ONLINE RETAIL APPLICATION MANAGEMENT SYSTEM

A PROJECT REPORT

Submitted by

HARIHARAN M (2303811710421052)

in partial fulfillment of requirements for the award of the course

CGB1221-DATABASE MANAGEMENT SYSTEMS

in

COMPUTER SCIENCE AND ENGINEERING

K. RAMAKRISHNAN COLLEGE OF TECHNOLOGY

(An Autonomous Institution, affiliated to Anna University Chennai and Approved by AICTE, New Delhi)

SAMAYAPURAM – 621 112

JUNE- 2025

K. RAMAKRISHNAN COLLEGE OF TECHNOLOGY (AUTONOMOUS)

SAMAYAPURAM – 621 112

BONAFIDE CERTIFICATE

Certified that this project report on **"ONLINE RETAIL APPLICATION MANAGEMENT SYSTEM"** is the bonafide work of **HARIHARAN M (2303811710421052)** who carried out the project work during the academic year 2024 - 2025 under my supervision.

SIGNATURE

Mrs.A.Delphin Carolina Rani, M.E.,Ph.D.,

HEAD OF THE DEPARTMENT

PROFESSOR

Department of CSE

K.Ramakrishnan College of Technology (Autonomous)

Samayapuram–621112.

SIGNATURE

Ms. S. Uma Mageshwari, M.E.,

SUPERVISOR

ASSISTANT PROFESSOR

Department of CSE

K.Ramakrishnan College of Technology (Autonomous)

Samayapuram–621112.

Submitted for the viva-voce examination held on 02.06.2025 .

INTERNAL EXAMINER

EXTERNAL EXAMINER

DECLARATION

I declare that the project report on **“ ONLINE RETAIL APPLICATION MANAGEMENT SYSTEM ”** is the result of original work done by us and best of our knowledge, similar work has not been submitted to **“ANNA UNIVERSITY CHENNAI”** for the requirement of Degree of **BACHELOR OF ENGINEERING**. This project report is submitted on the partial fulfilment of the requirement of the completion of the course **CGB1221 – DATABASE MANAGEMENT SYSTEMS.**

.

Signature

HARIHARAN M

Place: Samayapuram

Date : 02.06.2025

ACKNOWLEDGEMENT

It is with great pride that I express my gratitude and in-debt to our institution "**K.Ramakrishnan College of Technology (Autonomous)**", for providing us with the opportunity to do this project.

I glad to credit honourable chairman **Dr. K. RAMAKRISHNAN**, **B.E.,** for having provided for the facilities during the course of our study in college.

I would like to express my sincere thanks to our beloved Executive Director **Dr. S. KUPPUSAMY, MBA, Ph.D.,** for forwarding to our project and offering adequate duration in completing our project.

I would like to thank **Dr. N. VASUDEVAN, M.Tech., Ph.D.,** Principal, who gave opportunity to frame the project the full satisfaction.

I whole heartily thanks to **Dr. A. DELPHIN CAROLINA RANI, M.E.,Ph.D.,** Head of the department, **COMPUTER SCIENCE AND ENGINEERING** for providing her encourage pursuing this project.

I express my deep expression and sincere gratitude to our project supervisor **Ms. S. UMA MAGESHWARI, M.E.,** Department of **COMPUTER SCIENCE AND ENGINEERING,** for her incalculable suggestions, creativity, assistance and patience which motivated us to carry out this project.

I render my sincere thanks to Course Coordinator and other staff members for providing valuable information during the course.

I wish to express my special thanks to the officials and Lab Technicians of our departments who rendered their help during the period of the work progress.

VISION OF THE INSTITUTION

To serve the society by offering top-notch technical education on par with global standards

MISSION OF THE INSTITUTION

- Be a center of excellence for technical education in emerging technologies by exceeding the needs of the industry and society.
- Be an institute with world class research facilities
- Be an institute nurturing talent and enhancing the competency of students to transform them as all-round personality respecting moral and ethical values

VISION OF DEPARTMENT

To be a center of eminence in creating competent software professionals with research and innovative skills.

MISSION OF DEPARTMENT

M1: Industry Specific: To nurture students in working with various hardware and software platforms inclined with the best practices of industry.

M2: Research: To prepare students for research-oriented activities.

M3: Society: To empower students with the required skills to solve complex technological problems of society.

PROGRAM EDUCATIONAL OBJECTIVES

1. PEO1: Domain Knowledge

To produce graduates who have strong foundation of knowledge and skills in the field of Computer Science and Engineering.

2. PEO2: Employability Skills and Research

To produce graduates who are employable in industries/public sector/research organizations or work as an entrepreneur.

3. PEO3: Ethics and Values

To develop leadership skills and ethically collaborate with society to tackle real-world challenges.

PROGRAM SPECIFIC OUTCOMES (PSOs)

PSO 1: Domain Knowledge

To analyze, design and develop computing solutions by applying foundational concepts of Computer Science and Engineering.

PSO 2: Quality Software

To apply software engineering principles and practices for developing quality software for scientific and business applications.

PSO 3: Innovation Ideas

To adapt to emerging Information and Communication Technologies (ICT) to innovate ideas and solutions to existing/novel problems

PROGRAM OUTCOMES (POs)

Engineering students will be able to:

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences
3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations
4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions

5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations
6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice
7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

ABSTRACT

The Online Retail Application Management System is a desktop-based application developed using Python and Tkinter that offers an intuitive interface for managing user accounts, product inventories, customer orders, and sales statistics in a simulated e-commerce environment. It leverages SQLite as the backend database to store and manage structured data for users, products, orders, and transactions. The system supports both administrator and customer roles, with secure login and registration functionality. Customers can browse available products, add items to a virtual shopping cart, and place orders with real-time stock validation. Upon checkout, orders are recorded with timestamps and total payment information, and the inventory is updated accordingly. Administrators can add new products and monitor sales performance through a visual dashboard implemented using Matplotlib, which displays bar charts of product-wise revenue and quantities sold. The user interface is built entirely with Tkinter widgets including Treeview tables, frames, and dialogs, providing responsive and interactive navigation across the application. Robust input handling and validation mechanisms help maintain data consistency, and the database schema is designed with normalization and foreign key constraints to ensure referential integrity. The modular codebase encourages scalability and future integration with features such as order history, product filtering, or web-based deployment. This project serves as a comprehensive example of desktop GUI-based inventory and order management using Python and SQLite.

ABSTRACT WITH POs AND PSOs MAPPING

CO 5 : BUILD DATABASE MANAGEMENT SYSTEM APPLICATION FOR SOLVING REAL-TIME PROBLEMS.

ABSTRACT	POs MAPPED	PSOs MAPPED
The Online Retail Application Management System is a software solution designed to streamline retail operations by integrating customer interaction, inventory control, and sales management into a unified desktop platform. It offers core functionalities such as user authentication, product browsing, cart management, order placement, and administrative control over inventory and sales analytics. The system features a clean, user-friendly graphical interface built with Tkinter, backed by a structured SQLite database for persistent and reliable data storage. It supports role-based access, enabling customers to place and track orders while providing administrators with tools to manage product listings and view sales statistics through interactive graphs. Real-time stock updates, data validation, and secure login mechanisms contribute to operational accuracy and system reliability. The modular structure of the application facilitates future enhancements such as order history tracking, advanced filtering, and web integration. Designed with maintainability and scalability in mind, the system is well-suited for educational, prototyping, or small business use cases where desktop-based retail solutions are needed. Its emphasis on functional clarity and ease of use reduces complexity and enhances overall productivity.	**PO1-3** **PO2-3** **PO3-3** **PO4-3** **PO5-3** **PO6-3** **PO7-3** **PO8-3** **PO9-3** **PO10-3** **PO11-3** **PO12-3**	**PSO1 -3** **PSO2 -3** **PSO3 -3**

Note: 1- Low, 2-Medium, 3- High

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

ABBREVIATIONS

CURD - Create, Read, Update, and Delete

MYSQL - MY Structured Query Language

UI - User Interface

UX - User Experience

API - Application Programming Interface

CHAPTER 1
INTRODUCTION

1.1 Objective

The primary objective of the Online Retail Application Management System is to provide a user-friendly and efficient desktop platform for managing retail operations in a simulated online store environment. By integrating a graphical user interface (GUI) with a local SQLite database, the system aims to centralize product inventory, customer orders, and user account management within a cohesive and accessible application. Key functionalities include customer registration and login, product browsing with real-time stock display, dynamic cart management, and order processing with automatic inventory updates. For administrators, the system offers tools to add new products and visualize sales data through statistical charts. The application emphasizes data accuracy through built-in validation and role-based access to ensure functional integrity. Through these features, the system enhances retail workflow efficiency, supports informed decision-making via analytics, and serves as a practical model for database-driven desktop applications in educational or small business settings.

1.2 Overview

The Online Retail Application Management System is a Python-based desktop application that utilizes Tkinter for its graphical user interface and SQLite for backend data storage. It is designed to simulate a retail environment where users can interact with a product catalog, manage a shopping cart, and place orders. The system supports two primary user roles: customers and administrators. Customers can register, log in, view available products, and complete purchases with real-time stock validation. Administrators have access to a dedicated dashboard where they can add new products and view sales statistics presented through interactive bar charts using Matplotlib. The application implements key CRUD operations for managing users, products, and orders, ensuring smooth retail workflows. Data validation is incorporated to maintain the integrity of inputs such as stock levels and prices. The interface is built with user experience in mind, featuring responsive elements such as Treeviews for product and cart listings, dialog boxes for input, and organized layout structures. SQLite integration ensures reliable data persistence and fast query execution without the need for external database servers. Overall, the system provides a complete and practical solution for managing retail operations in an educational, prototype, or small-scale business context.

1.3 SQLand Database Concepts

Structured Query Language (SQL) plays a central role in the Online Retail Application Management System by enabling structured interaction between the Tkinter GUI and the SQLite database. The system performs various operations like user authentication, product management, and order processing using SQL queries. SQLite is embedded directly within the application using Python's sqlite3 module, making the system efficient, lightweight, and easy to maintain for local use.

1.3.1Database Connection

The application uses Python's sqlite3 module to connect to a local SQLite database file named retail.db. This connection is initialized in the db_init() function during startup. SQL queries are executed through a cursor object, and the database remains open throughout the session. The system automatically creates necessary tables if they don't exist and inserts default admin credentials and sample products to ensure proper system initialization and access.

1.3.2 CRUD Operations

CRUD operations—Create, Read, Update, and Delete—are implemented for managing users, products, orders, and order items. Users can register (Create), login (Read), and place orders, which update stock (Update). Admins can add products using INSERT queries. Customers' carts update stock through UPDATE commands upon checkout. DELETE operations are used when customers remove items from the cart, ensuring the application supports full lifecycle data management as reflected in the GUI.

1.3.3 Data Validation

Data validation is handled within the GUI to ensure database integrity. For example, when admins add products, the price must be a valid number and stock a non-negative integer. During login and registration, fields are checked for emptiness and password confirmation. Customers cannot exceed available stock when adding items to the cart. These checks prevent invalid entries from being committed to the database and align with user input workflows in the app.

1.3.4 Parameterized Queries

To prevent SQL injection, all SQL statements in the system use parameterized queries. Inputs like usernames, passwords, and product data are passed as parameters using placeholders (?) instead of direct string concatenation. For

example, user registration uses a safe INSERT statement with user inputs passed as a tuple. This approach improves security and ensures that user-supplied data does not interfere with the logic or structure of the SQL commands.

1.3.5 Error Handling

The application includes basic but effective error handling for database operations. For example, if a customer attempts to register with an existing username, an sqlite3.IntegrityError is caught, and a friendly error message is displayed using Tkinter's messagebox. This prevents application crashes and ensures the user understands what went wrong. All database changes are committed only after successful operations, preserving both system reliability and data consistency.

1.3.6 SQL Execution

SQL operations follow a clear execution pattern: after establishing a connection, a cursor object runs SQL statements. After each write operation—such as inserting orders or updating stock—self.conn.commit() is used to save changes. Queries like SELECT retrieve product and user data for display in GUI elements like Treeview. This process ensures that user interactions are synchronized with the backend database, supporting real-time updates across the application.

CHAPTER 2

PROJECT METHODOLOGY

2.1 ProposedWork

The proposed work aims to develop a user-friendly Online Retail Application Management System for efficient desktop-based retail operations. Customers can register, log in, browse products, manage carts, and place orders, while administrators can add products and view sales statistics. A centralized application logic interacts with an SQLite database to handle Create, Read, and Update operations. The backend ensures data integrity with structured tables and foreign key constraints linking users, products, orders, and order items. Product stock is updated dynamically at checkout, and transactions are recorded with timestamps and payment status. The admin panel features a Matplotlib-based dashboard for product-wise sales visualization, supporting better decision-making. Input validation ensures accurate data entry, such as stock checks and proper field completion. The modular design allows easy maintenance and future upgrades, while the GUI enhances accessibility for non-technical users. Overall, the system offers a scalable, maintainable, and interactive solution for managing small-scale online retail operations.

2.2 Block Diagram

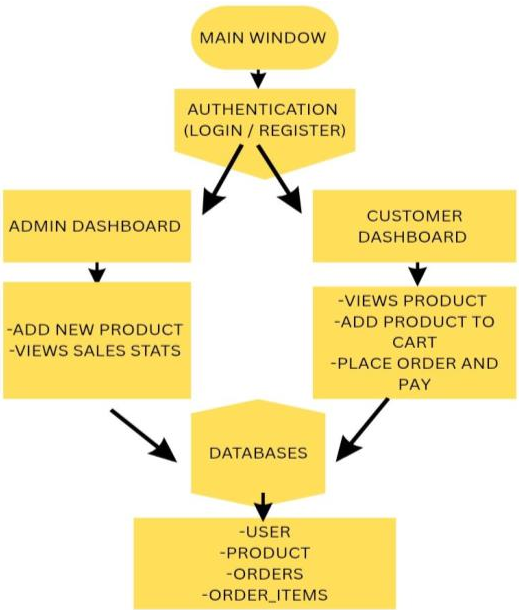


FIG 2.1 BLOCK DIAGRAM

CHAPTER 3
MODULE DESCRIPTION

3.1 User Management Module

This module handles all user-related operations, including registration, login, and role-based access. It allows customers to register and log in, while administrators use default credentials to access advanced features. The module validates user inputs and communicates with the database to verify credentials or create new user accounts. Upon successful login, users are redirected to role-specific dashboards tailored to their access level.

3.2 Product Management Module

The Product Management Module is responsible for displaying available products to customers and enabling admins to add new inventory. Customers can browse products in a Treeview interface, view prices and stock levels, and select items to add to their cart. Administrators can input new product details including name, price, and stock quantity through dedicated form fields, which are then inserted into the database using secure queries.

3.3 Order and Cart Module

This module manages the shopping cart and order processing features. Customers can add products to their cart, specify quantities, and proceed to checkout. The system ensures that selected quantities do not exceed available stock and updates the database upon order confirmation. Orders are stored with a timestamp and payment status. The cart interface allows customers to review, update, or remove items before placing the final order.

3.4 Sales Visualization Module

The Sales Visualization Module is available to administrators and displays statistical sales data through interactive bar charts. It queries the database to calculate the total quantity sold and total revenue per product. Using Matplotlib, this data is rendered into a bar graph within the GUI. The module also includes a data table summarizing product performance, supporting business insights and inventory planning.

3.5 Database Module

This module represents the backend SQLite database, which stores structured information about users, products, orders, and order items. It supports full CRUD operations and uses foreign keys to maintain relationships between tables. The

database is initialized at runtime, creating all necessary tables if they do not exist, and populating default data like admin credentials and products. This module ensures persistent and secure data management.

3.6 Validation and Error Handling Module

This module ensures data accuracy and system stability. It performs input checks such as empty field validation, stock limit enforcement, and type validation (e.g., numeric price and stock). Errors such as duplicate usernames or invalid input values trigger descriptive messages using Tkinter's messagebox. It also uses try-except blocks to catch database-related exceptions, preventing crashes and maintaining a smooth user experience.

CHAPTER 4

CONCLUSION AND FUTURE ENHANCEMENT

4.1 CONCLUSION

In conclusion, the Online Retail Application Management System effectively integrates core retail functionalities into a user-friendly desktop application using Python, Tkinter, and SQLite. It supports customer and admin roles, product browsing, cart management, secure order placement, and real-time sales tracking. The system ensures data consistency through validation and structured database operations, enhancing usability and reliability. Its modular design allows for easy maintenance and future upgrades, making it suitable for educational projects or small business use. Overall, it provides a practical and scalable solution for managing online retail operations efficiently.

4.2 FUTURE ENHANCEMENT

While the application is functional, it can be significantly enhanced in future versions. One major upgrade would be to replace the simulated payment with a real gateway like Stripe or Razorpay. Additionally, an admin panel could be added to manage product listings and user orders. Other improvements include password hashing, user order history, discount coupons, and real-time notifications. Expanding the application to a web or mobile platform would increase accessibility. With cloud database integration and multi-user handling, this system could evolve into a full-fledged commercial solution suitable for small to medium-sized retail businesses.

APPENDIX A
(SOURCE CODE)

```
import tkinter as tk
from tkinter import messagebox, simpledialog, ttk
import sqlite3
import datetime
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
from matplotlib.figure import Figure


class OnlineRetailApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Online Retail Application Management System")
        self.root.configure(bg="#f0f0f0")
        self.db_init()
        self.current_user = None
        self.cart = {}

        # Center the window
        window_width = 1000
        window_height = 600
        screen_width = self.root.winfo_screenwidth()
        screen_height = self.root.winfo_screenheight()
        center_x = int(screen_width/2 - window_width/2)
        center_y = int(screen_height/2 - window_height/2)

self.root.geometry(f'{window_width}x{window_height}+{center_x}+{center_y}')

        self.create_login_screen()
        self.add_default_products()
```

```
    def db_init(self):
        self.conn = sqlite3.connect("retail.db")
        self.cursor = self.conn.cursor()

        # Users: user_id (PK), username, password, role ('admin' or 'customer')
        self.cursor.execute("""
            CREATE TABLE IF NOT EXISTS users (
                user_id INTEGER PRIMARY KEY AUTOINCREMENT,
                username TEXT UNIQUE NOT NULL,
                password TEXT NOT NULL,
                role TEXT NOT NULL CHECK(role IN ('admin', 'customer'))
            )
        """)

        # Products: product_id (PK), name, price, stock_quantity
        self.cursor.execute("""
            CREATE TABLE IF NOT EXISTS products (
                product_id INTEGER PRIMARY KEY AUTOINCREMENT,
                name TEXT NOT NULL,
                price REAL NOT NULL,
                stock_quantity INTEGER NOT NULL
            )
        """)

        # Orders: order_id (PK), user_id (FK), order_date, total_amount,
payment_status
        self.cursor.execute("""
            CREATE TABLE IF NOT EXISTS orders (
                order_id INTEGER PRIMARY KEY AUTOINCREMENT,
                user_id INTEGER NOT NULL,
                order_date TEXT NOT NULL,
                total_amount REAL NOT NULL,
```

```
            payment_status TEXT NOT NULL CHECK(payment_status IN ('paid',
'pending')),
            FOREIGN KEY(user_id) REFERENCES users(user_id)
        )
    """)

    # OrderItems: id (PK), order_id (FK), product_id (FK), quantity, price_each
    self.cursor.execute("""
        CREATE TABLE IF NOT EXISTS order_items (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            order_id INTEGER NOT NULL,
            product_id INTEGER NOT NULL,
            quantity INTEGER NOT NULL,
            price_each REAL NOT NULL,
            FOREIGN KEY(order_id) REFERENCES orders(order_id),
            FOREIGN KEY(product_id) REFERENCES products(product_id)
        )
    """)
    self.conn.commit()

    # Ensure admin user exists with default credentials (admin/admin123)
    self.cursor.execute("SELECT * FROM users WHERE role='admin'")
    admin = self.cursor.fetchone()
    if not admin:
        self.cursor.execute("INSERT INTO users (username, password, role)
VALUES (?, ?, ?)",
                            ("admin", "admin123", "admin"))
        self.conn.commit()

  def add_default_products(self):
    self.cursor.execute("SELECT COUNT(*) FROM products")
    if self.cursor.fetchone()[0] == 0:
```

```
            default_products = [
                ("Laptop", 999.99, 50),
                ("Smartphone", 699.99, 100),
                ("Headphones", 149.99, 200),
                ("Tablet", 399.99, 75),
                ("Smartwatch", 199.99, 150)
            ]
            self.cursor.executemany(
                "INSERT INTO products (name, price, stock_quantity) VALUES (?, ?,
?)",
                default_products
            )
            self.conn.commit()

    def create_login_screen(self):
        for widget in self.root.winfo_children():
            widget.destroy()
        self.root.geometry("350x200")
        self.root.configure(bg="#f0f0f0")

        # Center the login window
        window_width = 350
        window_height = 200
        screen_width = self.root.winfo_screenwidth()
        screen_height = self.root.winfo_screenheight()
        center_x = int(screen_width/2 - window_width/2)
        center_y = int(screen_height/2 - window_height/2)

self.root.geometry(f'{window_width}x{window_height}+{center_x}+{center_y}')

        tk.Label(self.root, text="Online Retail Application", font=("Arial", 16),
bg="#f0f0f0").pack(pady=10)
```

```
        tk.Label(self.root, text="Login", font=("Arial", 12),
bg="#f0f0f0").pack(pady=5)

        tk.Label(self.root, text="Username", bg="#f0f0f0").pack()
        self.login_username = tk.Entry(self.root)
        self.login_username.pack()

        tk.Label(self.root, text="Password", bg="#f0f0f0").pack()
        self.login_password = tk.Entry(self.root, show="*")
        self.login_password.pack()

        tk.Button(self.root, text="Login", command=self.login,
bg="#99ccff").pack(pady=5)
        tk.Button(self.root, text="Register as Customer",
command=self.create_register_screen, bg="#99ff99").pack()
        tk.Button(self.root, text="Logout", command=self.logout,
bg="#ff6666").pack(pady=5)

    def create_register_screen(self):
        for widget in self.root.winfo_children():
            widget.destroy()
        self.root.geometry("350x240")
        self.root.configure(bg="#f0f0f0")

        # Center the register window
        window_width = 350
        window_height = 240
        screen_width = self.root.winfo_screenwidth()
        screen_height = self.root.winfo_screenheight()
        center_x = int(screen_width/2 - window_width/2)
        center_y = int(screen_height/2 - window_height/2)
```

```
self.root.geometry(f'{window_width}x{window_height}+{center_x}+{center_y}')

        tk.Label(self.root, text="Register (Customer)", font=("Arial", 16),
bg="#f0f0f0").pack(pady=10)

        tk.Label(self.root, text="Username", bg="#f0f0f0").pack()
        self.reg_username = tk.Entry(self.root)
        self.reg_username.pack()

        tk.Label(self.root, text="Password", bg="#f0f0f0").pack()
        self.reg_password = tk.Entry(self.root, show="*")
        self.reg_password.pack()

        tk.Label(self.root, text="Confirm Password", bg="#f0f0f0").pack()
        self.reg_confirm_password = tk.Entry(self.root, show="*")
        self.reg_confirm_password.pack()

        tk.Button(self.root, text="Register", command=self.register_customer,
bg="#99ff99").pack(pady=10)
        tk.Button(self.root, text="Back to Login", command=self.create_login_screen,
bg="#ffcc99").pack()
        tk.Button(self.root, text="Logout", command=self.logout,
bg="#ff6666").pack(pady=5)

    def login(self):
        username = self.login_username.get().strip()
        password = self.login_password.get().strip()
        if not username or not password:
            messagebox.showerror("Error", "Please enter both username and
password.")
            return
```

```
        self.cursor.execute("SELECT user_id, role, password FROM users WHERE
username=?", (username,))
        row = self.cursor.fetchone()
        if row and row[2] == password:
            self.current_user = {"user_id": row[0], "username": username, "role":
row[1]}
            if row[1] == "admin":
                self.create_admin_dashboard()
            else:
                self.create_customer_dashboard()
        else:
            messagebox.showerror("Error", "Invalid username or password.")

    def register_customer(self):
        username = self.reg_username.get().strip()
        password = self.reg_password.get().strip()
        confirm_password = self.reg_confirm_password.get().strip()

        if not username or not password or not confirm_password:
            messagebox.showerror("Error", "Please fill all fields.")
            return
        if password != confirm_password:
            messagebox.showerror("Error", "Passwords do not match.")
            return
        try:
            self.cursor.execute("INSERT INTO users (username, password, role)
VALUES (?, ?, ?)",
                                (username, password, "customer"))
            self.conn.commit()
            messagebox.showinfo("Success", "Registration successful! Please login.")
            self.create_login_screen()
        except sqlite3.IntegrityError:
```

```
            messagebox.showerror("Error", "Username already exists. Choose
another.")

    def create_customer_dashboard(self):
        self.cart = {}
        for widget in self.root.winfo_children():
            widget.destroy()
        self.root.geometry("1000x600")
        self.root.configure(bg="#f0f0f0")

        header_frame = tk.Frame(self.root, bg="#f0f0f0")
        header_frame.pack(fill=tk.X, pady=5)
        tk.Label(header_frame, text=f"Welcome, {self.current_user['username']}
(Customer)",
                 font=("Arial", 14), bg="#f0f0f0").pack(side=tk.LEFT, padx=10)
        btn_back = tk.Button(header_frame, text="Back",
command=self.create_login_screen, bg="#ffcc99")
        btn_back.pack(side=tk.RIGHT, padx=5)
        btn_logout = tk.Button(header_frame, text="Logout", command=self.logout,
bg="#ff6666")
        btn_logout.pack(side=tk.RIGHT, padx=5)

        # Main content frame
        main_frame = tk.Frame(self.root, bg="#f0f0f0")
        main_frame.pack(fill=tk.BOTH, expand=True, padx=20, pady=10)

        # Product list frame
        product_frame = tk.Frame(main_frame, bg="#e6f2ff", bd=2,
relief=tk.GROOVE)
        product_frame.pack(side=tk.LEFT, fill=tk.BOTH, expand=True, padx=10,
pady=10)
```

```
        tk.Label(product_frame, text="Available Products", font=("Arial", 12),
bg="#e6f2ff").pack()

        self.product_tree = ttk.Treeview(product_frame, columns=("name", "price",
"stock"), show="headings", height=20)
        self.product_tree.heading("name", text="Product Name")
        self.product_tree.heading("price", text="Price")
        self.product_tree.heading("stock", text="In Stock")
        self.product_tree.column("name", width=200, anchor=tk.W)
        self.product_tree.column("price", width=100, anchor=tk.CENTER)
        self.product_tree.column("stock", width=100, anchor=tk.CENTER)
        self.product_tree.pack(fill=tk.BOTH, expand=True)

        self.product_tree.bind("<Double-1>", self.add_product_to_cart_dialog)

        tk.Label(product_frame, text="Double-click a product to add to cart",
bg="#e6f2ff").pack(pady=5)

        # Cart frame
        cart_frame = tk.Frame(main_frame, bg="#ffe6e6", bd=2, relief=tk.GROOVE)
        cart_frame.pack(side=tk.RIGHT, fill=tk.BOTH, expand=True, padx=10,
pady=10)

        tk.Label(cart_frame, text="Your Cart", font=("Arial", 12),
bg="#ffe6e6").pack()
        self.cart_tree = ttk.Treeview(cart_frame, columns=("name", "quantity",
"price_each", "total"), show="headings", height=15)
        self.cart_tree.heading("name", text="Product")
        self.cart_tree.heading("quantity", text="Quantity")
        self.cart_tree.heading("price_each", text="Price Each")
        self.cart_tree.heading("total", text="Total Price")
        self.cart_tree.column("name", width=150, anchor=tk.W)
```

```
        self.cart_tree.column("quantity", width=75, anchor=tk.CENTER)
        self.cart_tree.column("price_each", width=100, anchor=tk.CENTER)
        self.cart_tree.column("total", width=100, anchor=tk.CENTER)
        self.cart_tree.pack(fill=tk.BOTH, expand=True)

        btn_frame = tk.Frame(cart_frame, bg="#ffe6e6")
        btn_frame.pack(pady=10)

        tk.Button(btn_frame, text="Remove Selected Item",
command=self.remove_selected_cart_item, bg="#ff9999").pack(side=tk.LEFT,
padx=5)
        tk.Button(btn_frame, text="Place Order & Pay", command=self.place_order,
bg="#99ff99").pack(side=tk.LEFT, padx=5)

        self.load_products()

    def load_products(self):
        for row in self.product_tree.get_children():
            self.product_tree.delete(row)
        self.cursor.execute("SELECT product_id, name, price, stock_quantity FROM
products")
        for product_id, name, price, stock in self.cursor.fetchall():
            self.product_tree.insert("", tk.END, iid=str(product_id),
                                     values=(name, f"${price:.2f}", stock))

    def add_product_to_cart_dialog(self, event):
        selected_item = self.product_tree.focus()
        if not selected_item:
            return
        product_id = int(selected_item)
        item_data = self.product_tree.item(selected_item)
        name = item_data['values'][0]
```

```
        price = float(item_data['values'][1].replace('$', ''))
        stock = int(item_data['values'][2])

        if stock <= 0:
            messagebox.showinfo("Out of Stock", f"The product '{name}' is out of
stock.")
            return

        quantity = simpledialog.askinteger("Quantity", f"Enter quantity for '{name}'
(max {stock}):",
                                           minvalue=1, maxvalue=stock)
        if quantity is None:
            return

        if product_id in self.cart:
            new_qty = self.cart[product_id]["quantity"] + quantity
            if new_qty > stock:
                messagebox.showerror("Error", "Quantity exceeds available stock.")
                return
            self.cart[product_id]["quantity"] = new_qty
        else:
            self.cart[product_id] = {"name": name, "price": price, "quantity": quantity}
        self.load_cart()

    def load_cart(self):
        for row in self.cart_tree.get_children():
            self.cart_tree.delete(row)
        for pid, item in self.cart.items():
            total_price = item["price"] * item["quantity"]
            self.cart_tree.insert("", tk.END, iid=str(pid),
                                  values=(item["name"], item["quantity"],
                                          f"${item['price']:.2f}", f"${total_price:.2f}"))
```

```
    def remove_selected_cart_item(self):
        selected = self.cart_tree.focus()
        if not selected:
            messagebox.showinfo("Info", "Please select an item to remove.")
            return
        pid = int(selected)
        if pid in self.cart:
            del self.cart[pid]
            self.load_cart()

    def place_order(self):
        if not self.cart:
            messagebox.showerror("Error", "Your cart is empty.")
            return

        # Check stock availability again before placing order
        for pid, item in self.cart.items():
            self.cursor.execute("SELECT stock_quantity FROM products WHERE
product_id=?", (pid,))
            stock = self.cursor.fetchone()[0]
            if item["quantity"] > stock:
                messagebox.showerror("Error", f"Insufficient stock for '{item['name']}'.
Available: {stock}")
                return

        total_amount = sum(item["price"] * item["quantity"] for item in
self.cart.values())
        confirm = messagebox.askyesno("Confirm Order", f"Total amount:
${total_amount:.2f}\nProceed to pay?")
        if not confirm:
            return
```

```
        # Insert order
        order_date = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        self.cursor.execute(
            "INSERT INTO orders (user_id, order_date, total_amount, payment_status)
VALUES (?, ?, ?, ?)",
            (self.current_user["user_id"], order_date, total_amount, "paid"))
        order_id = self.cursor.lastrowid

        # Insert order items and update stock
        for pid, item in self.cart.items():
            self.cursor.execute(
                "INSERT INTO order_items (order_id, product_id, quantity, price_each)
VALUES (?, ?, ?, ?)",
                (order_id, pid, item["quantity"], item["price"])
            )
            self.cursor.execute(
                "UPDATE products SET stock_quantity = stock_quantity - ? WHERE
product_id = ?",
                (item["quantity"], pid)
            )

        self.conn.commit()
        messagebox.showinfo("Success", "Order placed and payment done
successfully!")
        self.cart = {}
        self.load_cart()
        self.load_products()

    def create_admin_dashboard(self):
        for widget in self.root.winfo_children():
            widget.destroy()
```

```
        self.root.geometry("1000x600")
        self.root.configure(bg="#f0f0f0")

        header_frame = tk.Frame(self.root, bg="#f0f0f0")
        header_frame.pack(fill=tk.X, pady=5)
        tk.Label(header_frame, text=f"Welcome, {self.current_user['username']} (Admin)",
                 font=("Arial", 14), bg="#f0f0f0").pack(side=tk.LEFT, padx=10)
        btn_back = tk.Button(header_frame, text="Back", command=self.create_login_screen, bg="#ffcc99")
        btn_back.pack(side=tk.RIGHT, padx=5)
        btn_logout = tk.Button(header_frame, text="Logout", command=self.logout, bg="#ff6666")
        btn_logout.pack(side=tk.RIGHT, padx=5)

        # Tab control for admin
        tab_control = ttk.Notebook(self.root)
        tab_control.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

        # Tab 1: Add product
        add_product_tab = ttk.Frame(tab_control)
        tab_control.add(add_product_tab, text="Add New Product")

        tk.Label(add_product_tab, text="Product Name").grid(row=0, column=0, pady=5, padx=10, sticky=tk.E)
        self.admin_prod_name = tk.Entry(add_product_tab, width=40)
        self.admin_prod_name.grid(row=0, column=1, pady=5, sticky=tk.W)

        tk.Label(add_product_tab, text="Price ($)").grid(row=1, column=0, pady=5, padx=10, sticky=tk.E)
        self.admin_prod_price = tk.Entry(add_product_tab, width=20)
        self.admin_prod_price.grid(row=1, column=1, pady=5, sticky=tk.W)
```

```
        tk.Label(add_product_tab, text="Stock Quantity").grid(row=2, column=0,
pady=5, padx=10, sticky=tk.E)
        self.admin_prod_stock = tk.Entry(add_product_tab, width=20)
        self.admin_prod_stock.grid(row=2, column=1, pady=5, sticky=tk.W)

        tk.Button(add_product_tab, text="Add Product",
command=self.admin_add_product,
                  bg="#99ff99").grid(row=3, column=1, pady=10, sticky=tk.W)

        # Tab 2: View product sales statistics with graph
        stats_tab = ttk.Frame(tab_control)
        tab_control.add(stats_tab, text="Product Sales Statistics")

        # Create a frame for the graph
        graph_frame = tk.Frame(stats_tab)
        graph_frame.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

        # Create a frame for the table
        table_frame = tk.Frame(stats_tab)
        table_frame.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

        self.stats_tree = ttk.Treeview(table_frame, columns=("total_quantity",
"total_sales"),
                                       show="headings", height=10)
        self.stats_tree.heading("total_quantity", text="Total Quantity Sold")
        self.stats_tree.heading("total_sales", text="Total Sales ($)")
        self.stats_tree.column("total_quantity", width=150, anchor=tk.CENTER)
        self.stats_tree.column("total_sales", width=150, anchor=tk.CENTER)
        self.stats_tree.pack(fill=tk.BOTH, expand=True)

        # Button to refresh statistics
```

```
        btn_refresh = tk.Button(stats_tab, text="Refresh Statistics",
                                command=self.load_statistics, bg="#99ccff")
        btn_refresh.pack(pady=5)

        # Initialize graph
        self.figure = Figure(figsize=(8, 4), dpi=100)
        self.plot = self.figure.add_subplot(111)
        self.canvas = FigureCanvasTkAgg(self.figure, master=graph_frame)
        self.canvas.get_tk_widget().pack(fill=tk.BOTH, expand=True)

        self.load_statistics()

    def admin_add_product(self):
        name = self.admin_prod_name.get().strip()
        price_str = self.admin_prod_price.get().strip()
        stock_str = self.admin_prod_stock.get().strip()

        if not name or not price_str or not stock_str:
            messagebox.showerror("Error", "Please enter product name, price and stock
quantity.")
            return
        try:

            price = float(price_str)
            stock = int(stock_str)
            if price < 0 or stock < 0:
                raise ValueError
        except ValueError:
            messagebox.showerror("Error", "Price must be positive number and stock
must be a positive integer.")
            return
```

```
    self.cursor.execute(
        "INSERT INTO products (name, price, stock_quantity) VALUES (?, ?, ?)",
        (name, price, stock)
    )
    self.conn.commit()
    messagebox.showinfo("Success", f"Product '{name}' added successfully!")
    self.admin_prod_name.delete(0, tk.END)
    self.admin_prod_price.delete(0, tk.END)
    self.admin_prod_stock.delete(0, tk.END)

def load_statistics(self):
    # Clear existing data
    for row in self.stats_tree.get_children():
        self.stats_tree.delete(row)
    self.plot.clear()

    # Get sales statistics: total quantity sold and total sales per product
    self.cursor.execute("""
        SELECT p.name,
               IFNULL(SUM(oi.quantity), 0) AS total_quantity,
               IFNULL(SUM(oi.quantity * oi.price_each), 0) AS total_sales
        FROM products p
        LEFT JOIN order_items oi ON p.product_id = oi.product_id
        GROUP BY p.name
        ORDER BY total_sales DESC
    """)
    stats_data = self.cursor.fetchall()

    if not stats_data:
        return

    # Prepare data for the table and graph
```

```
        product_names = []
        quantities = []
        sales = []

        for name, qty, sales_amount in stats_data:
            self.stats_tree.insert("", tk.END, values=(qty, f"${sales_amount:.2f}"),
text=name)
            product_names.append(name)
            quantities.append(qty)
            sales.append(sales_amount)

        # Create bar graph
        x = range(len(product_names))
        bars = self.plot.bar(x, sales, color='skyblue')
        self.plot.set_title('Product Sales Performance')
        self.plot.set_xlabel('Products')
        self.plot.set_ylabel('Total Sales ($)')
        self.plot.set_xticks(x)
        self.plot.set_xticklabels(product_names, rotation=45, ha='right')

        # Add value labels on top of bars
        for bar in bars:
            height = bar.get_height()
            self.plot.text(bar.get_x() + bar.get_width()/2., height,
                    f'${height:.2f}',
                    ha='center', va='bottom')

        self.figure.tight_layout()
        self.canvas.draw()

    def logout(self):
        self.current_user = None
```

```
        self.cart = {}
        self.create_login_screen()


if __name__ == "__main__":
    root = tk.Tk()
    app = OnlineRetailApp(root)
    root.mainloop()
```

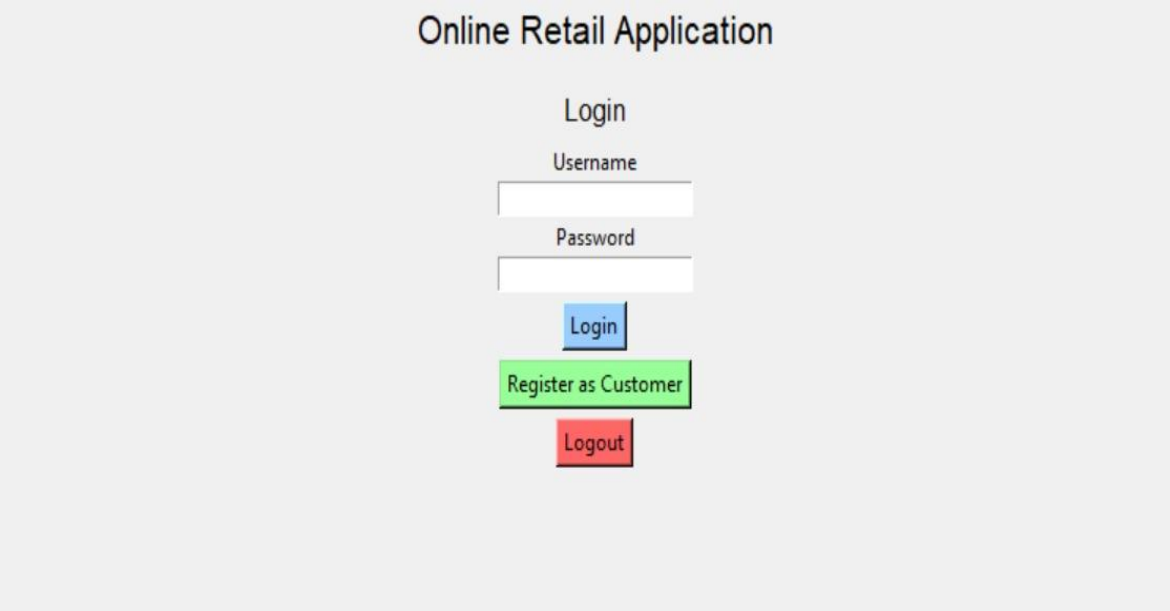
SQL Code

```
-- Users table: stores login info and roles
CREATE TABLE IF NOT EXISTS users (
    user_id INTEGER PRIMARY KEY AUTOINCREMENT,
    username TEXT UNIQUE NOT NULL,
    password TEXT NOT NULL,
    role TEXT NOT NULL CHECK(role IN ('admin', 'customer'))
);

-- Products table: item listings
CREATE TABLE IF NOT EXISTS products (
    product_id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL,
    price REAL NOT NULL,
    stock_quantity INTEGER NOT NULL
);

-- Orders table: records of purchases
CREATE TABLE IF NOT EXISTS orders (
    order_id INTEGER PRIMARY KEY AUTOINCREMENT,
    user_id INTEGER NOT NULL,
    order_date TEXT NOT NULL,
    total_amount REAL NOT NULL,
```

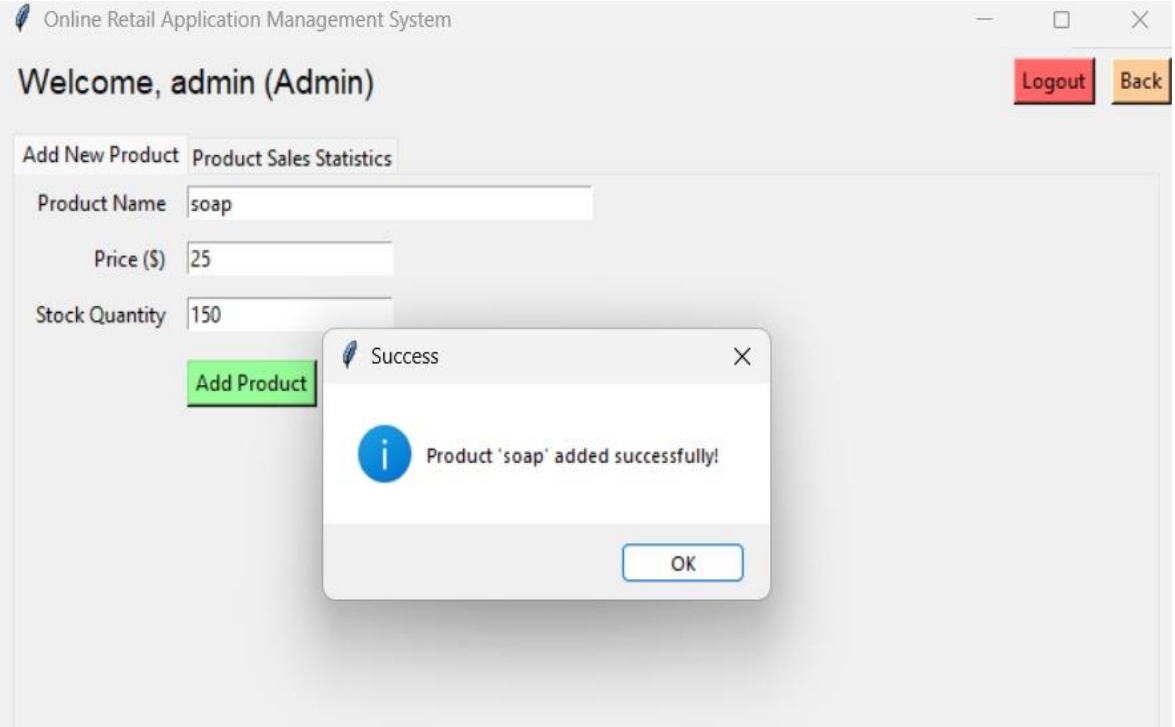
```
    payment_status TEXT NOT NULL CHECK(payment_status IN ('paid',
'pending')),
    FOREIGN KEY(user_id) REFERENCES users(user_id)
);

-- Order Items table: details of products in each order
CREATE TABLE IF NOT EXISTS order_items (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    order_id INTEGER NOT NULL,
    product_id INTEGER NOT NULL,
    quantity INTEGER NOT NULL,
    price_each REAL NOT NULL,
    FOREIGN KEY(order_id) REFERENCES orders(order_id),
    FOREIGN KEY(product_id) REFERENCES products(product_id)
);
```

APPENDIX B
(SCREENSHOT)

1. LOGIN PAGE

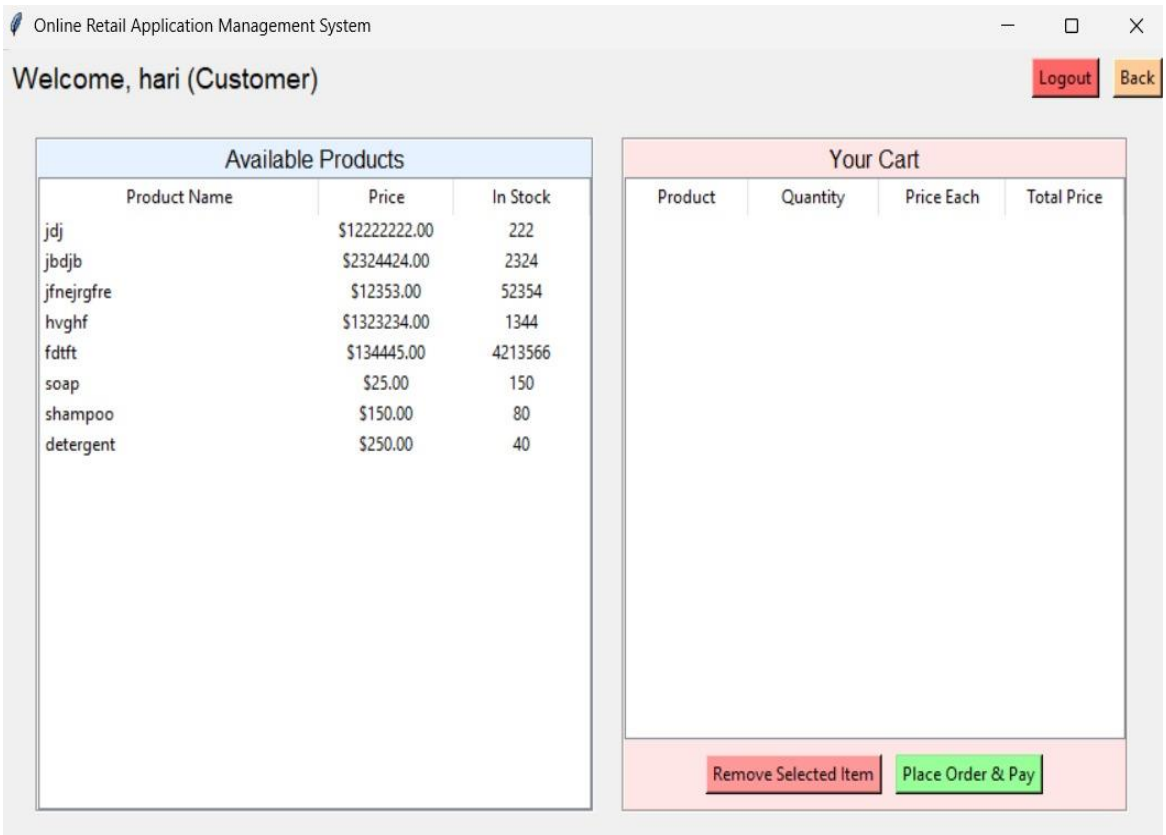


2. LOGIN AS ADMIN AND ADD NEW PRODUCT

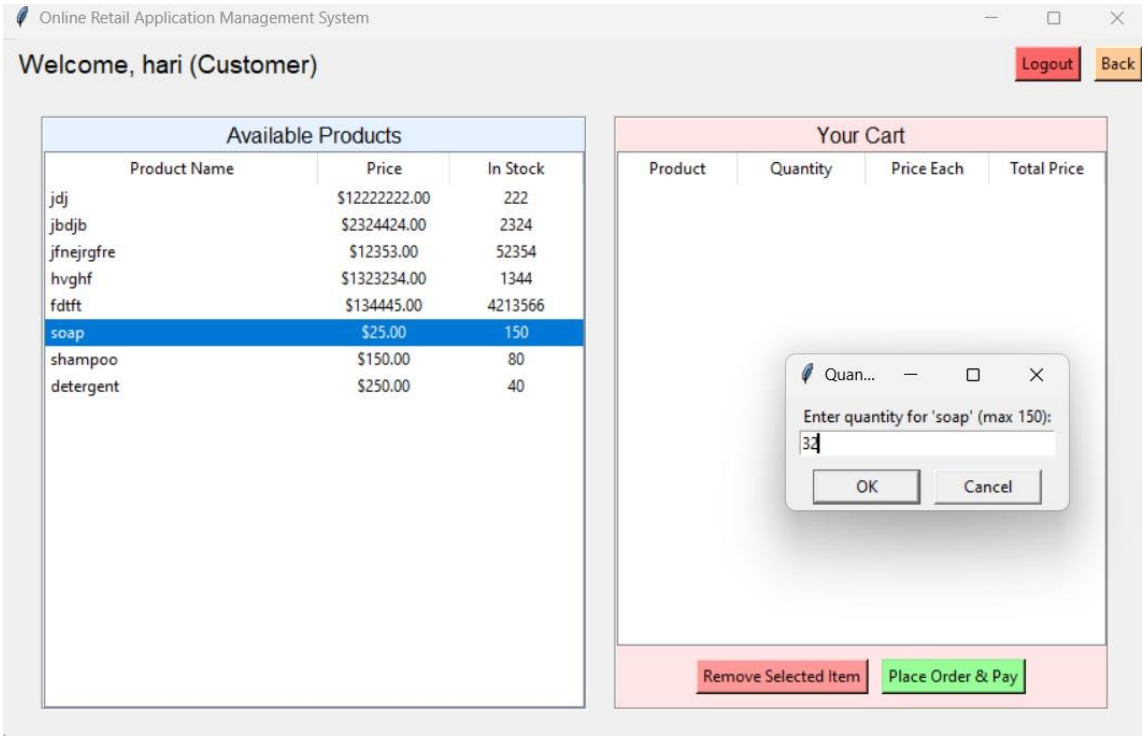


3. CUSTOMER LOGIN PAGE

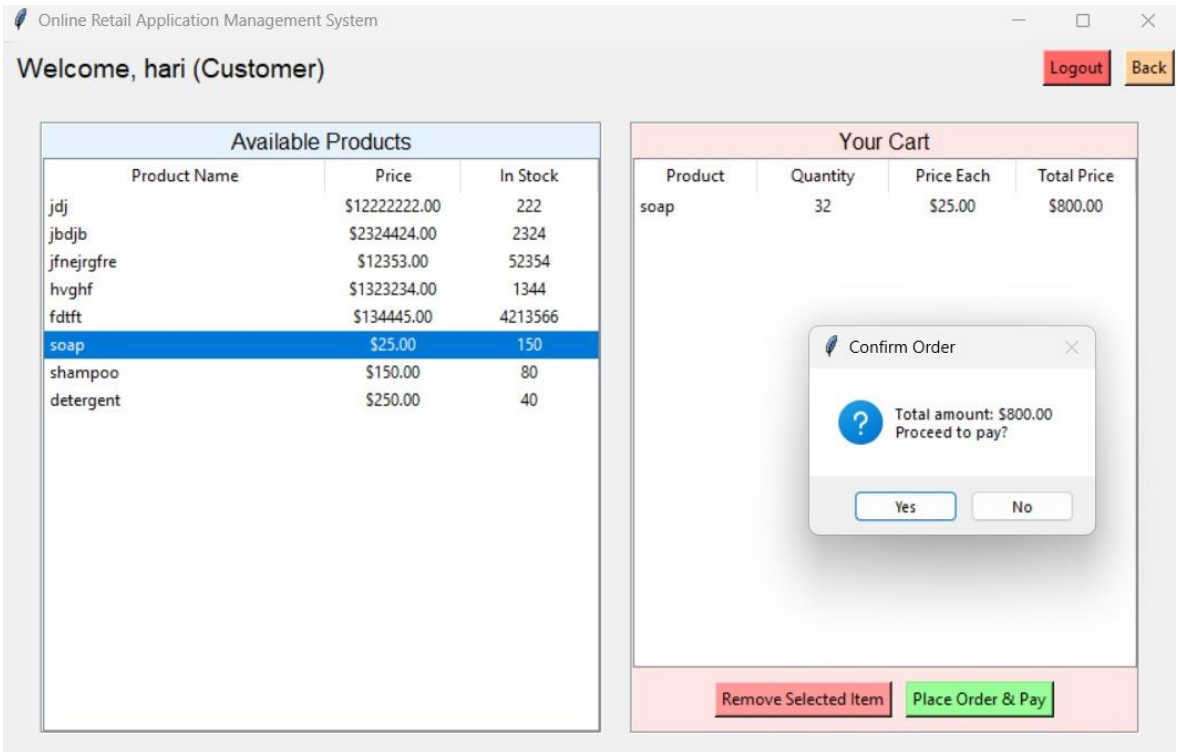
4. AVAILABLE PRODUCT LIST



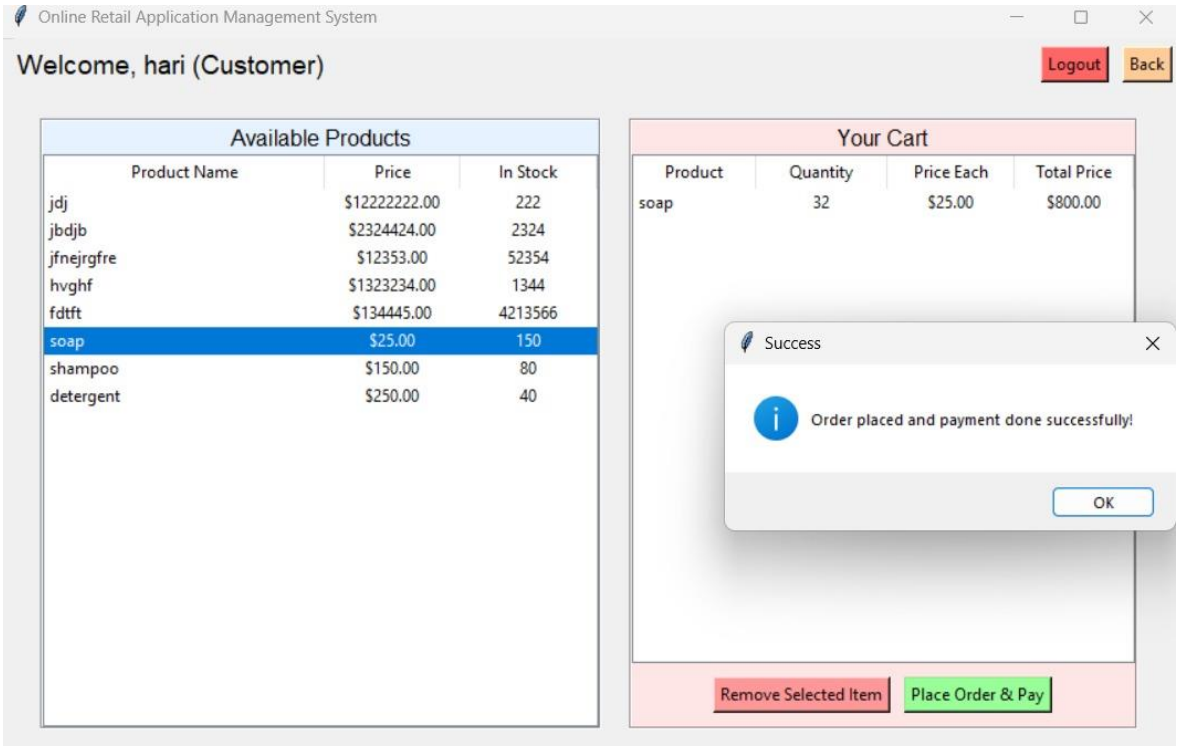
5. ADD PRODUCT TO CART



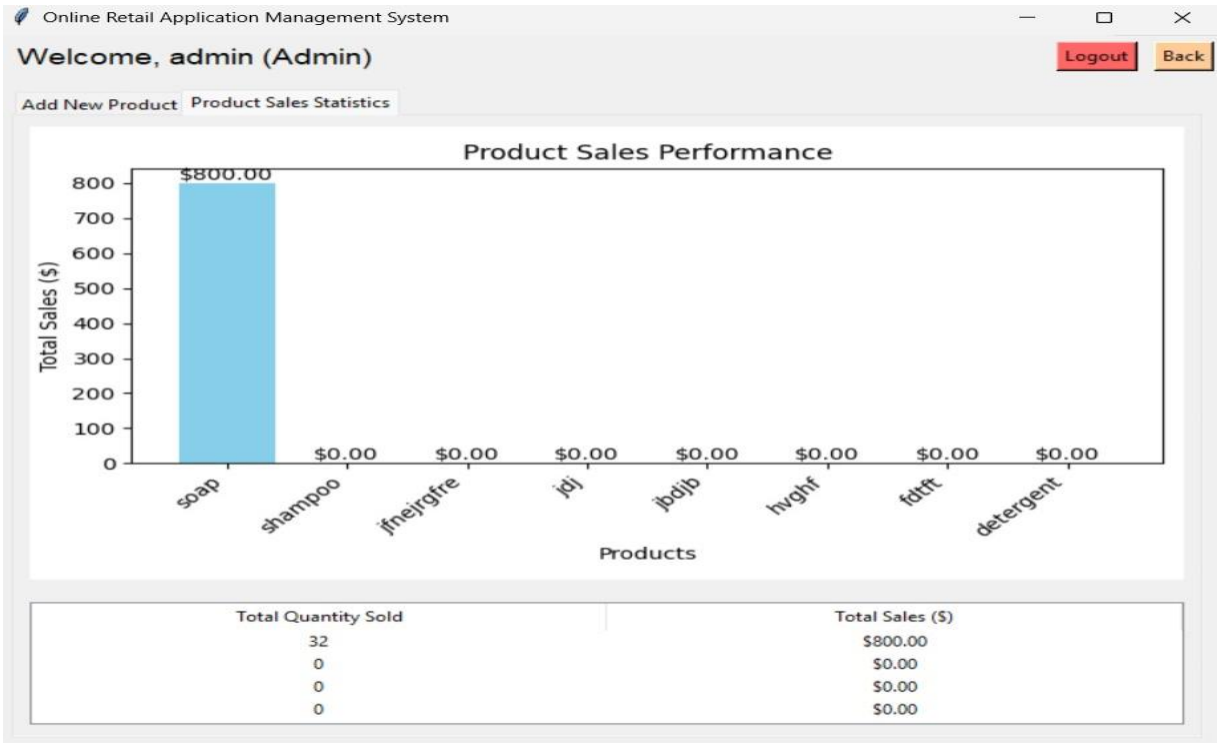
6. ORDER CONFIRMATION



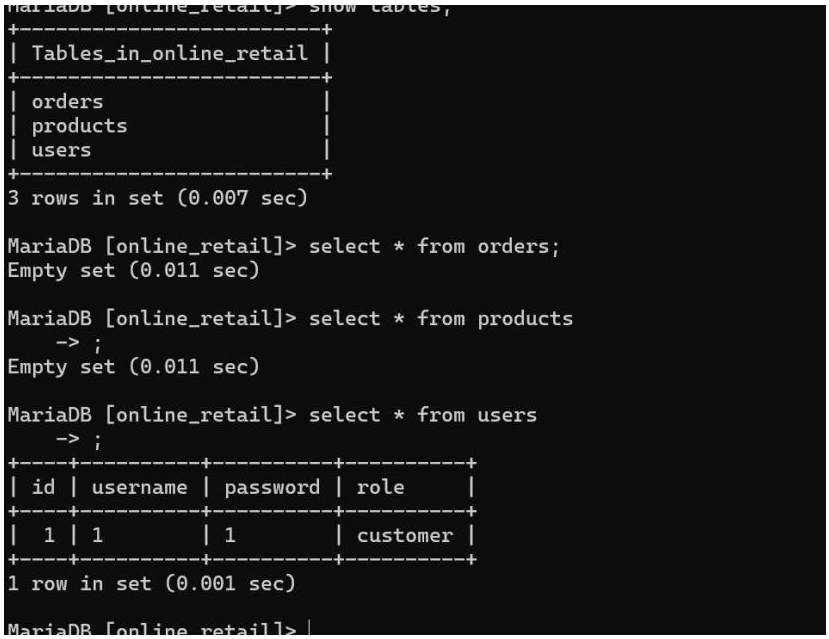
7. ORDER PROCESSED AND PAYMENT



8. ADMIN VIEWING SALES STATS



9.DATABASE FOR RETAIL APPLICATION



```
MariaDB [online_retail]> show tables;
+------------------------+
| Tables_in_online_retail |
+------------------------+
| orders                 |
| products               |
| users                  |
+------------------------+
3 rows in set (0.007 sec)

MariaDB [online_retail]> select * from orders;
Empty set (0.011 sec)

MariaDB [online_retail]> select * from products
    -> ;
Empty set (0.011 sec)

MariaDB [online_retail]> select * from users
    -> ;
+----+----------+----------+----------+
| id | username | password | role     |
+----+----------+----------+----------+
|  1 | 1        | 1        | customer |
+----+----------+----------+----------+
1 row in set (0.001 sec)

MariaDB [online_retail]>
```

REFERENCES

1. Python Official Documentation – for Tkinter GUI development and standard libraries https://docs.python.org/3/
2. MySQL Official Documentation – for database design, schema creation, and SQL queries https://dev.mysql.com/doc/
3. W3Schools SQL Tutorial – for SQL syntax, normalization, and CRUD operations https://www.w3schools.com/sql/
4. GeeksforGeeks – Python Tkinter Tutorials – for designing the GUI components and handling events https://www.geeksforgeeks.org/python-gui-tkinter/
5. GitHub Repository: Online Shopping System Using Tkinter and MySQL https://github.com/VaishnaviVV/Online-Shopping-system-using-Tkinter-and-SQL
6. YouTube Tutorial: Python Tkinter Project with MySQL Database https://www.youtube.com/watch?v=SOU4TubaDf0
7. MySQL Connector/Python Documentation – for Python-MySQL integration and executing SQL commands secure https://dev.mysql.com/doc/connector-python/en/