Project Report

Shop Management System

Name: Harshvardhan Singh Rathore

Course: Problem Solving and Programming

Registration No.: 25BCY10085

Introduction

The Shop Management System is a desktop-based Python application designed to streamline the daily operations of small retail stores.
The system allows the user to manage products, maintain records, and generate bills for customers.

It is developed using **Python Tkinter** for the graphical user interface and **MySQL** for persistent storage of product and sales data.
This project demonstrates the integration of GUI programming, database handling, and basic software architecture.

Problem Statement

Small shops traditionally maintain their product lists and sales records manually, which leads to:

- Difficulty tracking stock and prices
- High chances of human error
- Time-consuming billing process
- No proper storage of past customer sales
- Inefficient data retrieval

A simple computer-based system is needed to automate these tasks and assist shopkeepers in managing billing and product details accurately.

Functional Requirements

The system must support the following functions:

1. **Add Product**
    - Input: Date, Product Name, Price
    - Store product into MySQL database.
2. **Delete Product**
    - Remove a product by entering its name.
3. **View All Products**
    - Display all stored products with date, name, and price.
4. **Generate Bill**
    - Input: Date, Customer Name, Quantity of products
    - Calculate total price
    - Show bill summary on screen
    - Save sale records to database
5. **Exit / Quit**

- Close the application safely.

Non-Functional Requirements

- **Usability:**
  GUI must be simple, clean, and easy for non-technical users.
- **Performance:**
  Product retrieval, addition, and billing should be quick.
- **Reliability:**
  Database operations must execute successfully without data loss.
- **Maintainability:**
  Code should be modular with well-defined functions.
- **Portability:**
  Should run on any system with Python & MySQL installed.

System Architecture

The system follows a **two-tier architecture**:

GUI Layer (Frontend – Tkinter)

- Displays windows for product actions and billing
- Takes input from user
- Displays results

Database Layer (Backend – MySQL)

- Stores product records
- Stores sales data
- Responds to queries from Python

Design Diagrams:

Use Case Diagram

Actor: Shop Owner

Use Cases:

- Add Product
- Delete Product
- View Products
- Generate Bill
- Save Sale Record

Workflow Diagram

1. User selects an operation
2. System opens respective window
3. User enters information
4. System stores/fetches data
5. Result displayed

Sequence Diagram

User → GUI Input → Python Function → MySQL Query

MySQL Response → Python → GUI Output

Class/Component Diagram

Components:

- GUI Module
- Billing Module
- Database Module

ER Diagram

Table: products

- date
- prodName

- prodPrice

Table: sale

- custName
- date
- prodName
- qty
- price

Design Decisions & Rationale

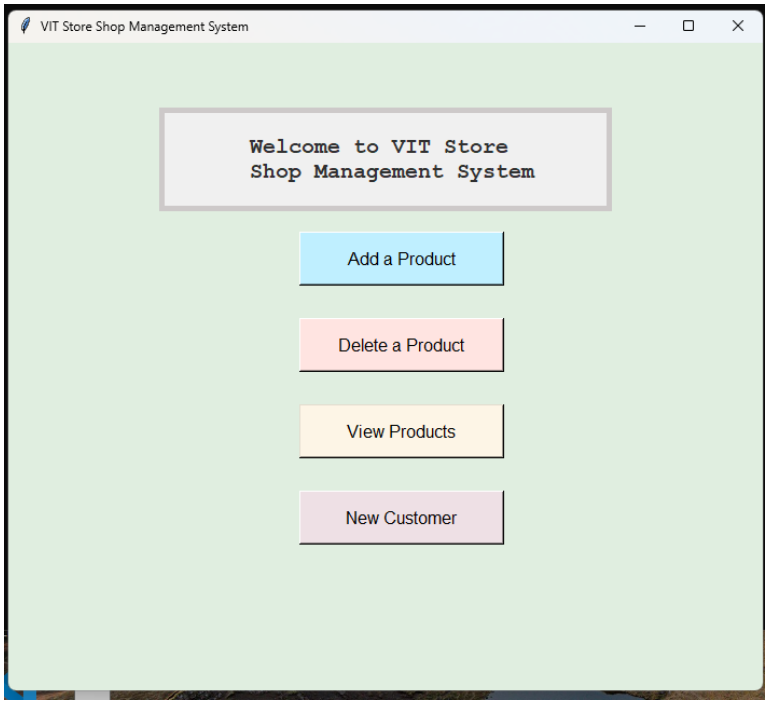
- **Tkinter chosen** for simplicity and availability in standard Python.
- **MySQL** selected for stability and well-structured data storage.
- **Modular functions** used to simplify debugging and maintenance.
- GUI colors selected to make sections visually distinguishable.

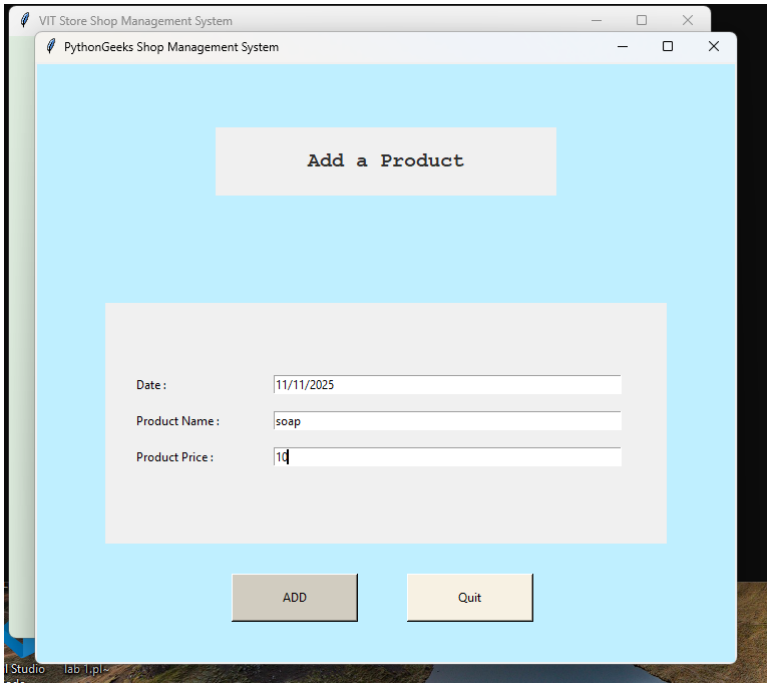
Implementation Details

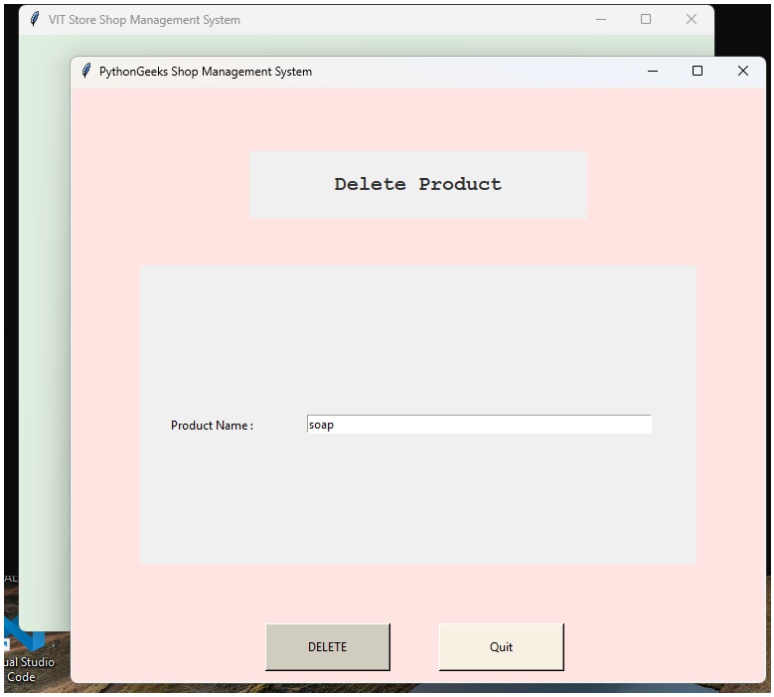
- MySQL database **Shop** is created automatically if not present.

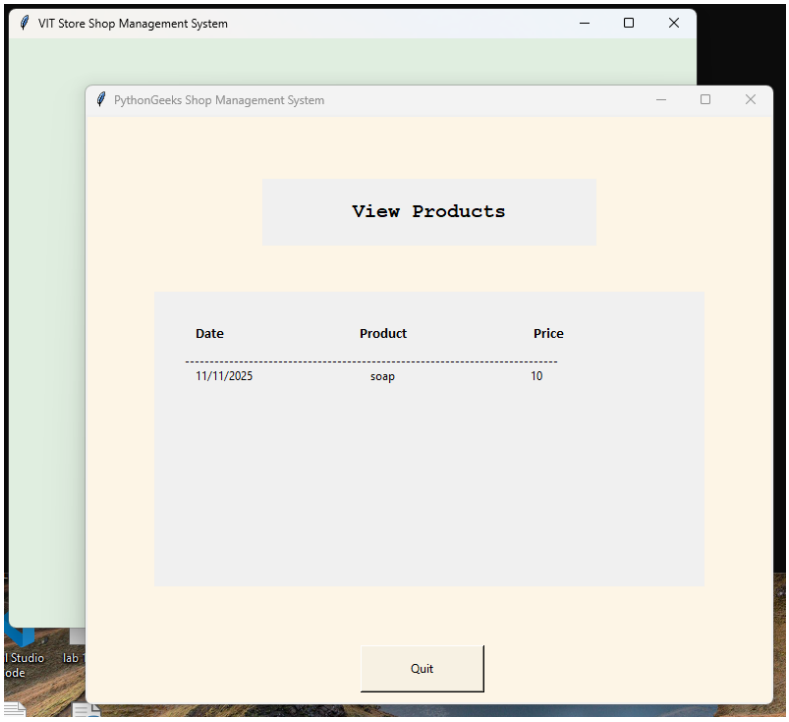
- Tables **products** and **sale** are created on program startup.
- Python functions handle CRUD operations using SQL queries.
- Bill generation uses logic for price × quantity + total aggregation.
- Each window of the project is built using Tkinter frames and labels.

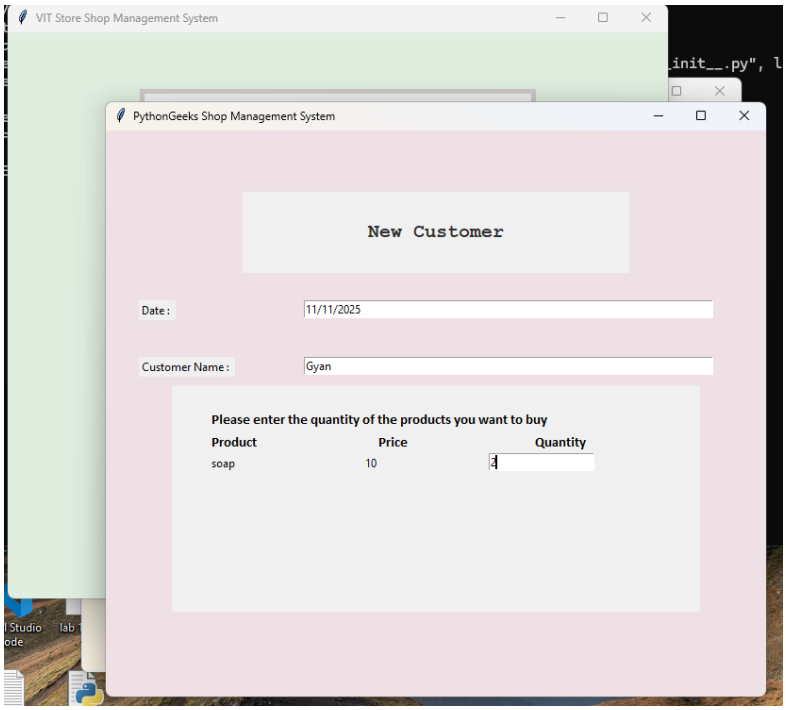
Screenshots / Results:




VIT Store Shop Management System
PythonGeeks Shop Management System
Add a Product
Date :
11/11/2025
Product Name :
soap
Product Price :
10
ADD
Quit


VIT Store Shop Management System
PythonGeeks Shop Management System
Delete Product
Product Name :
soap
DELETE
Quit





Testing Approach

Testing Methods Used

- **Unit Testing:**

    - Tested product addition, deletion, and DB queries
- **GUI Testing:**
    - Checked window opening and responsiveness
- **Integration Testing:**
    - Verified Tkinter + MySQL communication

Test Cases

1. Add valid product → Success
2. Add invalid data (blank fields) → Error message
3. Delete existing & non-existing products
4. Generate bill with valid quantity
5. Leave quantity empty → Bill ignores item

Challenges Faced

- Managing Tkinter window layouts
- Handling multiple windows without conflicts
- Maintaining connection to MySQL for each function
- Debugging SQL exceptions due to wrong formatting

Learnings & Key Takeaways

- Learned how to integrate GUI + Database
- Understood event handling in Tkinter
- Understood how SQL queries work in Python
- Improved debugging & modular coding practices

Future Enhancements

- Export bill as PDF
- Add authentication/login feature
- Add product editing functionality
- Improve UI with custom themes
- Add stock quantity tracking
- Add searchable product list

References

- Python Tkinter Documentation
- MySQL Official Documentation
- StackOverflow (debugging)

THE END