

RETAIL POINT SALE SYSTEM



A PROJECT REPORT

Submitted by

NIRANJAN KUMAR M (2303811710421107)

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 – 112 621

JUNE- 2025

**K. RAMAKRISHNAN COLLEGE OF TECHNOLOGY
(AUTONOMOUS)**

SAMAYAPURAM – 112 621

BONAFIDE CERTIFICATE

Certified that this project report on “RETAIL POINT SALE SYSTEM ” is the bonafide work of **NIRANJAN KUMAR M (230381171024107)** who carried out the project work during the academic year 2025 - 2024 under my supervision.



SIGNATURE

Mrs.A.DELPHIN CAROLINA RANI,
M.E.,Ph.D.,

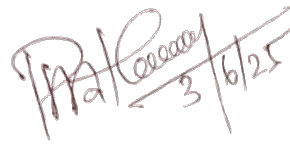
HEAD OF THE DEPARTMENT

PROFESSOR

Department of CSE

K.Ramakrishnan College of Technology
(Autonomous)

Samayapuram-.621112



SIGNATURE

Mr.P.MATHESWARAN, M.E.,Ph.D.,

SUPERVISOR

ASSISTANT PROFESSOR

Department of CSE

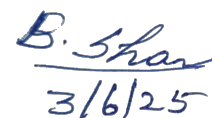
K.Ramakrishnan College of Technology
(Autonomous)

Samayapuram-.621112

Submitted for the viva-voce examination held on 03/06/2025



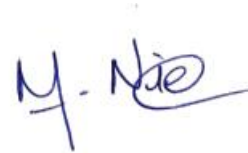
INTERNAL EXAMINER



EXTERNAL EXAMINER

DECLARATION

I declare that the project report on “**RETAIL POINT SALE 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

Niranjan kumar M

Place: Samayapuram

Date: 03/06/2025

ACKNOWLEDGEMENT

It is with great pride that I express our 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 our 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 our deep expression and sincere gratitude to our project supervisor **Mr.P.MATHESWARAN, M.E.,Ph.D.**, Department of **COMPUTER SCIENCE AND ENGINEERING**, for his incalculable suggestions, creativity, assistance and patience which motivated us to carry out this project.

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

I wish to express our 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:

Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

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

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

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

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

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

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

Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

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.

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.

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 Retail Point of Sale (POS) System is a user-friendly desktop application designed to streamline retail operations by managing product inventory and processing sales efficiently. Built using Python with a Tkinter graphical user interface and SQLite database backend, the system allows shopkeepers to add, update, and delete product details, as well as perform sales transactions with quantity tracking and automatic stock updates. It features a clean full-screen interface for optimal usability, providing a smooth experience for both small and medium-scale retail environments. The system also maintains a detailed sales history, allowing store managers to monitor product movement and analyze sales performance. This application offers an all-in-one solution for managing retail transactions without the need for expensive commercial software. Its design emphasizes simplicity, clarity, and effectiveness, making it suitable for users with minimal technical knowledge. By integrating inventory management and billing into a single interface, the POS system significantly reduces manual workload, improves accuracy in sales records, and helps maintain up-to-date stock information.

ABSTRACT WITH POs AND PSOs MAPPING

CO 5 : BUILD JAVA APPLICATIONS FOR SOLVING REAL-TIME PROBLEMS.

ABSTRACT	POs MAPPED	PSOs MAPPED
<p>This project is a desktop-based Retail Point of Sale (POS) system developed using Python and MySQL. It allows users to manage product inventory, perform sales transactions, and view sales history through a user-friendly Tkinter interface. A Flask-based API is also integrated to provide remote access to product and sales data. The project demonstrates practical skills in real-time application development, database integration, and user interface design.</p>	<p>PO3- 1 PO3- 2 PO3- 3 PO3- 4 PO3- 5 PO3- 6 PO3- 7 PO3- 8 PO3- 9 PO3- 10 PO3-11 PO3- 12</p>	<p>PSO3- 1 PSO3- 2 PSO3- 3</p>

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

TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO.
	ABSTRACT	VIII
1	INTRODUCTION	1
	1.1 Objective	1
	1.2 Overview	1
	1.3 SQL and Database concepts	2
2	PROJECT METHODOLOGY	3
	2.1 Proposed Work	3
	2.2 Block Diagram	3
3	MODULE DESCRIPTION	4
	3.1 Product Management Module	4
	3.2 Sales Management Module	4
	3.3 Database Management Module	4
	3.4 Backend Api Module	4
	3.5 User Interface Module	5
4	CONCLUSION & FUTURE SCOPE	6
5	APPENDIX A SOURCE CODE	7
6	APPENDIX B SCREENSHOTS	15
	REFERENCES	22

CHAPTER 1

INTRODUCTION

1.1 OBJECTIVE

The objective of the Retail Point of Sale (POS) System is to develop a comprehensive and efficient software application tailored for small and medium-sized retail businesses. It is designed to simplify the complexities of daily sales and inventory tracking by integrating all essential retail functions into a single platform.

The system supports operations such as:

- Adding, updating, and deleting product records
- Managing and updating stock levels in real time
- Recording sales transactions and calculating totals
- Generating detailed billing and receipts for each transaction
- Maintaining sales history and enabling quick data retrieval

The goal is to reduce manual paperwork, prevent stock-related errors, and ensure a faster, more accurate checkout process. This results in enhanced customer satisfaction, improved staff productivity, and better data-driven decision-making by store owners.

1.2 OVERVIEW

The Retail POS System is a desktop-based application built using Python and integrated with a MySQL database. It leverages three core technologies:

- **Flask** (Backend): Handles the logic for processing data, API handling, and communication between the GUI and database.
- **MySQL** (Database): Stores and organizes all data including product information, sales history, and stock levels.
- **Tkinter** (Frontend): Provides a clean and interactive GUI through which users can perform various tasks.

The system's key features include:

- **Product Management:** Add, update, and delete product entries.
- **Inventory Tracking:** Automatically adjust stock levels after each sale.
- **Sales Processing:** Perform sales transactions, calculate totals, apply discounts, and generate receipts.
- **Transaction History:** View past sales by date, product, or user.
- **Data Validation:** Ensure data accuracy by validating user input and preventing duplication or errors.

1.3 SQL AND DATABASE CONCEPTS

Structured Query Language (SQL) plays a central role in data handling within the POS system. Using SQL, the application performs various operations on the database such as creating tables, inserting new data, updating stock quantities, and generating queries for reports. The MySQL database ensures data integrity, fast retrieval, and secure storage.

Core Concepts:

- **Tables:** Logical structures to organize and store data (e.g., products, sales).
- **Primary Key:** A unique field (such as `product_id`) that identifies each record in a table.
- **Foreign Key:** A field that establishes relationships between tables (e.g., linking sales to products).
- **Normalization:** A design process used to eliminate data redundancy and improve data consistency.
- **Queries:** SQL commands used to retrieve and manipulate data, such as:

```
SELECT * FROM sales WHERE date = '2025-05-01';
```

```
UPDATE products SET stock = stock - 1 WHERE product_id = 101;
```

Project-Specific Tables:

- **products:**
Columns: `product_id` (PK), `name`, `price`, `stock_quantity`, `category`
Function: Manages product-related data used in transactions and inventory tracking.
- **sales:**
Columns: `sale_id` (PK), `product_id` (FK), `quantity_sold`, `sale_date`, `total_amount`
Function: Logs each transaction, useful for reports and inventory updates.

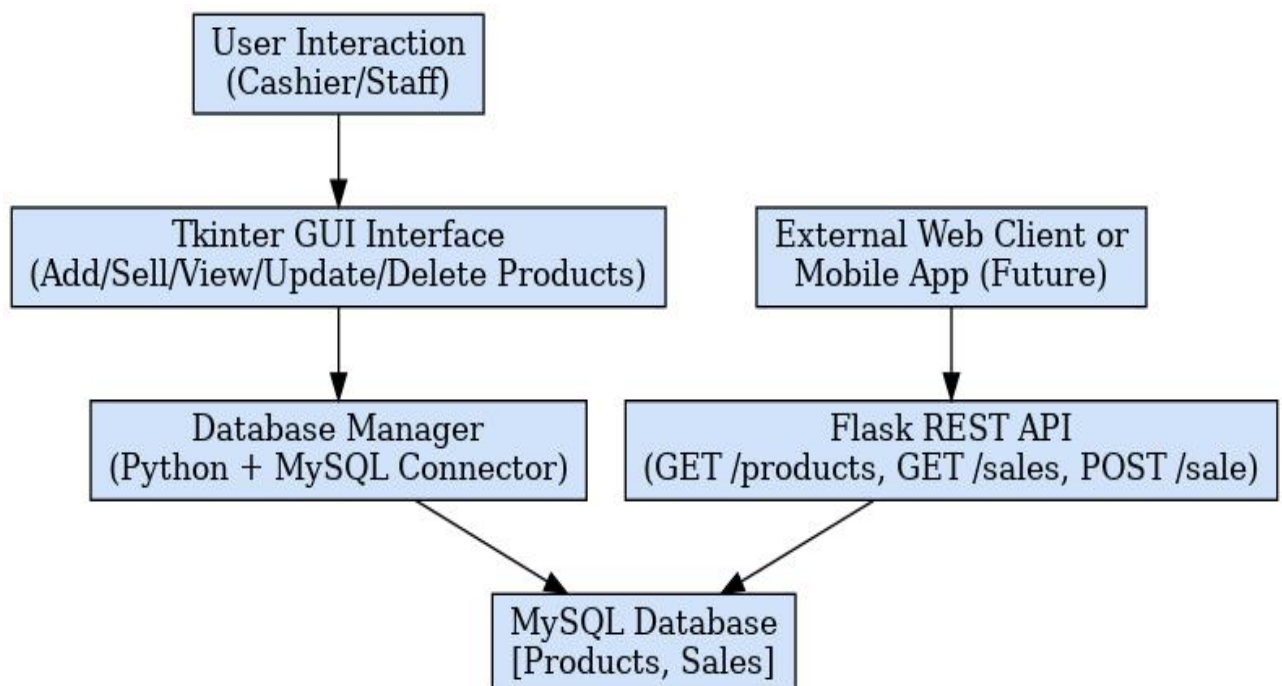
CHAPTER 2

PROJECT METHODOLOGY

2.1 Proposed Work

The proposed Retail POS System offers a comprehensive solution for managing day-to-day retail operations by integrating product entry, inventory tracking, and sales processing into a single, user-friendly platform. It is developed using Python's Tkinter library for the frontend, Flask for the backend server logic, and MySQL for robust and reliable data storage. The system allows users to add and update product details, process sales transactions, generate receipts, and automatically manage stock levels in real time. By maintaining accurate sales and inventory records, the system minimizes manual errors, reduces operational delays, and supports quick decision-making through data analytics. Its modular architecture ensures ease of development, testing, and debugging, while also providing a strong foundation for future enhancements like user role management, barcode scanning, mobile integration, and cloud-based access. The system ultimately aims to enhance business efficiency, customer service, and the overall retail experience for small to medium-sized enterprises.

2.2 Block Diagram



CHAPTER 3

MODULE DESCRIPTION

3.1 PRODUCT MANAGEMENT MODULE

This module manages all product-related operations. Users can add new products, edit existing product details, and view product listings. Each product entry includes attributes such as product name, price, stock quantity, category, and a unique identifier. The module performs validation checks to prevent duplicate entries and alerts users in case of invalid or missing data. Real-time stock monitoring is supported, enabling proactive inventory management. It also allows batch updates and supports product search and filtering to improve manageability.

3.2 SALES MANAGEMENT MODULE

The Sales Management Module processes customer purchases. It allows users to select products, input quantities, and calculates the subtotal, applicable taxes, and total amount due. Once the sale is completed, the module automatically updates product stock levels and logs the transaction with a unique sales ID. It supports features such as applying discounts, handling multiple payment modes (cash, card, UPI), and generating itemized receipts. The module maintains a secure transaction history, which is useful for auditing, reporting, and tracking customer purchase trends.

3.3 DATABASE MANAGEMENT MODULE

This module is responsible for all data handling activities using MySQL. It stores information on products, sales, users, and system logs. The module ensures data integrity and consistency through relational constraints and transaction control. Backup and recovery features are included to prevent data loss in case of system failure. It supports data indexing for fast retrieval and enables report generation based on time range, product performance, or sales volume. User authentication data can also be securely stored in this module if login functionality is extended.

3.4 BACKEND API MODULE

The API module, developed with Flask, handles communication between the front end and the database. It provides RESTful endpoints to perform operations like adding, updating, retrieving, or deleting data. The API supports both GET and POST methods and uses JSON for data exchange. It includes robust input validation, structured error handling, and returns clear status

messages. Security features such as route protection, request throttling, and logging can be integrated. The modular design of the API allows scalability and future integration with web or mobile applications

3.5 USER INTERFACE MODULE

This module delivers a graphical user interface (GUI) using Tkinter, making system interaction intuitive and efficient. Users can manage products, view stock levels, process sales, and access transaction history through clearly labeled screens. The interface features user-friendly components like entry fields, combo boxes, buttons, listboxes, and pop-up dialogs for alerts and confirmations. It is designed to be responsive, supporting window resizing and keyboard navigation. Visual cues guide the user through tasks, reducing the learning curve. Future improvements may include support for dark mode, real-time notifications, and dashboard analytics widgets for better decision-making.

CHAPTER 4

CONCLUSION AND FUTURE ENHANCEMENT

4.1 CONCLUSION

The Retail Point of Sale System successfully integrates product, sales, and inventory management into a unified platform. By providing a centralized and automated solution, the system improves transaction speed, inventory accuracy, and overall operational efficiency for small businesses. The use of Python technologies ensures flexibility, and the modular structure makes future development and maintenance easier. With a user-friendly interface, business owners can easily manage daily tasks without needing extensive technical knowledge. Additionally, the seamless communication between the frontend, backend, and database ensures that every sale is accurately captured and reflected in stock updates and reports.

This system not only addresses current needs but also lays a foundation for digital transformation in small-scale retail environments. It eliminates the challenges of manual record keeping, reduces human errors, and enhances customer service by ensuring real-time stock availability and swift billing.

4.2 FUTURE ENHANCEMENTS

- **Implementing user authentication and role-based access control:** Ensures data security and restricts sensitive operations to authorized users only.
- **Adding barcode scanning:** Speeds up the sales process and reduces errors in product selection.
- **Generating detailed sales reports and analytics:** Helps in identifying trends, tracking business performance, and making informed decisions.
- **Cloud integration:** Allows real-time access to data from anywhere, supporting remote management and collaboration.
- **Email or SMS notifications for low stock alerts:** Assists in proactive inventory management by warning when stock reaches a critical level.
- **Mobile application integration:** Enables business owners and staff to access and operate the POS system from smartphones and tablets.

APPENDIX A

(PROJECT SOURCE CODE)

REQUIRED MODULES

```
import tkinter as tk
from tkinter import messagebox
import mysql.connector
from flask import Flask, request, jsonify
import threading
```

DATABASE MANAGER CLASS

```
class DatabaseManager:
    def __init__(self):
        self.conn = mysql.connector.connect(
            host="localhost",
            user="root",
            password="1234",
            database="pos_db"
        )
        self.cursor = self.conn.cursor()
        self.create_tables()
```

TABLE CREATION FOR PRODUCTS AND SALES

```
def create_tables(self):
    self.cursor.execute("""
        CREATE TABLE IF NOT EXISTS products (
            id INT AUTO_INCREMENT PRIMARY KEY,
            name VARCHAR(100),
            price DECIMAL(10,2),
            stock INT
        )
    """)
```

```

""")
self.cursor.execute("""
    CREATE TABLE IF NOT EXISTS sales (
        id INT AUTO_INCREMENT PRIMARY KEY,
        product_id INT,
        quantity INT,
        total_price DECIMAL(10,2),
        FOREIGN KEY (product_id) REFERENCES products(id)) """)
self.conn.commit()
self.cursor.execute("SELECT COUNT(*) FROM products")
count = self.cursor.fetchone()[0]

```

DEFAULT PRODUCTS

```

if count == 0:
    default_products = [
        ("Notebook", 30.00, 50),
        ("Pen", 10.00, 100),
        ("Pencil", 5.00, 120),
        ("Eraser", 3.00, 80),
        ("Stapler", 45.00, 25)
    ]
    for name, price, stock in default_products:
        self.add_product(name, price, stock)

```

ADD ITEM TO PRODUCT

```

def add_product(self, name, price, stock):
    self.cursor.execute("INSERT INTO products (name, price, stock) VALUES (%s, %s, %s)",
        (name, price, stock))
    self.conn.commit()

```

UPDATE ITEM TO PRODUCT

```

def update_stock(self, product_id, quantity):

```

```

self.cursor.execute("UPDATE products SET stock = stock - %s WHERE id = %s", (quantity,
    product_id))

self.conn.commit()

```

ADD ITEM TO SALES

```

def record_sale(self, product_id, quantity, total_price):

    self.cursor.execute("INSERT INTO sales (product_id, quantity, total_price) VALUES
    (%s, %s, %s)",

        (product_id, quantity, total_price))

    self.conn.commit()

```

DISPLAY PRODUCT AND SALES

```

def get_products(self):

    self.cursor.execute("SELECT * FROM products")

    return self.cursor.fetchall()

def get_sales(self):

    self.cursor.execute("SELECT * FROM sales")

    return self.cursor.fetchall()

```

FLASK API SETUP

```

app = Flask(__name__)

db = DatabaseManager()

@app.route("/products", methods=["GET"])

def get_products():

    products = db.get_products()

    return jsonify(products)

@app.route("/sales", methods=["GET"])

def get_sales():

    sales = db.get_sales()

    return jsonify(sales)

@app.route("/sale", methods=["POST"])

def make_sale():

```

```

data = request.json
product_id = data['product_id']
quantity = data['quantity']
total_price = data['total_price']
db.update_stock(product_id, quantity)
db.record_sale(product_id, quantity, total_price)
return jsonify({"message": "Sale recorded."})

```

POINT OF SALE INTERFACE (Tkinter GUI)

```
class POSInterface:
```

```

    def __init__(self, root, db_manager):
        self.root = root
        self.db = db_manager
        self.root.title("  Point of Sale (POS) System")
        self.root.geometry("500x700")
        self.root.configure(bg="#f0f8ff") # light blue background
        title = tk.Label(root, text="Retail POS System", font=("Arial", 20, "bold"), bg="#f0f8ff",
fg="#004080")
        title.pack(pady=10)

```

PRODUCT LIST FRAME

```

list_frame = tk.Frame(root, bg="#f0f8ff")
list_frame.pack(pady=10)
scrollbar = tk.Scrollbar(list_frame)
self.product_list = tk.Listbox(list_frame, width=60, height=10, yscrollcommand=scrollbar.set,
font=("Arial", 10))
scrollbar.config(command=self.product_list.yview)
scrollbar.pack(side=tk.RIGHT, fill=tk.Y)
self.product_list.pack(side=tk.LEFT)

```

INPUT FRAME

```
input_frame = tk.Frame(root, bg="#f0f8ff")
```

```

input_frame.pack(pady=10)

self._create_label_entry(input_frame, "Product Name:", "name_entry")
self._create_label_entry(input_frame, "Price:", "price_entry")
self._create_label_entry(input_frame, "Stock:", "stock_entry")
self._create_label_entry(input_frame, "Quantity to Sell:", "quantity_entry")

```

BUTTON FRAME

```

button_frame = tk.Frame(root, bg="#f0f8ff")
button_frame.pack(pady=20)

self._create_button(button_frame, "Sell", self.process_sale, "#28a745")
self._create_button(button_frame, "Add New Product", self.add_product, "#007bff")
self._create_button(button_frame, "Update Selected Product", self.update_product, "#ffc107",
fg="black")

self._create_button(button_frame, "Delete Selected Product", self.delete_product, "#dc3545")
self._create_button(button_frame, "Show Sales History", self.show_sales, "#17a2b8")

self.refresh_products()

```

UTILITY METHODS FOR UI

```

def _create_label_entry(self, frame, text, attr_name):
    label = tk.Label(frame, text=text, bg="#f0f8ff", font=("Arial", 10))
    label.pack()

    entry = tk.Entry(frame, font=("Arial", 10))
    entry.pack(pady=5)

    setattr(self, attr_name, entry)

def _create_button(self, frame, text, command, bg_color, fg="white"):
    button = tk.Button(frame, text=text, command=command, bg=bg_color, fg=fg,
font=("Arial", 10, "bold"), width=25, pady=5)
    button.pack(pady=5)

def refresh_products(self):
    self.product_list.delete(0, tk.END)

    products = self.db.get_products()

```

```

for product in products:

    self.product_list.insert(tk.END, f"{{product[0]}} - {{product[1]}} - ${{product[2]}} - Stock:
{{product[3]}}")

def add_product(self):

    name = self.name_entry.get()

    try:

        price = float(self.price_entry.get())

        stock = int(self.stock_entry.get())

    except ValueError:

        messagebox.showerror("Error", "Enter valid price and stock")

        return

    if not name:

        messagebox.showerror("Error", "Enter product name")

        return

    self.db.add_product(name, price, stock)

    messagebox.showinfo("Success", "Product added successfully")

    self.refresh_products()

def update_product(self):

    selected = self.product_list.curselection()

    if not selected:

        messagebox.showwarning("Warning", "No product selected to update")

        return

    product_data = self.product_list.get(selected[0]).split(" - ")

    product_id = int(product_data[0])

    try:

        new_price = float(self.price_entry.get())

        new_stock = int(self.stock_entry.get())

    except ValueError:

        messagebox.showerror("Error", "Enter valid price and stock to update")

```

```

        return

self.db.cursor.execute("UPDATE products SET price = %s, stock = %s WHERE id = %s",
                        (new_price, new_stock, product_id))

self.db.conn.commit()

messagebox.showinfo("Updated", "Product updated successfully!")

self.refresh_products()

def delete_product(self):
    selected = self.product_list.curselection()

    if not selected:
        messagebox.showwarning("Warning", "No product selected to delete")
        return

    product_data = self.product_list.get(selected[0]).split(" - ")
    product_id = int(product_data[0])

    confirm = messagebox.askyesno("Confirm Delete", "Are you sure you want to delete this
        product?")

    if confirm:
        self.db.cursor.execute("DELETE FROM products WHERE id = %s", (product_id,))
        self.db.conn.commit()
        messagebox.showinfo("Deleted", "Product deleted successfully!")
        self.refresh_products()

def process_sale(self):
    selected = self.product_list.curselection()

    if not selected:
        messagebox.showwarning("Warning", "No product selected")
        return

    product_data = self.product_list.get(selected[0]).split(" - ")
    product_id = int(product_data[0])
    price = float(product_data[2].replace("$", ""))

    try:

```

```

        quantity = int(self.quantity_entry.get())
except ValueError:
    messagebox.showerror("Error", "Enter a valid quantity")
    return
self.db.update_stock(product_id, quantity)
self.db.record_sale(product_id, quantity, price * quantity)
messagebox.showinfo("Success", "Sale processed")
self.refresh_products()
def show_sales(self):
    sales = self.db.get_sales()
    if not sales:
        messagebox.showinfo("Sales History", "No sales yet.")
        return
    sale_list = "\n".join([
        f'Sale ID: {sale[0]}, Product ID: {sale[1]}, Qty: {sale[2]}, Total: ${sale[3]}'
        for sale in sales
    ])
    messagebox.showinfo("Sales History", sale_list)

```

RUN FLASK API IN BACKGROUND THREAD

```

def run_api():
    app.run(port=5000)
api_thread = threading.Thread(target=run_api)
api_thread.daemon = True
api_thread.start()

```

MAIN ENTRY POINT

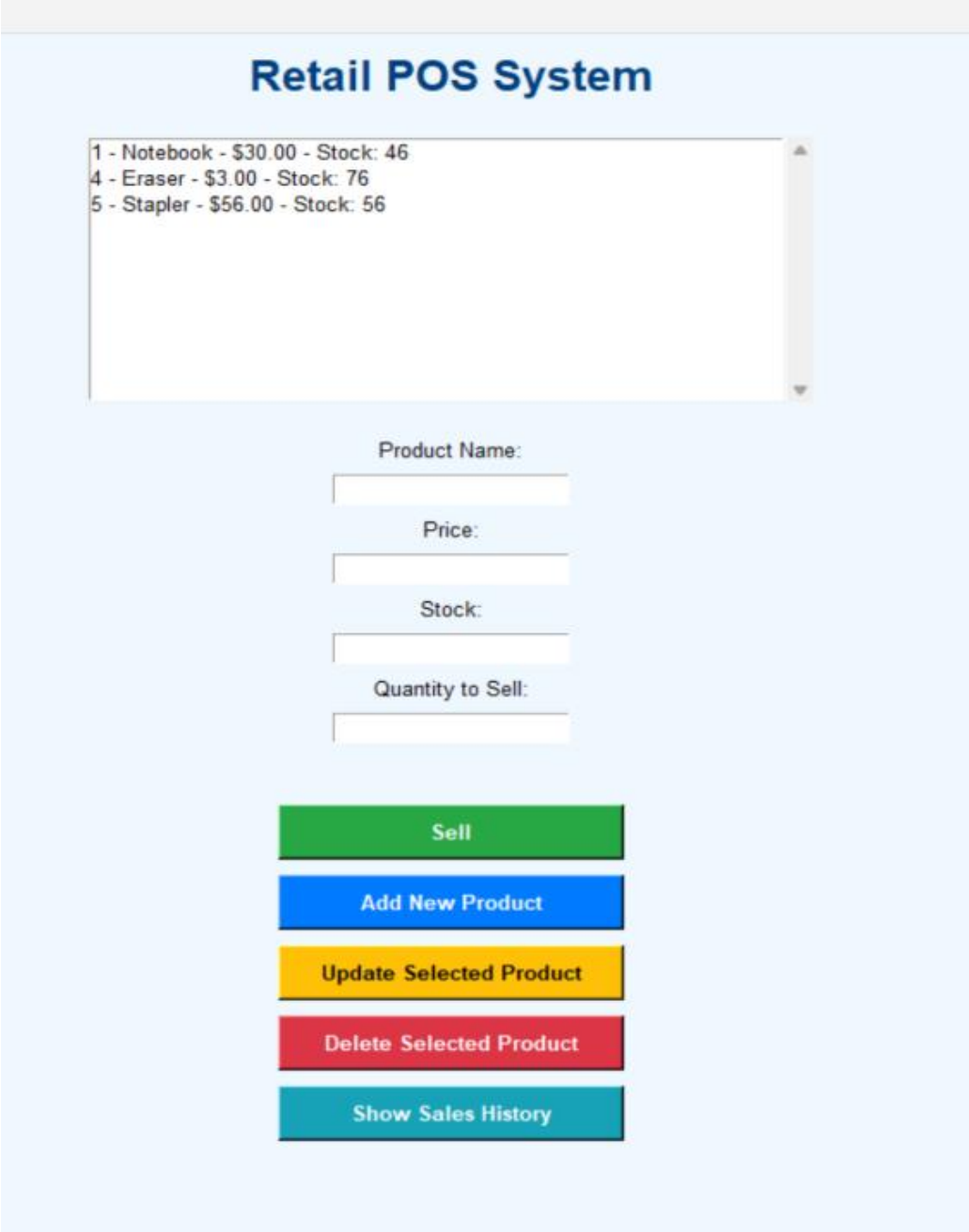
```

if __name__ == "__main__":
    root = tk.Tk()
    interface = POSInterface(root, db)
    root.mainloop()

```


APPENDIX B

(PASTE SCREENSHOTS WITH TITLE AND DESCRIPTION)



The screenshot displays a web-based application titled "Retail POS System". At the top, there is a scrollable list of products: "1 - Notebook - \$30.00 - Stock: 46", "4 - Eraser - \$3.00 - Stock: 76", and "5 - Stapler - \$56.00 - Stock: 56". Below this list, there are four input fields for product details: "Product Name:", "Price:", "Stock:", and "Quantity to Sell:". At the bottom of the interface, there are five colored buttons: a green "Sell" button, a blue "Add New Product" button, a yellow "Update Selected Product" button, a red "Delete Selected Product" button, and a teal "Show Sales History" button.

Retail POS System

1 - Notebook - \$30.00 - Stock: 46
4 - Eraser - \$3.00 - Stock: 76
5 - Stapler - \$56.00 - Stock: 56

Product Name:

Price:

Stock:

Quantity to Sell:

Sell

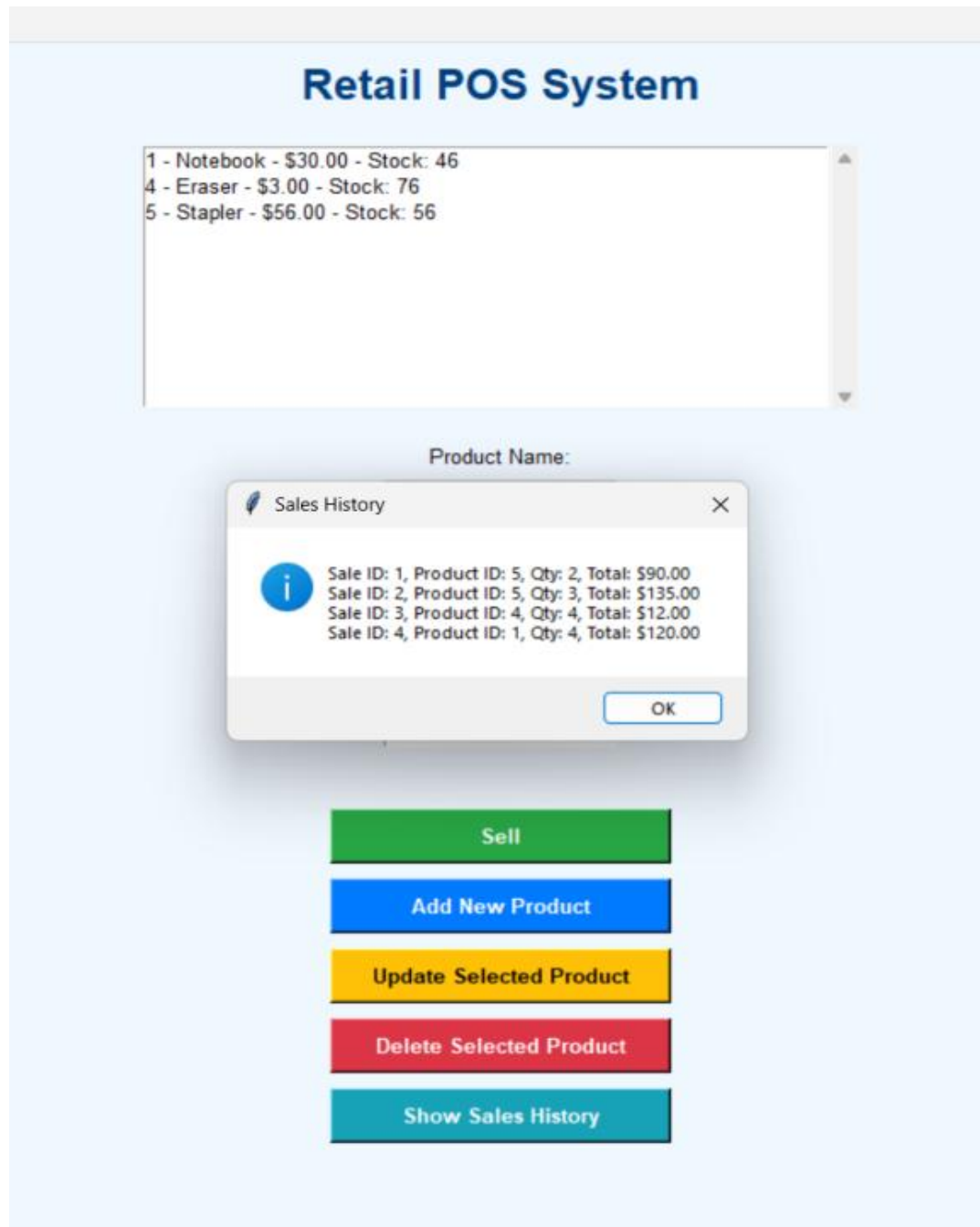
Add New Product

Update Selected Product

Delete Selected Product

Show Sales History

1.SHOW SELLS HISTORY



2.DELETE ITEM

Retail POS System

1 - Notebook - \$30.00 - Stock: 46
4 - Eraser - \$3.00 - Stock: 76
5 - Stapler - \$56.00 - Stock: 56
7 - lays - \$10.00 - Stock: 50
8 - lays - \$10.00 - Stock: 50

Product Name:

Price:

Deleted

i

Product deleted successfully!

OK

Sell

Add New Product

Update Selected Product

Delete Selected Product

Show Sales History

3.UPDATE ITEM

Retail POS System

1 - Notebook - \$30.00 - Stock: 46
4 - Eraser - \$3.00 - Stock: 76
5 - Stapler - \$56.00 - Stock: 56
8 - lays - \$10.00 - Stock: 42

Product Name:

Price:

Updated

i

Product updated successfully!

OK

Sell

Add New Product

Update Selected Product

Delete Selected Product

Show Sales History

4.ADD NEW ITEM

Retail POS System

1 - Notebook - \$30.00 - Stock: 46
4 - Eraser - \$3.00 - Stock: 76
5 - Stapler - \$67.00 - Stock: 67
8 - lays - \$10.00 - Stock: 42

Product Name:

pen

Price:

Success

i

Product added successfully

OK

Sell

Add New Product

Update Selected Product

Delete Selected Product

Show Sales History

5.SELL ITEM

Retail POS System

1 - Notebook - \$30.00 - Stock: 46
4 - Eraser - \$3.00 - Stock: 76
5 - Stapler - \$56.00 - Stock: 56
8 - lays - \$10.00 - Stock: 50

Product Name:

Price:

Success

i

Sale processed

OK

Sell

Add New Product

Update Selected Product

Delete Selected Product

Show Sales History

6.DATABASE OUTPUT

```
mysql> use pos_db;
```

```
Database changed
```

```
mysql> select * from products;
```

id	name	price	stock
1	Notebook	30.00	46
4	Eraser	3.00	76
5	Stapler	56.00	56
7	lays	10.00	50
8	lays	10.00	50

```
5 rows in set (0.00 sec)
```

```
mysql> select * from sales;
```

id	product_id	quantity	total_price
1	5	2	90.00
2	5	3	135.00
3	4	4	12.00
4	1	4	120.00

```
4 rows in set (0.03 sec)
```

REFERENCES:

BOOKS

"Learning Python" by Mark Lutz

- Covers core Python concepts thoroughly.
- Publisher: O'Reilly Media
- ISBN: 978-0-596-15806-4

"Python Programming – A Modern Approach" by Vamsi Kurama

- Suitable for beginners and college-level curriculum.
- Simple language and good for understanding OOP and GUI basics.

WEBSITES

Python Official Documentation

- <https://docs.python.org/3/>
- Reference for all built-in functions, syntax, and standard libraries.

GeeksforGeeks – Python Programming

- <https://www.geeksforgeeks.org/python-programming-language/>
- Beginner to advanced tutorials on Python, Tkinter, and MySQL connectivity.

YOUTUBE CHANNELS & VIDEOS

Programming with Mosh – Python for Beginners

- https://www.youtube.com/watch?v=_uQrJ0TkZlc
- A full 6-hour Python course (great for revision and beginners).

freeCodeCamp.org – Python Full Course

- <https://www.youtube.com/watch?v=rfscVS0vtbw>
- Covers everything from basics to object-oriented programming.