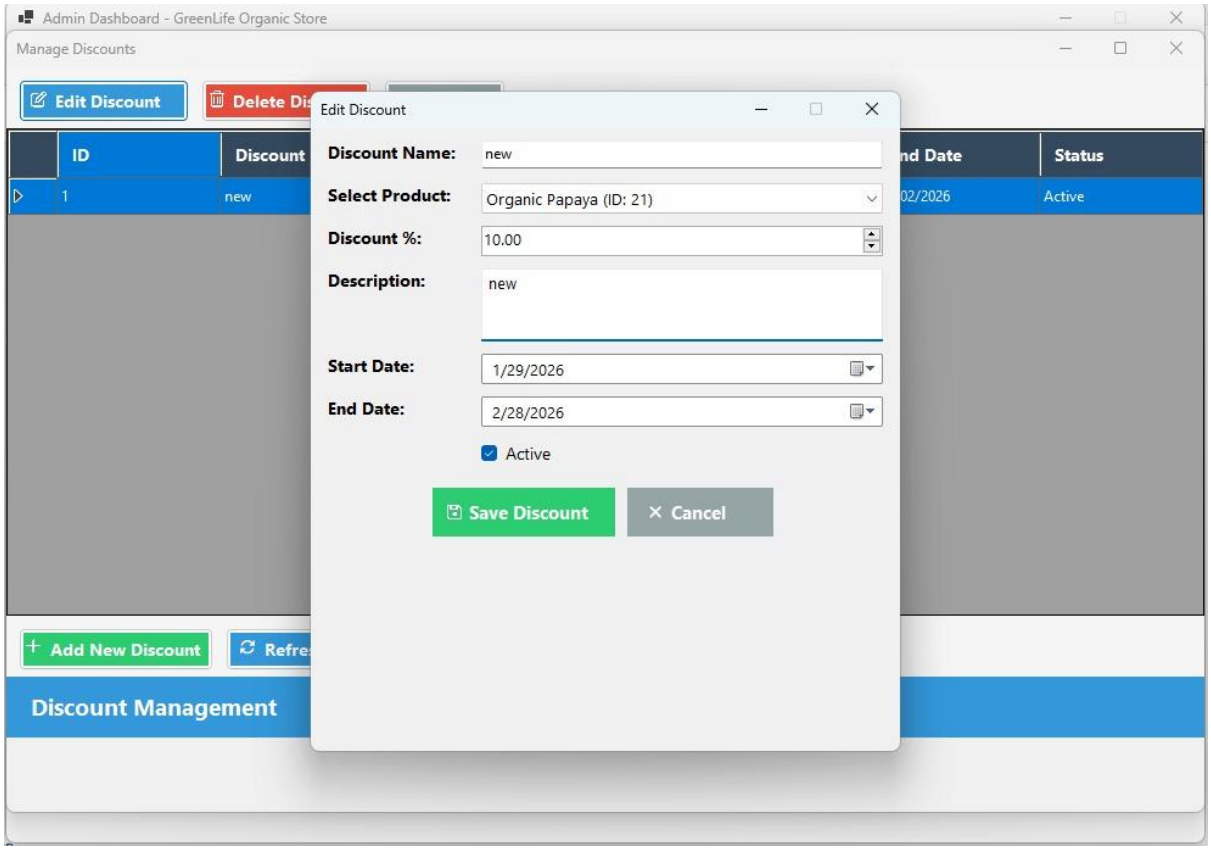
D.13 Reports and PDF/CSV export screens







D.14 Manage Discounts Screen



The UI artefacts demonstrate usability, layout consistency, and the application of DPI-aware design principles.

Supports Sections 6 and 9: Implemented Features and Testing.

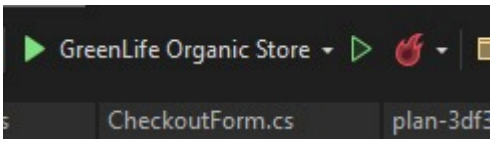
Appendix E: User Guide

The section contains complete user instructions for operating the GreenLife Organic Store application through its various functions. The guide covers both Admin and Customer operations.

E.1 Starting the Application

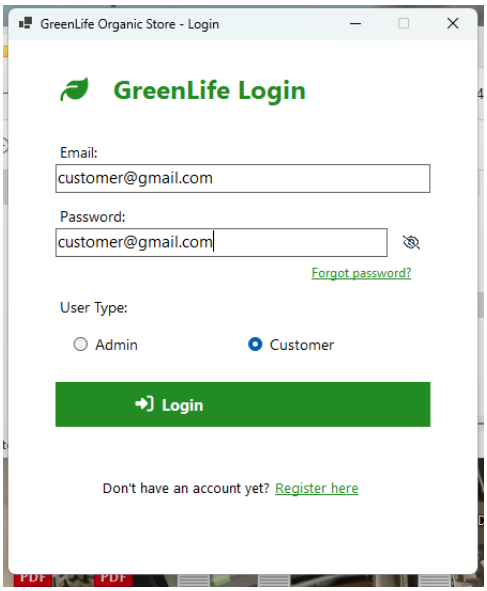
The application requires one of the following methods to start its functions:

- The user needs to open the project in Visual Studio and press **F5** or start.



- The user needs to execute the following compiled program **GreenLife Organic Store.exe**

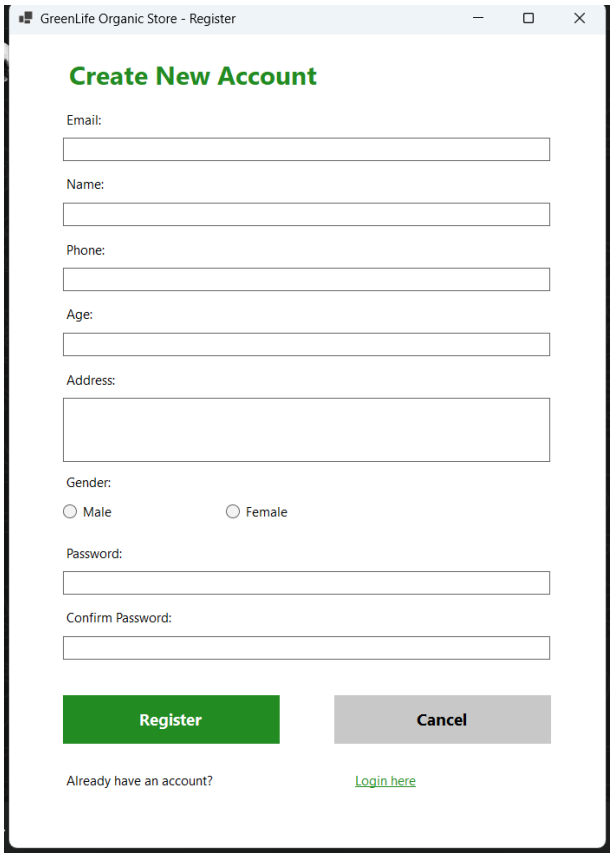
- The **Login Screen** will show up after the user starts the application.



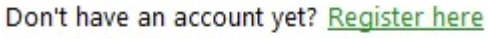
E.2 Login to the System

Enter your credentials:

Register with new email address.



Click.

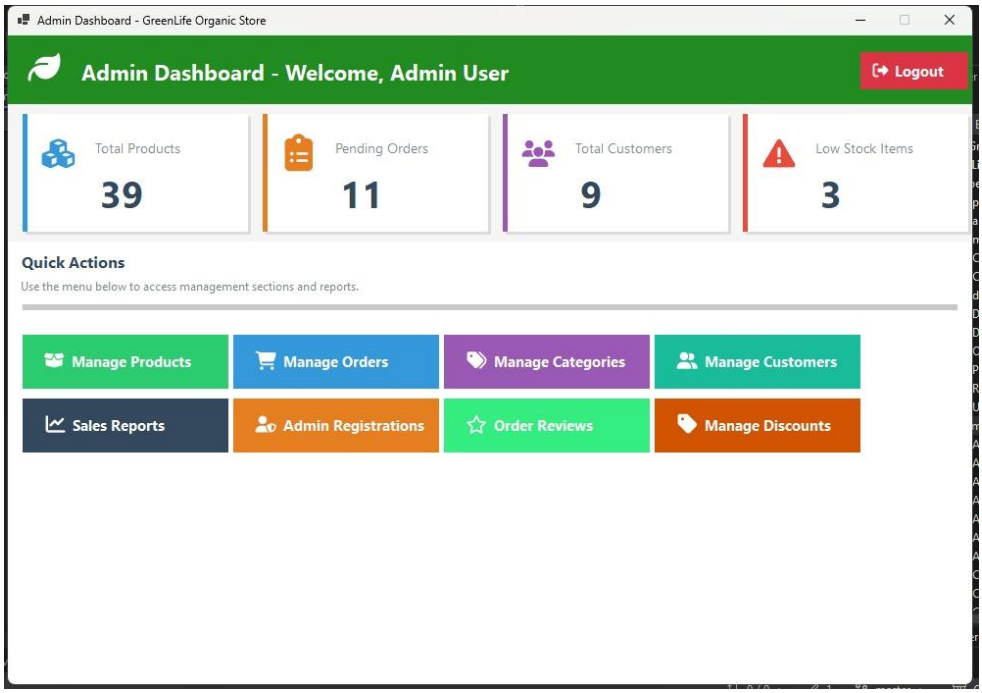


or use these.

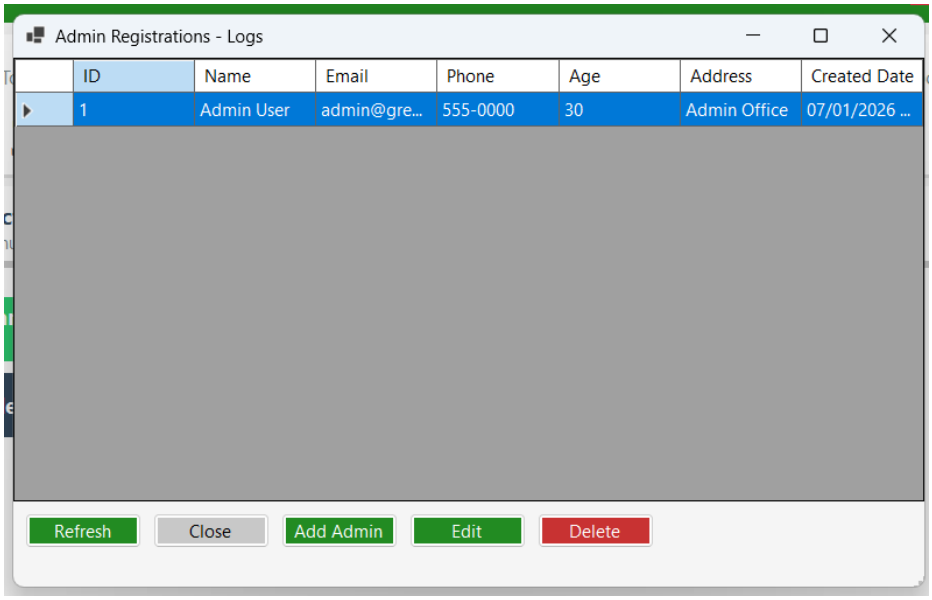
Role	**Email**	**Password**
Admin	admin@greenlife.com	admin@123
Customer	customer@gmail.com	customer@gmail.com

E.3 Admin User Guide

After logging in as Admin, the **Admin Dashboard** will appear.



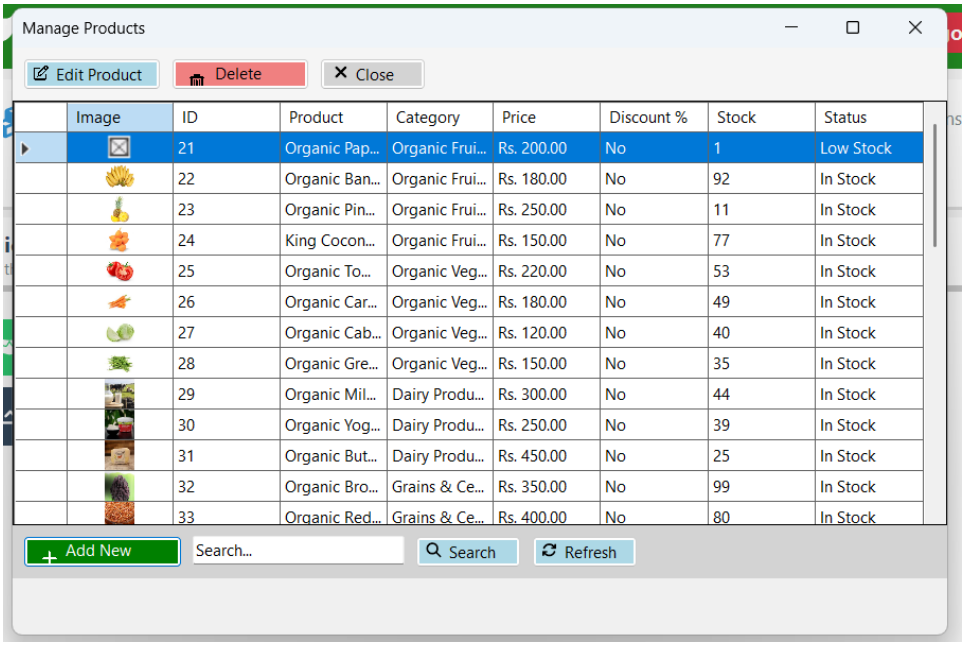
Use default admin password to **create** an **admin** user.



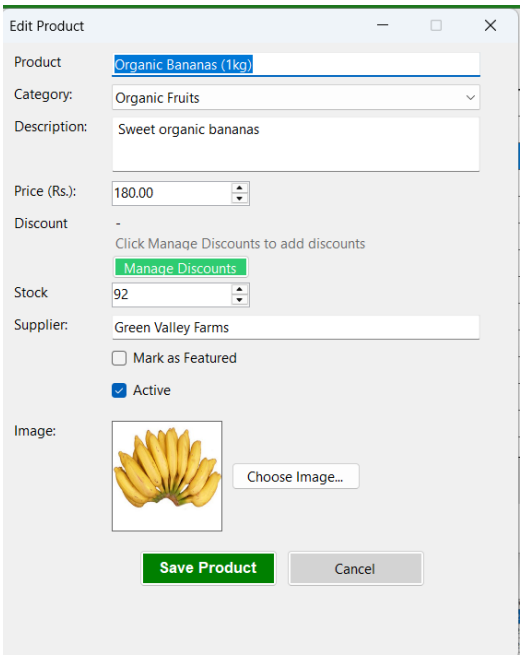
Please **make** sure **signup** with a **valid** email to get **email alerts** and **notifications**

E.3.1 Manage Products

1. Go to **Manage Products**



2. Click **Add** or **EDIT** Product

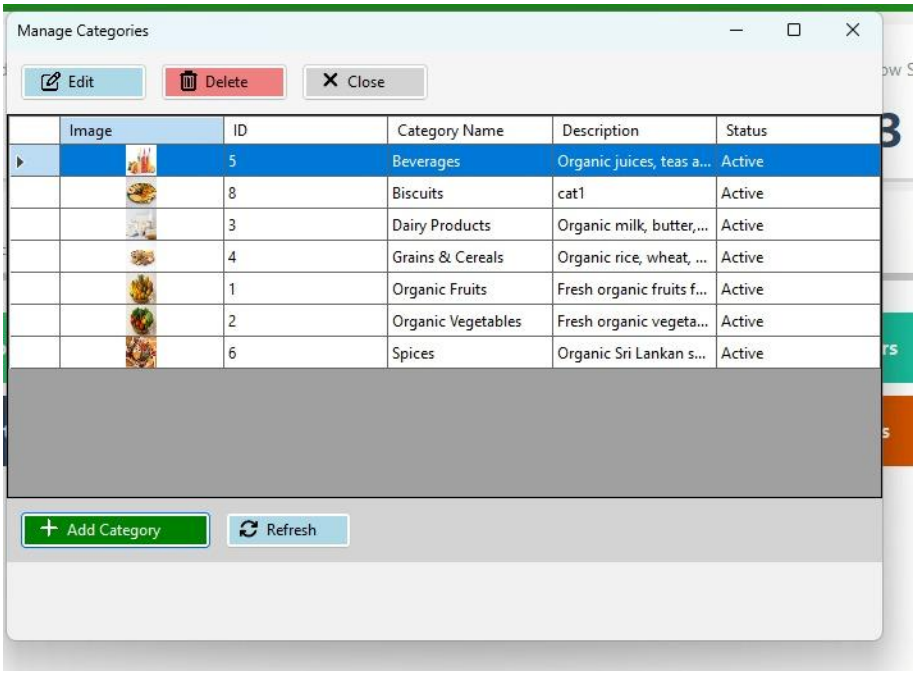


3. Enter product details (name, price, stock, category, image)
4. Click **Save**

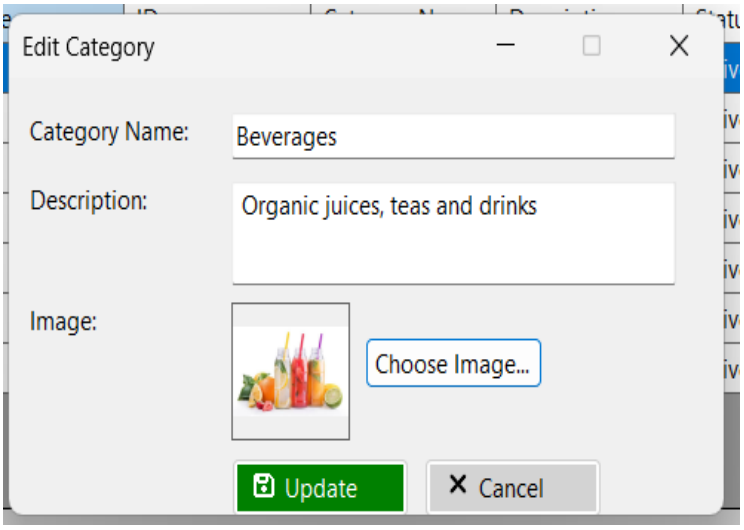
To edit or delete, select a product from the list.

E.3.2 Manage Categories

1. Open **Manage Categories**



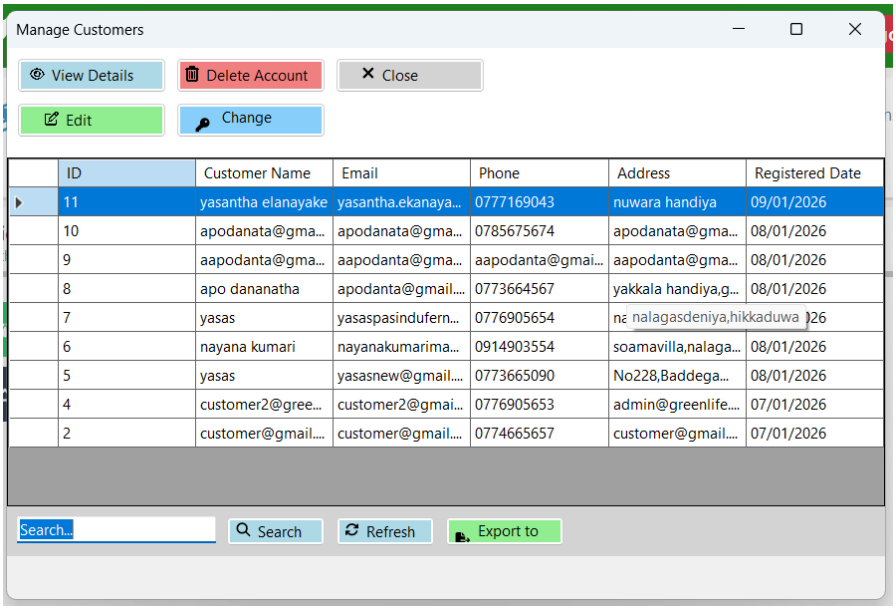
2. **Add**, **edit**, or **delete** categories.



3. Categories help organise products.

E.3.3 Manage Users & Customers

1. Open **Manage Customers**



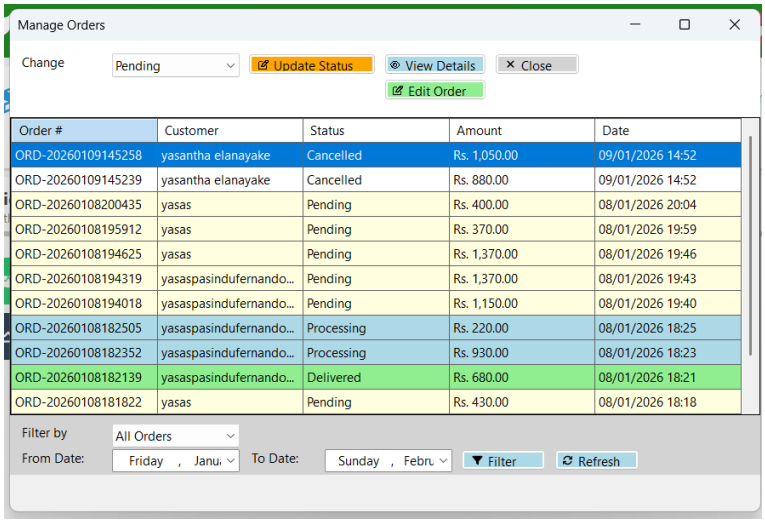
2. **Add**, **edit**, or **delete** Customers.

3. Categories help organise products.

Admins can:

- View customer accounts
- Activate or deactivate accounts.
- Update customer details.

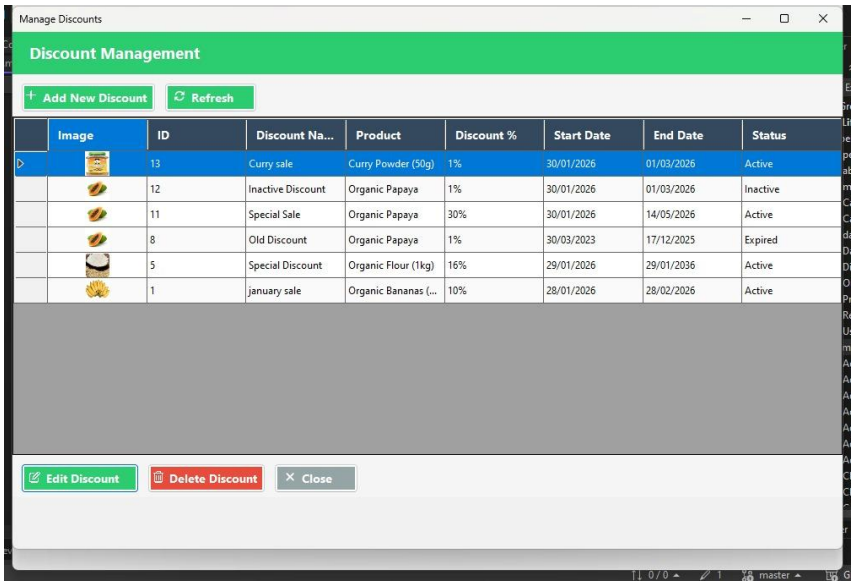
E.3.4 Manage Orders



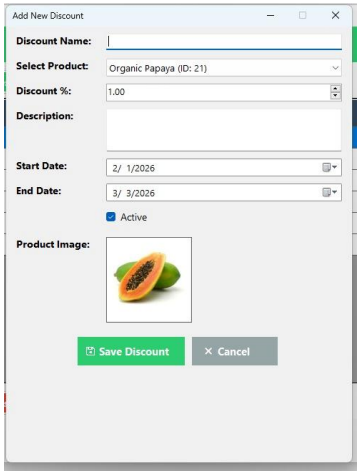
1. Open **Orders**
2. View all customer orders.
3. Update order status (Processing, Shipped, Delivered, Cancelled)

E.3.5 Manage Discounts

1. Go to **Discount Management**



2. Click **Add/Edit/Delete** Discounts



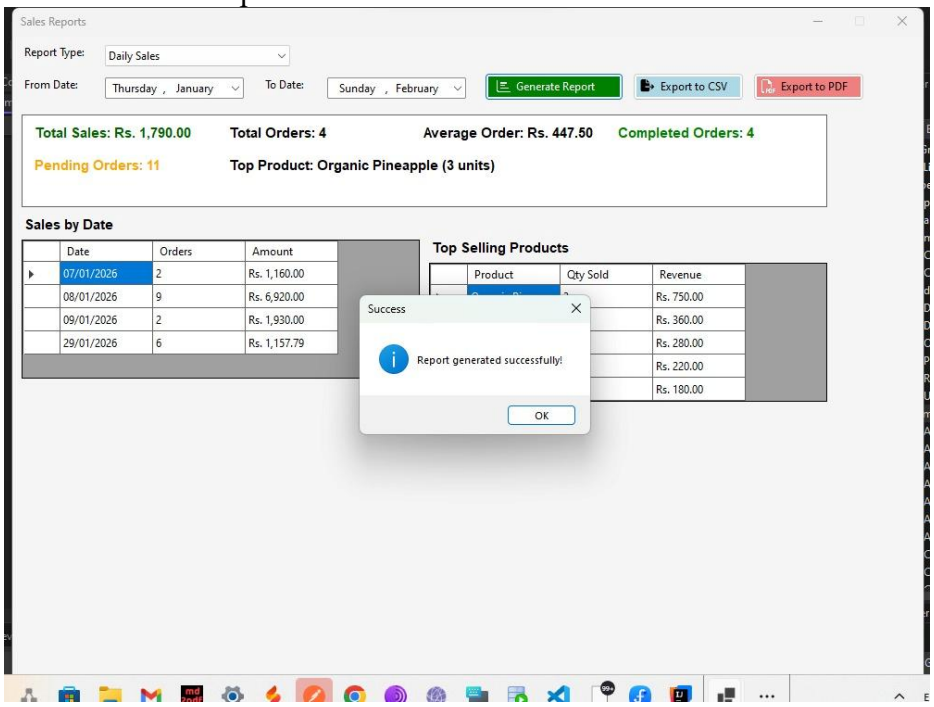
3. Select a product.
4. Enter discount percentage.
5. Set start and end dates.
6. Change the **Status**

7. **Save** the discount.

The system automatically applies valid discounts to product prices.

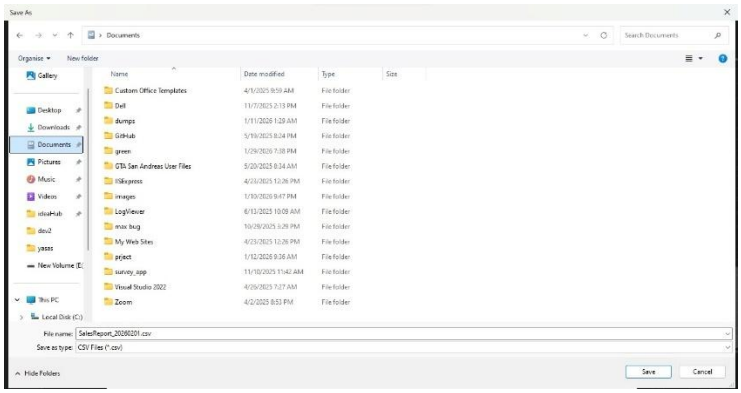
E.3.6 View Reports

1. Click Generate Reports



2. Add filters and sorting if want.

3. Export Reports as **CSV** or **PDF** format and save.

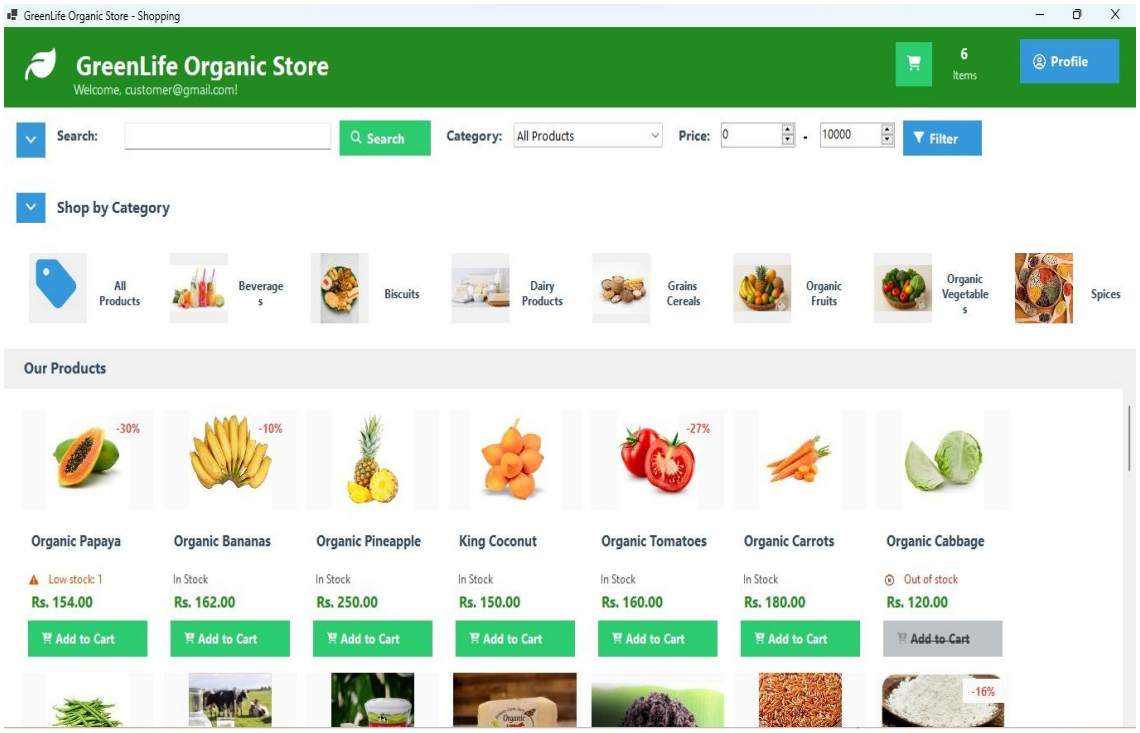


Admins can generate:

- Sales Reports
- Order Reports (PDF format)

E.4 Customer User Guide

After logging in as Customer, the **Customer Dashboard** appears.

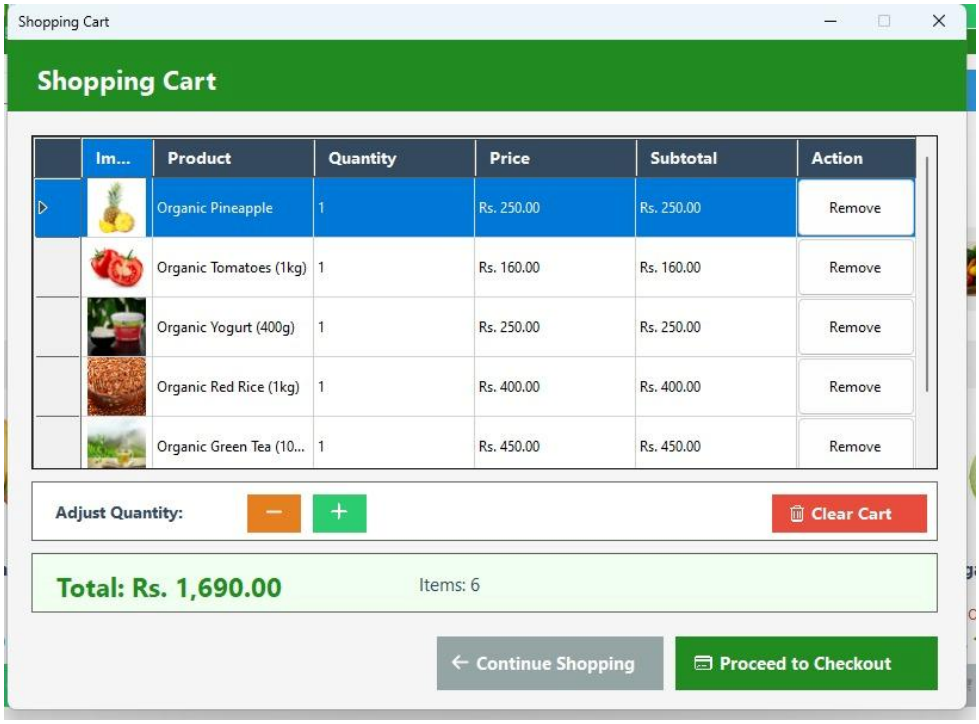


E.4.1 Browse Products

Customers can:

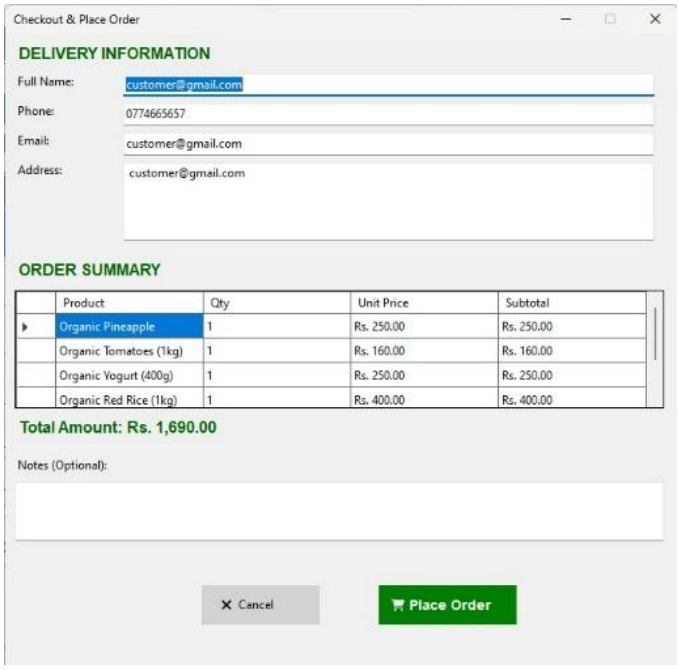
- View categories
- Search products by name.
- See discounted products (with badge)

E.4.2 Add to Cart



1. Click **Add to Cart** on a product.
2. Open **Shopping Cart**
3. Adjust quantity if needed.

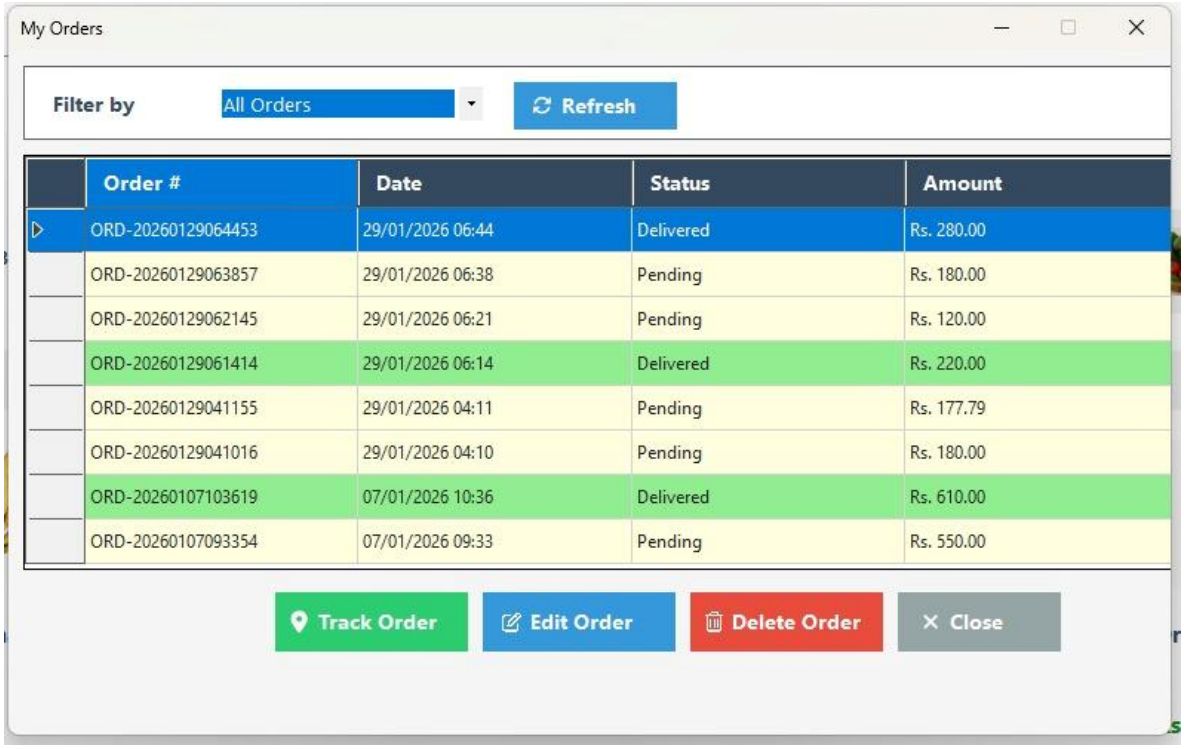
E.4.3 Checkout



1. Click **Checkout**
2. Confirm order details.
3. Place the order.

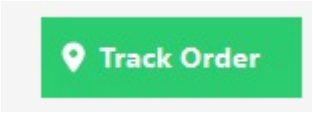
If a discount is active, the **reduced price** will be applied automatically.

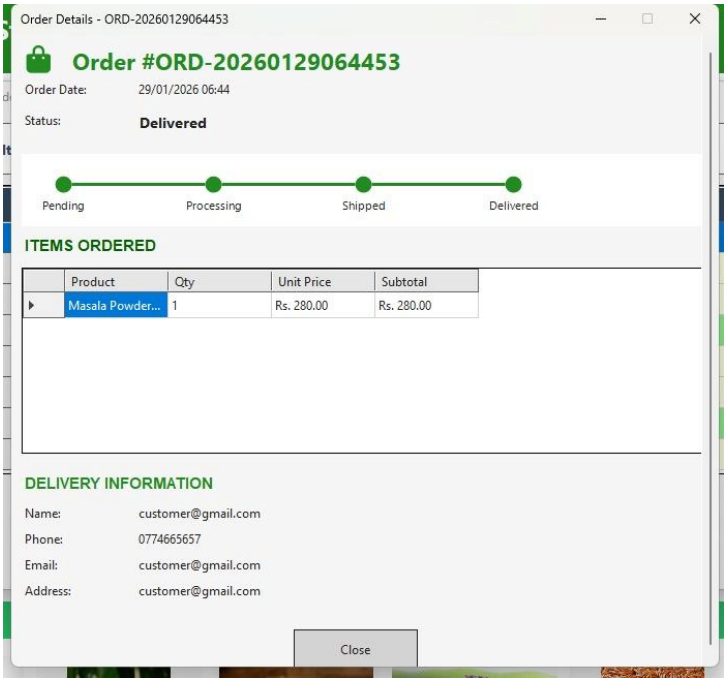
E.4.4 View Orders



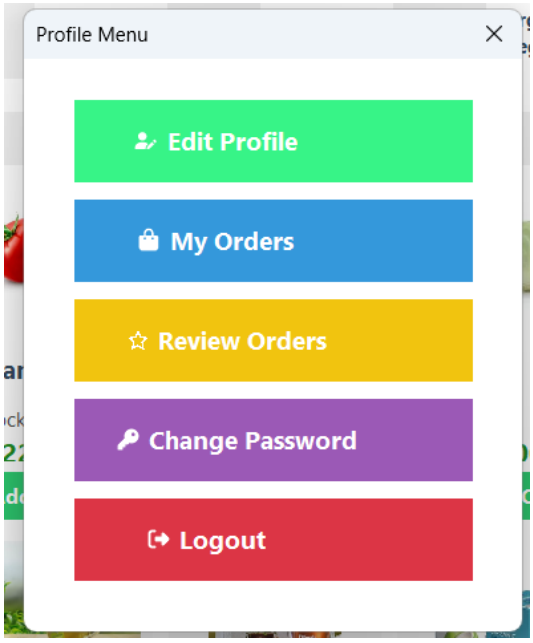
Customers can:

- View order history
- Track order status.





E.4.5 Profile Menu



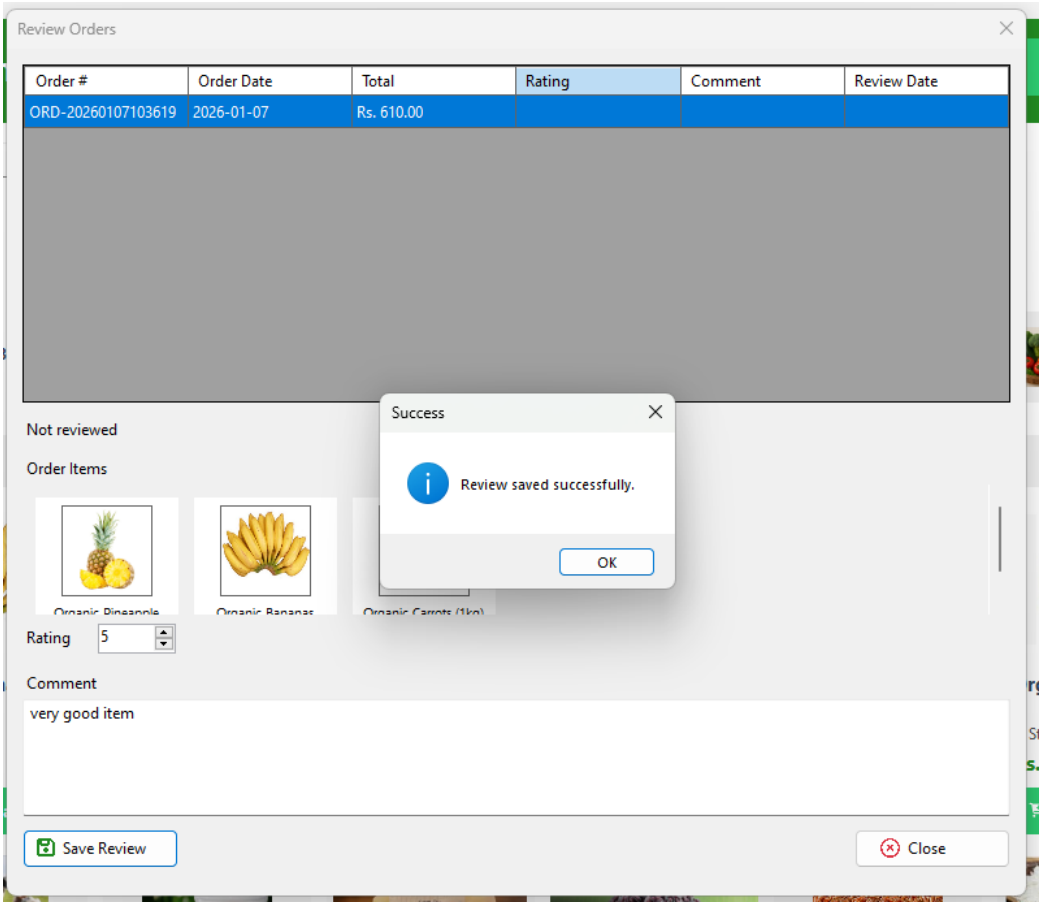
Customers can:

- Edit personal details.
- Change password.
- View orders
- Review orders

E.4.6 Submit Reviews

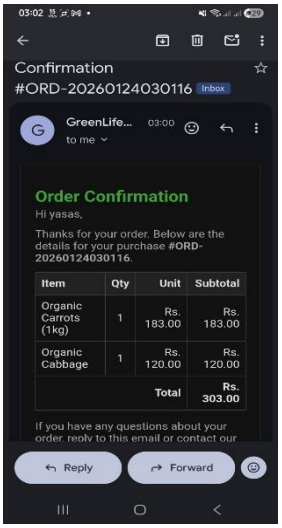
After receiving an order:

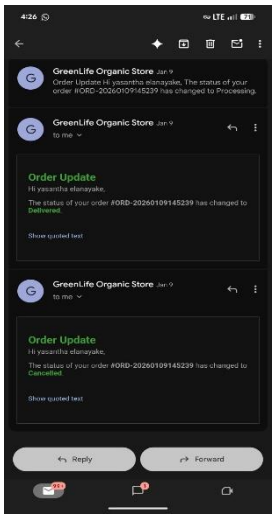
1. Open **Review Orders**

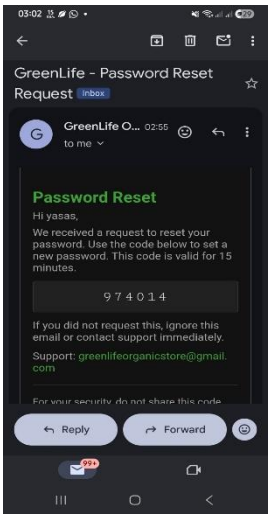


2. Select an order.
3. Leave a rating (1–5 stars)
4. Write a review comment.

E.5 Email Notifications



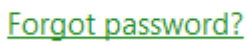




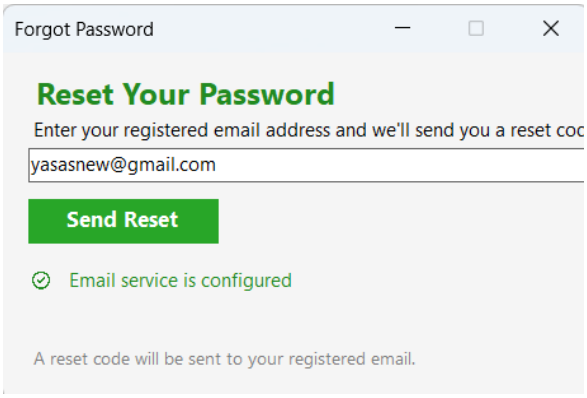
The system may send emails for:

- Order confirmations
- Password resets
- Order status updates.

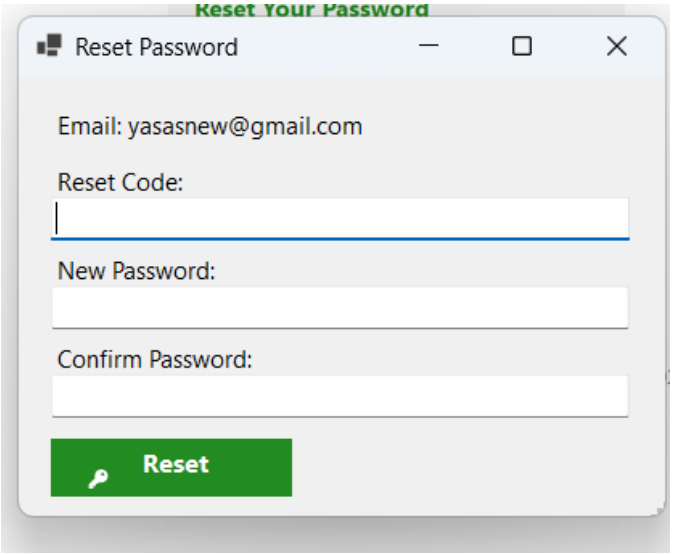
E.6 Password Reset



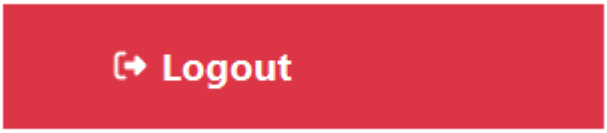
1. Click **Forgot Password?**
2. Enter valid **Registered Email**



3. **Enter** the **OTP** and **Reset** the **Password**



E.6 Logging Out



Click **Logout** from the dashboard to safely exit your account.