### SK SALES SALES CORPORATION

### A Project Report

Submitted in Partial fulfilment of the Requirements for the award of the Degree of

**BACHELOR OF SCIENCE (COMPUTER SCIENCE)**

# By ABDUL REHMAN KHAN

**Seat No.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Under the esteemed guidance of Prof. Javed Pathan** **Assistant Professor**

**DEPARTMENT OF COMPUTER SCIENCE**

**RIZVI COLLEGE OF ARTS, SCIENCE AND COMMERCE**

***(Affiliated to University of Mumbai)* MUMBAI-400050 MAHARASHTRA**

**2023-2024**

## RIZVI COLLEGE OF ARTS, SCIENCE AND COMMERCE

***(Affiliated to University of Mumbai)***

**MUMBAI-MAHARASHTRA-400050**

## DEPARTMENT OF COMPUTER SCIENCE



**CERTIFICATE**

This is to certify that the project entitled, **“SK SALES CORPORATION”,** is benefited work of **Khan Abdul Rehman** bearing **Seat** No.: **Roll No.** 39 submitted **in partial fulfilment of the requirements for the award of degree of BACHELOR OF SCIENCE in COMPUTER SCIENCE from University of Mumbai.**

### Project Guide HOD

**External Examiner**

**Date: …………. College Seal**

# DECLARATION

I hereby declare that the project entitled, **“SK SALES CORPORATION”** done at **Rizvi College of Arts, Science and Commerce,** has not been in any case duplicated to submit to any other university for the award of any degree. To the best of my knowledge other than me, no one has submitted to any other university.

The project is done in partial fulfilment of the requirements for the award of degree of **BACHELOR OF SCIENCE (COMPUTER SCIENCE)** to be submitted as final semester project as part of our curriculum.

**Khan Abdul Rehman**

**ACKNOWLEDGEMENT**

I owe special thanks to the Department of Computer Science of **Rizvi College of Arts Science and Commerce** for giving me a chance to prepare this project dissertation. I thank the Principal, **Professor Anjum Ara** for his leadership and management. I thank the Coordinator and Head of the Department **Professor Arif Patel** for providing us the required facilities and guidance throughout the course which culminated into this thesis. Last and not the least to the project guide this semester**- Professor Javed Pathan**. Deep gratitude to the staff and faculty of Rizvi College for their help and support. And also, my beloved **Parents** and **Classmates** for their infinite support and love.

# ABSTRACT

The Rexine Store Management System is an innovative software solution designed to streamline and enhance the operations of rexine and leather goods stores. In today's competitive retail landscape, efficient management of inventory, sales, and customer relationships is paramount. This software system aims to address these challenges by providing a user-friendly interface and a comprehensive set of features tailored specifically for rexine store owners and managers.

Key features of the Rexine Store Management System include inventory management, sales tracking, supplier management, customer relationship management, and reporting capabilities. With this system, store owners can easily monitor their stock levels, track sales trends, manage supplier relationships, and provide superior customer service. The intuitive user interface ensures that users, regardless of their technical proficiency, can navigate the system effortlessly, saving time and increasing productivity.

Additionally, the Rexine Store Management System offers real-time analytics and reporting tools, allowing store managers to make data-driven decisions. By analyzing sales patterns, identifying popular products, and predicting demand, store owners can optimize their inventory, reduce costs, and maximize profits. The system also includes features for customer engagement, enabling personalized communication and loyalty programs, which can enhance customer satisfaction and retention.

**TABLE OF CONTENT**

**CHAPTER 1. INTRODUCTION.......................................................……….01**

1.1 Introduction to the system................................................................................... 01

1.2 Problem Definition......... ................................................................................... .01

1.3 Aim...................................................................................................................... 02

1.4 Objective ...................................................................................... ..................... 02

1.5 Goal......................................................................................................... 02

**CHAPTER 2. REQUIREMENTS SPECIFICATION.......................04**

2.1 System Environment................................................................................04

2.2 Software Requirements........................................................................... 04

2.3 Hardware Requirements.......................................................................... 04

2.4 Methodology ........................................................................................... 04

**CHAPTER 3. SYSTEM ANALYSIS ....................................................07**

3.1 System Analysis.........................................................................................07

3.1.1 Analysis of Existing System....................................................................07

3.2 Gantt Chart.................................................................................................. 08

**CHAPTER 4. SURVEY OF TECHNOLOGY .......................................09**

4.1 PYTHON(tkinter).........................................................................................09

4.2 DB browser(sqlite)............................................................................... ....... 10

**CHAPTER 5. SYSTEM DESIGN.............................................................12**

5.1 Introduction................................................................................................. 12

5.2 System Architecture..................................................................................... 12

5.3 Data Flow Diagram...................................................................................... 13

5.4 Table Structure............................................................................................. 14

5.5 E-R Diagram. .............................................................................................. 15

**CHAPTER 6. SYSTEM IMPLEMENTATION.................................... 17**

6.1 Introduction ................................................................................................. 18

6.2 Flowchart..................................................................................................... 19

6.3Coding ......................................................................................................... 19

6.4 Testing Approach ...................................................................................... 91

**CHAPTER 7. RESULTS….................................................................... 94**

**CHAPTER 8. CONCLUSION AND FUTURE SCOPE. .................. 102**

8.1 Conclusion................................................................................................ 102

8.1.1 Advantages............................................................................................ 102

8.1.2 Limitations............................................................................................ 102

8.2 Future Enhancement................................................................................ 72

**CHAPTER 9. REFERENCES ............................................................ 103**

9.1 References............................................................................................... 103

**Chapter 1. INTRODUCTION**

* 1. **Introduction to the System**

Rexine is a synthetic material often used in the manufacturing of items such as bags, upholstery, clothing, and more. Effective store management is essential for ensuring smooth operations, customer satisfaction, and profitability. Here's an introduction to rexine store management:

**1. Business Overview:**

Begin by introducing the rexine store and its primary focus, which is the sale of rexine and related products. Mention the range of products you offer, such as rolls of rexine, finished goods, or customized products.

**2. Inventory Management:**

Explain how you handle inventory, including sourcing rexine materials, tracking stock levels, and ensuring a variety of colors, patterns, and types are available to meet customer demands.

**3.Customer Service:**

Emphasize your commitment to excellent customer service, including how you assist customers in choosing the right rexine for their needs, providing product information, and addressing any inquiries or concerns.

**4. Sales and Marketing:**

Describe your sales and marketing strategies, both online and offline. Discuss any promotions, discounts, or loyalty programs that you offer to attract and retain customers.

**5. Future Plans:**

Briefly touch upon your future plans for the rexine store, whether it's expanding to new locations, diversifying product offerings, or embracing new trends in rexine materials.

**6. Conclusion:**

Summarize the key points of your rexine store management approach and convey your dedication to providing top-quality rexine products and services to your customers.

**1.2 Problem Definition**

The management of a rexine store is currently facing several operational challenges that hinder efficiency, customer satisfaction, and overall business growth. To address these issues and streamline operations, there is a need to develop a comprehensive Rexine Store Management System (RSMS).

Key Challenges:

**Inventory Management:** The store struggles to maintain accurate inventory records, leading to overstocking or understocking of rexine materials. This results in financial losses due to tying up capital in excess inventory or losing sales opportunities.

**Order Processing:** Order processing is largely manual, which is time-consuming and error-prone. This affects order accuracy and customer satisfaction, as delays and errors can occur.

**Customer Relationship Management:** There is no centralized system for managing customer information and preferences. This hinders targeted marketing efforts and personalized customer service.

**Supplier Management:** Lack of effective supplier management leads to difficulties in tracking deliveries, negotiating favorable terms, and maintaining consistent quality and pricing.

**Sales Tracking**: The store lacks a system to track sales performance, analyze trends, and make informed decisions regarding pricing, promotions, and product offerings.

**Employee Productivity:** Employee productivity is affected by the absence of digital tools and automation. Manual data entry and paperwork consume valuable time that could be better utilized in serving customers and growing the business.

**Security and Data Privacy:** The store lacks adequate security measures to protect sensitive customer information and business data.

* 1. **AIM**

The primary aim of a Rexine Store Management System (RSMS) is to modernize and optimize the operations of a rexine store, ensuring efficient management of rexine materials, improving customer satisfaction, and enhancing overall business performance. The specific aims of an RSMS include:

**Efficient Inventory Management**: To automate and streamline inventory control processes, ensuring the right rexine materials are available in the right quantities to meet customer demand while minimizing excess stock.

**Seamless Order Processing:** To expedite order processing, reduce errors, and enhance order accuracy, ultimately improving customer satisfaction and order fulfillment efficiency.

**Enhanced Customer Relationship Management:** To build and maintain a loyal customer base by storing and utilizing customer data effectively, allowing for personalized marketing, promotions, and exceptional customer service.

**Security and Data Protection**: To implement robust security measures that safeguard customer information and business data, ensuring compliance with data protection regulations.

**Business Growth and Competitiveness:** To create a foundation for sustained growth by optimizing operations, increasing customer satisfaction, and staying competitive in the rexine industry.

* 1. **Objective**

**Efficient Inventory Management**: To automate and streamline inventory control processes, ensuring the right rexine materials are available in the right quantities to meet customer demand while minimizing excess stock.

**Seamless Order Processing:** To expedite order processing, reduce errors, and enhance order accuracy, ultimately improving customer satisfaction and order fulfillment efficiency.

**Enhanced Customer Relationship Management:** To build and maintain a loyal customer base by storing and utilizing customer data effectively, allowing for personalized marketing, promotions, and exceptional customer service.

**Security and Data Protection**: To implement robust security measures that safeguard customer information and business data, ensuring compliance with data protection regulations.

**Business Growth and Competitiveness:** To create a foundation for sustained growth by optimizing operations, increasing customer satisfaction, and staying competitive in the rexine industry.

* 1. **GOAL**

The goals for a Rexine Store Management System (RSMS) are specific, measurable, and achievable targets that guide its development and implementation. These goals align with the objectives and aims of the RSMS and provide a clear direction for the system's functionality and impact. Here are some key goals for an RSMS:

**Optimize Inventory Control:**

Goal: Reduce excess inventory levels by 20% within the first year of implementing the RSMS.

Measure: Regularly monitor inventory turnover rates and stockout incidents.

**Enhance Order Processing Efficiency:**

Goal: Decrease order fulfillment time by 30% to improve customer satisfaction.

Measure: Track the time taken from order placement to delivery.

**Improve Customer Engagement:**

Goal: Increase customer retention rates by 15% through personalized marketing and service.

Measure: Analyze customer retention and repeat purchase rates.

**Chapter 2. Requirement Specification**

* 1. **System Environment**

A Requirement Specification for a Rexine Store Management System (RSMS) outlines the detailed functional and non-functional requirements that the system must meet to effectively manage the operations of a rexine store. These requirements serve as a blueprint for system development and ensure that the RSMS aligns with the store's specific needs and goals

The system environment for a Rexine Store Management System (RSMS) includes the hardware, software, network, and other components necessary to support the system's operation.

**Server:** A dedicated server or cloud-based hosting infrastructure is required to host the RSMS software and database. The server should have sufficient processing power, memory, and storage capacity to handle the system's demands.

**Barcode Scanners:** Barcode scanners may be used for efficient inventory management and order processing.

**Printers:** Printers are needed for generating invoices, packing slips, shipping labels, and other documents related to order fulfillment.

**2.2 Software Requirements**

Technology: Python(tkinter)

Visual studio : It use for Coding.

Operating System: Microsoft Windows

DB Browser(SQLITE): Its uses for Storing and managing database

**2.3 Hardware Requirements**

• Processor (CPU): Intel Core i4 (sixth generation or newer)

• Operating System: Microsoft Windows 10 Professional x64

• Memory: 8 GB RAM

• Storage: 500 GB internal storage drive

• Other: Internal or external Webcam, lock, carrying case, external hard drive for backups

**2.4 Methodology**

Developing a Rexine Store Management System (RSMS) involves a structured approach or methodology to ensure that the system is effectively designed, developed, tested, and deployed. Here's a commonly used methodology for developing an RSMS:

**1. Requirements Gathering:**

Start by gathering detailed requirements for the RSMS. This involves working closely with store management and end-users to understand their needs and objectives. Document functional and non-functional requirements, as well as any specific customization or integration needs.

**2. Feasibility Study:**

Conduct a feasibility study to assess the technical, operational, and financial feasibility of the RSMS project. Determine if it aligns with the store's goals and budget.

**3. System Design:**

Create a comprehensive system design based on the gathered requirements. This includes defining the system architecture, database schema, user interfaces, and integration points with other systems. Consider scalability, security, and performance at this stage.

**4. Technology Selection:**

Choose the appropriate technology stack for the RSMS, including the programming language, database management system (DBMS), and any third-party libraries or frameworks. Ensure that the chosen technologies align with the project's goals and constraints.

**5. Development:**

Start the development phase by creating the core components of the RSMS, including inventory management, order processing, customer management, and analytics features. Follow best practices for coding standards and documentation.

**6. Testing:**

Conduct thorough testing of the RSMS to identify and fix bugs, ensure system stability, and validate that it meets the specified requirements. Types of testing may include unit testing, integration testing, system testing, and user acceptance testing (UAT).

**7. Deployment:**

Deploy the RSMS in the production environment. Ensure that all necessary hardware, software, and network configurations are in place. Conduct data migration and perform final system checks before making it accessible to users.

**8. Final Review:**

Conduct a final review of the RSMS project to assess its success in meeting the defined objectives.Document lessons learned and best practices for future projects.

**Chapter 3. System Analysis**

**1.1 System Analysis**

System analysis for a Rexine Store Management System (RSMS) involves a comprehensive study of the existing system, identifying requirements, and proposing solutions to streamline operations, enhance efficiency, and meet the specific needs of the rexine store. Here's a breakdown of the key aspects of system analysis for an RSMS:

**1. Understanding Current System:**

Conduct interviews and meetings with store owners, managers, and employees to understand existing processes, workflows, and challenges.Identify manual and automated processes currently in use.Document the strengths and weaknesses of the current system.

**2. Requirement Gathering:**

Gather detailed requirements from stakeholders, including store managers, employees, and customers.Identify functional requirements (features the system must have) and non-functional requirements (performance, security, etc.).Prioritize requirements based on their importance to the business operations.

**3. Use Case Development:**

Create detailed use cases to describe how different users interact with the system. This includes scenarios for inventory management, order processing, customer interactions, etc.Use use cases to understand the system's behavior from an end-user perspective.

**4. Data Analysis:**

Analyze the types of data the system will handle (inventory data, customer data, supplier data, etc.). Determine data storage requirements and data relationships.Plan for data security, including encryption, access controls, and data backup procedures.

**5. Feasibility Study:**

Conduct a feasibility study to assess the technical, economic, and operational feasibility of the proposed RSMS. Evaluate if the benefits of implementing the RSMS outweigh the costs and potential risks.

**6. System Design Specifications:**

Create detailed system design specifications outlining the architecture, database schema, user interfaces, and integration points.Develop wireframes or prototypes to visualize the system's user interface and gather feedback from stakeholders.

**7. Risk Analysis:**

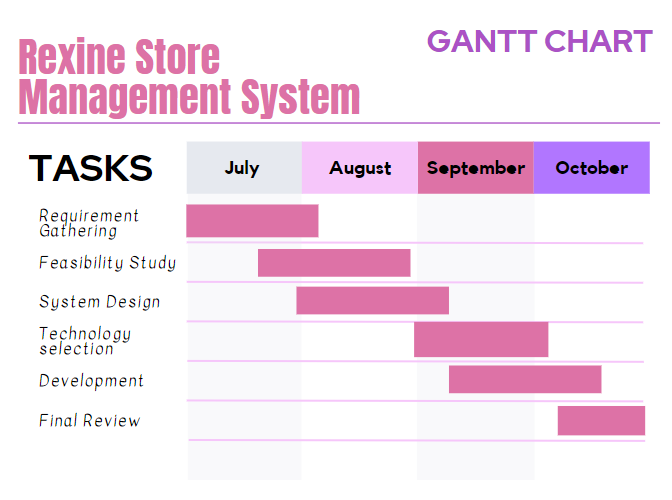
Identify potential risks related to technology, implementation, and user acceptance. Develop risk mitigation strategies to address identified risks.

**3.2.1 Analysis of Existing System**

Before we analyse the design of the proposed system, we need to carefully highlight the problems of the existing system so as to avoid recurrence. This analysis serves as a pointer on how to embark on building the proposed system that will help the women provide. The problems of the current system should be outlined. Below are some of the problems associated with the existing system.

In the existing system there is no monitoring system for girls, it should create many problems for them and the no safety mechanism to protect the girls from the misbehaviour activities. In addition, in the existing system there is no alert mechanism for the girl’s safety, it should be done by manually only

**3.3. GANTT CHART**

****

**Chapter 4. Survey of Technology**

* 1. **PYTHON(tkinter)**

****

Python's Tkinter library serves as a fundamental tool for creating graphical user interfaces (GUIs) in Python applications. Offering a straightforward and intuitive approach, Tkinter allows developers, especially beginners, to design interactive desktop applications with ease. It provides a wide range of widgets, including buttons, labels, and entry fields, enabling the creation of functional and visually appealing interfaces. Tkinter operates on an event-driven programming model, where user actions such as button clicks or keyboard input trigger specific functions, providing interactivity to applications. One of its significant advantages is its platform independence; applications developed using Tkinter can run seamlessly on various platforms, including Windows, macOS, and Linux. Integrated into Python's standard library, Tkinter ensures compatibility with other Python libraries and tools, facilitating seamless integration into diverse Python projects. While it may lack some advanced features found in specialized GUI libraries, Tkinter's simplicity and ease of learning make it an excellent choice for rapid prototyping and developing basic to moderately complex GUI applications. Its active community and extensive documentation provide valuable resources for developers, making it a popular choice for those venturing into GUI programming with Python.

DB Browser for SQLite, often simply referred to as DB Browser, is a powerful and open-source tool designed for managing SQLite databases. SQLite is a lightweight, file-based database engine widely used in various applications due to its simplicity and efficiency. DB Browser provides users with a user-friendly graphical interface to interact with SQLite databases, making it accessible to both beginners and experienced developers.

One of the key features of DB Browser is its intuitive interface, which allows users to create, edit, and manage SQLite databases visually. Users can design database schemas, define tables, modify data, and execute SQL queries, all through a convenient graphical user interface. This is particularly useful for those who prefer working with databases visually rather than writing complex SQL commands.

DB Browser also supports the import and export of data in various formats, facilitating easy data migration between different systems. It provides tools for analyzing database structures and querying data, enabling users to understand the relationships within their data effectively. Additionally, it supports the execution of complex SQL scripts, making it a versatile tool for developers working with SQLite databases.

Furthermore, DB Browser is a cross-platform application, available for Windows, macOS, and Linux operating systems. This platform independence ensures that users can seamlessly work with SQLite databases regardless of their operating system preference.

In summary, DB Browser for SQLite is a valuable tool for developers, data analysts, and database administrators, offering a convenient and user-friendly way to interact with SQLite databases. Its intuitive interface, data management features, and cross-platform compatibility make it a popular choice within the SQLite community.

* 1. **DB Browser(sqlite)**

****

DB Browser for SQLite, often simply referred to as DB Browser, is a powerful and open-source tool designed for managing SQLite databases. SQLite is a lightweight, file-based database engine widely used in various applications due to its simplicity and efficiency. DB Browser provides users with a user-friendly graphical interface to interact with SQLite databases, making it accessible to both beginners and experienced developers.

One of the key features of DB Browser is its intuitive interface, which allows users to create, edit, and manage SQLite databases visually. Users can design database schemas, define tables, modify data, and execute SQL queries, all through a convenient graphical user interface. This is particularly useful for those who prefer working with databases visually rather than writing complex SQL commands.

DB Browser also supports the import and export of data in various formats, facilitating easy data migration between different systems. It provides tools for analyzing database structures and querying data, enabling users to understand the relationships within their data effectively. Additionally, it supports the execution of complex SQL scripts, making it a versatile tool for developers working with SQLite databases.

Furthermore, DB Browser is a cross-platform application, available for Windows, macOS, and Linux operating systems. This platform independence ensures that users can seamlessly work with SQLite databases regardless of their operating system preference.

In summary, DB Browser for SQLite is a valuable tool for developers, data analysts, and database administrators, offering a convenient and user-friendly way to interact with SQLite databases. Its intuitive interface, data management features, and cross-platform compatibility make it a popular choice within the SQLite community.

**CHAPTER 5. SYSTEM DESIGN**

**5.1 Introduction**

The system design involves structuring the software architecture and components to efficiently manage various aspects of the store's operations. The system needs to address inventory management, sales tracking, customer interactions, and administrative tasks in a seamless manner.

Firstly, the database design is critical. It involves creating a database schema to store product information, including details such as product names, categories, prices, and available quantities. This database also includes customer information, sales transactions, and supplier details. Normalization techniques are employed to ensure efficient data storage and retrieval.

The user interface design focuses on creating an intuitive and user-friendly interface for both employees and customers. For employees, the interface includes functionalities for managing inventory, processing sales, and generating reports. For customers, it includes features for browsing products, placing orders, and tracking their purchase history. Visual elements, such as product images and intuitive navigation menus, enhance the user experience.

In terms of functionality, the system should incorporate features such as real-time inventory tracking, automatic restocking notifications, and sales analytics. Barcode scanning technology can be integrated to streamline the checkout process, reducing the chances of manual errors. A secure authentication and authorization system ensures that only authorized personnel can access sensitive information and perform specific tasks within the system.

Considering security, the system should implement encryption protocols to safeguard sensitive data, such as customer payment information. Regular security audits and penetration testing are essential to identify and address vulnerabilities.

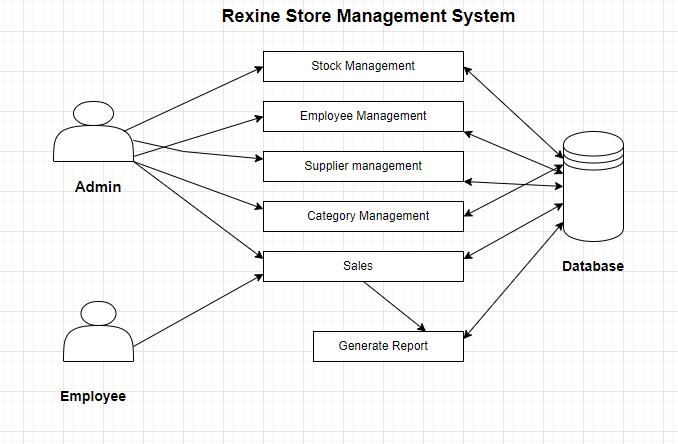
Additionally, the system design includes scalability and performance considerations. The architecture must handle a growing database of products and customer information efficiently. Load balancing techniques and caching mechanisms can be employed to ensure the system performs optimally even during peak usage times.

Error handling and recovery mechanisms are integrated to manage unexpected system failures or data inconsistencies. Regular backups of the database are scheduled to prevent data loss and facilitate recovery in case of system failures.

Lastly, comprehensive documentation detailing the system architecture, database schema, user manuals, and troubleshooting guides is essential. It ensures that developers, administrators, and users have access to necessary information for system setup, maintenance, and usage.

In conclusion, the Rexine store management system's design encompasses database organization, user interface intuitiveness, robust functionality, security implementations, scalability provisions, error management, and thorough documentation. A well-designed system ensures efficient store operations, enhances user experience, and provides a solid foundation for future enhancements and expansions.

**5.2 System Architecture**



The system architecture is a crucial blueprint that defines the arrangement and interaction of its components, ensuring efficient operation and seamless integration of various functionalities. At its core, the system typically consists of several interconnected modules: User Interface (UI), Application Logic, Database Management System (DBMS), and External Interfaces.

The User Interface (UI) is the front-facing component that interacts directly with the users, both employees and customers. It provides an intuitive and user-friendly interface, allowing users to browse products, place orders, and manage inventory. In the context of a Rexine store, the UI should feature product categorization, search functionalities, and a streamlined checkout process. The UI can be web-based or a standalone application, depending on the specific requirements and accessibility preferences.

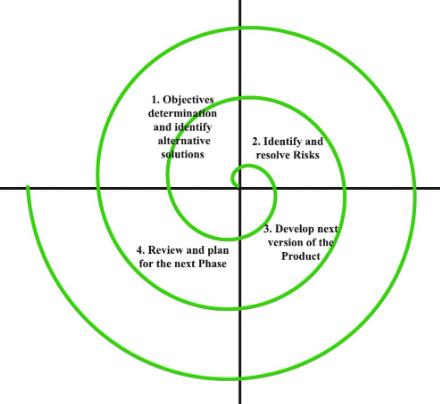
The Application Logic, also known as the business logic, serves as the brain of the system. It processes user inputs from the UI, performs necessary computations, and communicates with the database to retrieve or store data. In the context of a Rexine store management system, the application logic handles tasks such as inventory management, sales processing, order fulfillment, and generating reports. It ensures that business rules are followed, and transactions are accurate and secure.

The Database Management System (DBMS) is responsible for storing, retrieving, and managing data. In the case of a Rexine store management system, the database stores information about products, customers, suppliers, inventory levels, and sales transactions. It provides a structured and efficient way to store and retrieve data, ensuring data integrity and consistency. The DBMS can use relational database management systems (such as MySQL, PostgreSQL, or SQLite) to organize the data into tables and establish relationships between different entities.

External Interfaces are essential components that enable the system to interact with external entities. This can include integration with payment gateways for processing online transactions, connecting with barcode scanners for quick inventory management, or linking with supplier databases for automatic restocking. These interfaces enhance the system's functionality and allow for a seamless exchange of information between the store management system and external parties.

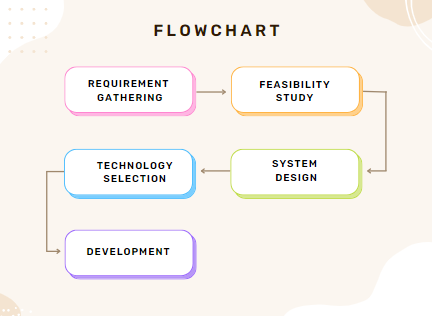
The system architecture must also consider factors such as security, scalability, and fault tolerance. Security measures like encryption and authentication protocols safeguard sensitive data, while scalability provisions ensure that the system can handle a growing volume of products, customers, and transactions. Fault tolerance mechanisms, such as backup and recovery procedures, are put in place to minimize data loss in case of system failure.

**5.3 Spiral Model**

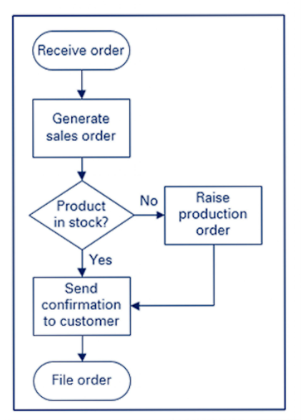
****

**5.3 Flow diagram**

**Data Flow Diagram**

****

**STOCK CHECKING FLOW DIAGRAM.**

****

**5.4 Table Structure**

Designing the table structure for a Rexine store management system involves creating a database schema that can efficiently store and retrieve information about products, customers, sales transactions, suppliers, and other relevant data. Here's an example of a basic table structure for such a system:

**Product Table:**

product\_id (Primary Key): Unique identifier for each product.

product\_name: Name of the product.

category: Category of the product (e.g., rexine rolls, adhesives, tools).

price: Price of the product.

quantity\_in\_stock: Quantity of the product available in the inventory.

supplier\_id: Foreign Key linking to the Supplier Table, indicating the supplier of the product.

**Customer Table:**

customer\_id (Primary Key): Unique identifier for each customer.

first\_name: First name of the customer.

last\_name: Last name of the customer.

email: Email address of the customer.

phone: Phone number of the customer.

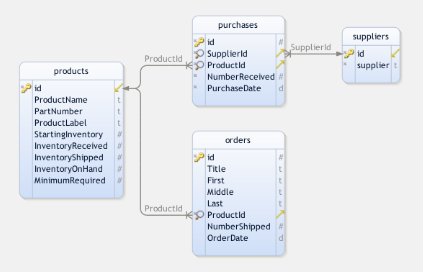
**Supplier Table:**

supplier\_id (Primary Key): Unique identifier for each supplier.

supplier\_name: Name of the supplier.

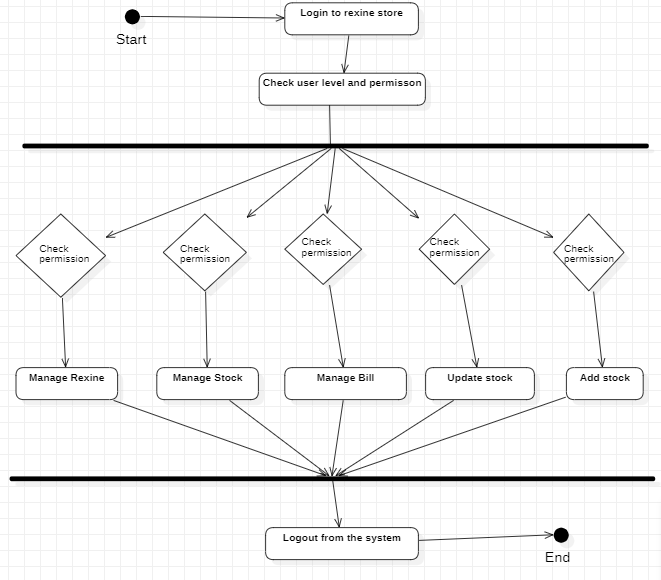
contact\_person: Name of the contact person at the supplier's company.

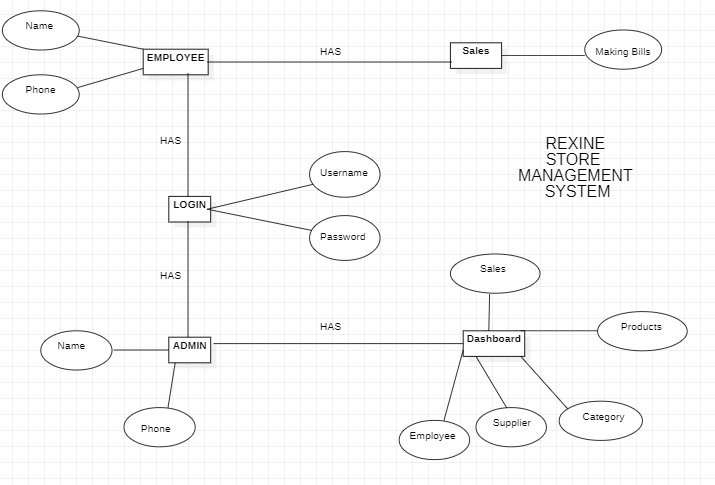
phone: Phone number of the supplier.

****

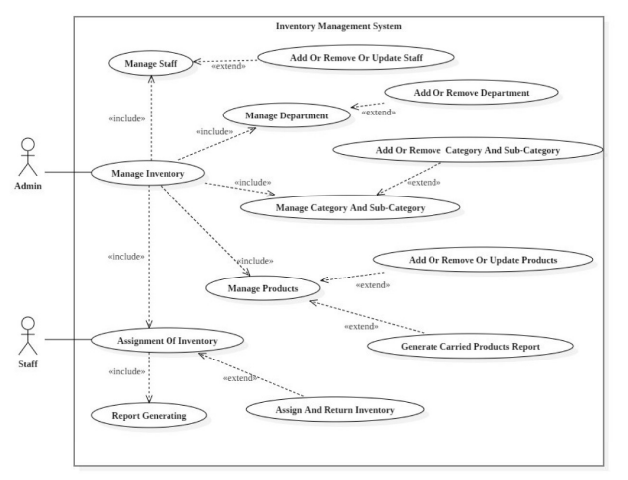
**5.5 E-R Diagram**

An ER diagram shows the relationship among entity sets. An entity set is a group of similar entities and these entities can have attributes. In terms of DBMS, an entity is a table or attribute of a table in database, so by showing relationship among tables and their attributes, ER diagram shows the complete logical structure of a database. Entity Relational (ER) Model is a high-level conceptual data model diagram. ER modelling helps you to analyse data requirements systematically to produce a well-designed database. The Entity-Relation model represents real-world entities and the relationship between them. It is considered a best practice to complete ER modelling before implementing your database.



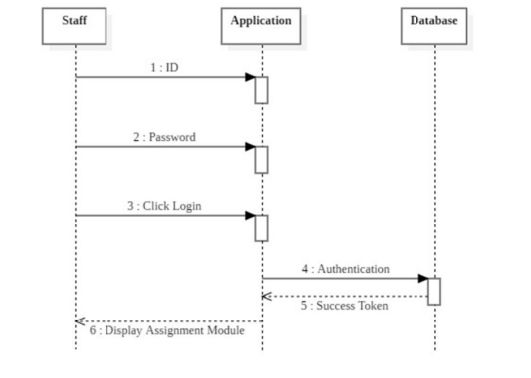
****

**5.6 Use Case Diagram**

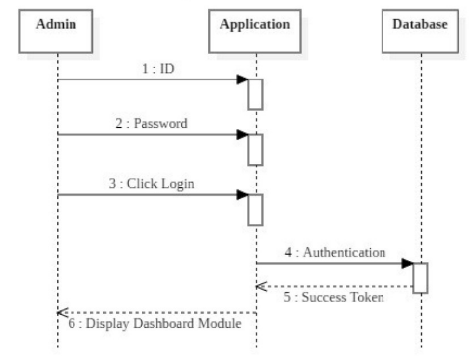
****

**5.7 Sequence Diagram**

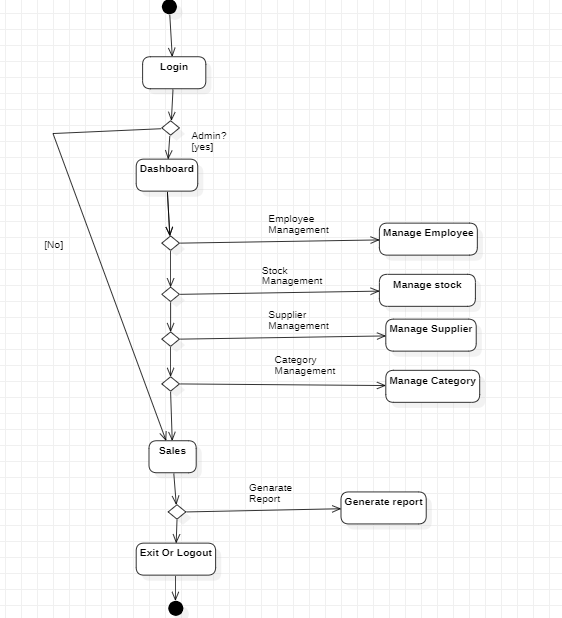
**Sequence digram for login(**staff**)**

****

**Sequence digram for login(**Admin**)**

****

**5.5 Activity Diagram**

****

**Chapter 6. System Implementation**

**6.1 Introduction**

System implementation for a Rexine Store Management System (RSMS) involves the actual coding and development of the software based on the system design and requirements. During this phase, the system design is translated into a functional application through programming, testing, and integration. Project implementation is the process of putting a project plan into action to produce the deliverables, otherwise known as the products or services, for clients or stakeholders. It takes place after the planning phase, during which a team determines the key objectives for the project, as well as the timeline and budget.

**6.2 Coding**

1. **Login and SignUp file.**

from tkinter import \*

from PIL import Image,ImageTk #pip install pillow

from tkinter import ttk,messagebox

import sqlite3

import os

# import email\_pass

import smtplib #pip install smtplib ` `

import time

class Login\_System:

def \_\_init\_\_(self,root):

self.root=root

self.root.title("Login System | Project")

self.root.geometry("450x650+500+60")

self.root.config(bg="#fafafa")

self.otp=''

# #====images=====

# self.phone\_image=ImageTk.PhotoImage(file="images/phone.png")

# self.lbl\_image=Label(self.root,image=self.phone\_image,bd=0).place(x=180,y=70,height=620)

#====Login Frame======

self.var\_emp\_id=StringVar()

self.var\_password=StringVar()

Login\_Frame=Frame(self.root,bd=2,relief=RIDGE,bg="white")

Login\_Frame.place(x=50,y=30,width=350,height=500)

Frametitle=Label(Login\_Frame,text="Login System",font=("Elephant",30,"bold"),bg="aliceblue",fg="navy").place(x=0,y=10,relwidth=1)

lbl\_emp\_id=Label(Login\_Frame,text="Employee ID",font=("Andalus",15),bg="#ECECEC",fg="#767171").place(x=50,y=100)

txt\_emp\_id=Entry(Login\_Frame,textvariable=self.var\_emp\_id,font=("times new roman",15),bg="lightyellow",fg="blue").place(x=50,y=140,width=180,height=30)

lbl\_password=Label(Login\_Frame,text="Password",font=("Andalus",15),bg="#ECECEC",fg="#767171").place(x=50,y=200)

txt\_password=Entry(Login\_Frame,textvariable=self.var\_password,show="\*",font=("times new roman",15),bg="lightyellow",fg="blue").place(x=50,y=240,width=180,height=30)

#print(self.var\_emp\_id,self.var\_password)

btn\_login=Button(Login\_Frame,text=" Log In ",command=self.login,font=("Arial Rounded MT Bold",20,"bold"),bg="#00B0F0",activebackground="#00B0F0",fg="yellow",cursor="hand2").place(x=50,y=300,width=250,height=40)

hr=Label(Login\_Frame,bg="lightgreen").place(x=50,y=370,width=250,height=5)

\_or\_=Label(Login\_Frame,text="OR",font=("Helvetica",10,"bold"),bg="white",fg="lightgreen").place(x=160,y=357,height=30)

lbl\_forgot\_pass=Label(Login\_Frame,text="Forgot Password\n↓Click Below↓",font=("Impact",15),bg="#FFFF00",fg="skyblue").place(x=50,y=390,width=250,height=50)

btn\_forget\_pass=Button(Login\_Frame,text=" Forget Password ? ",command=self.for\_pass\_win,font=("Comic Sams MS",15,"bold"),bg="white",fg="aquamarine",bd=0,cursor="hand2",activebackground="white",activeforeground="aquamarine").place(x=65,y=450,width=220,height=20)

#===Register Frame=====

Sign\_Up\_Frame=Frame(self.root,bd=2,relief=RIDGE,bg="white")

Sign\_Up\_Frame.place(x=50,y=550,width=350,height=70)

lbl\_register\_info=Label(Sign\_Up\_Frame,text=" ↓Don't have an Account??↓ ",font=("Candara",15,"bold"),bg="azure",fg="teal").place(x=55,y=5)

btn\_sign\_up=Button(Sign\_Up\_Frame,text=" ..Sign Up.. ",command=self.sign\_up,font=("",15,"bold"),bg="white",fg="lime",cursor="hand2",bd=0,activebackground="white",activeforeground="lime").place(x=90,y=40,width=150,height=20)

# #====Animation Images=======

#

# self.im1=ImageTk.PhotoImage(file="images/im1.png")

# self.im2=ImageTk.PhotoImage(file="images/im2.png")

# self.im3=ImageTk.PhotoImage(file="images/im3.png")

#

# self.lbl\_change\_image=Label(self.root,bg="white")

# self.lbl\_change\_image.place(x=347,y=177,width=240,height=428)

#

# self.animate\_image()

# #self.send\_email('sppr.8448@gmail.com')

#

# def animate\_image(self):

# self.im0=self.im1

# self.im1=self.im2

# self.im2=self.im3

# self.im3=self.im0

# self.lbl\_change\_image.config(image=self.im0)

# self.lbl\_change\_image.after(2000,self.animate\_image)

def login(self):

con=sqlite3.connect(database=r'ims.db')

cur=con.cursor()

try:

if self.var\_emp\_id.get()=="" or self.var\_password.get()=="":

messagebox.showerror("Error","All Fields are Required",parent=self.root)

else:

cur.execute("select utype from employee where eid=? AND pass=?",(self.var\_emp\_id.get(),self.var\_password.get()))

employee=cur.fetchone()

if employee==None:

messagebox.showwarning("Warning","Invalid Employee ID or Password",parent=self.root)

else:

#print(employee)

if employee[0]=="Admin":

self.root.destroy()

os.system("python dashboard.py")

else:

self.root.destroy()

os.system("python billing.py")

except Exception as ex:

messagebox.showerror("Error",f"Error due to : {str(ex)}",parent=self.root)

'''if self.var\_emp\_id.get()=="" or self.var\_password.get()=="":

messagebox.showerror("Error","All Fields are Required!!!")

elif self.var\_emp\_id.get()!="Ayush" or self.var\_password.get()!="19875":

messagebox.showwarning("Warning","Invalid Username or Password\nTry Again with Correct Credentials")

else:

messagebox.showinfo("Information",f"Welcome : {self.var\_emp\_id.get()}\n Your PAssword is {self.var\_password.get()}")'''

# def for\_pass\_win(self):

con=sqlite3.connect(database=r'ims.db')

cur=con.cursor()

try:

if self.var\_emp\_id.get()=="":

messagebox.showerror("Error","Employee ID is Mandatory!!!",parent=self.root)

else:

cur.execute("select email from employee where eid=?",(self.var\_emp\_id.get(),))

email=cur.fetchone()

if email[0]==None:

messagebox.showwarning("Warning","Invalid Employee ID\nTry Again",parent=self.root)

else:

#print(email[0])

#====Forget Password Window=========

self.var\_otp=StringVar()

self.var\_new\_pass=StringVar()

self.var\_con\_pass=StringVar()

#call send\_email\_function()

chk=self.send\_email(email[0])

if chk=='f':

messagebox.showerror("Error","Connection Error Try Again",parent=self.root)

else:

self.forget\_password\_window=Toplevel(self.root)

self.forget\_password\_window.title("RESET PASSOWRD")

self.forget\_password\_window.geometry("400x400+500+100")

self.forget\_password\_window.focus\_force()

#======labels====

forpasstitle=Label(self.forget\_password\_window,text="Reset Password",font=("goudy old style",15,"bold"),bg="#3f51b5",fg="white").pack(side=TOP,fill=X)

lbl\_reset=Label(self.forget\_password\_window,text="Enter the OTP sent on the Registered Email",font=("times new roman",15)).place(x=20,y=60)

txt\_reset=Entry(self.forget\_password\_window,textvariable=self.var\_otp,font=("times new roman",15),bg="lightyellow").place(x=20,y=100,width=250,height=30)

self.btn\_reset=Button(self.forget\_password\_window,text="SUBMIT/RESET",command=self.validate\_otp,font=("times new roman",10),bg="lightblue")

self.btn\_reset.place(x=280,y=100,width=100,height=30)

#self.btn\_reset.after(500,self.send\_email('sppr.8448@gmail.com'))

lbl\_new\_pass=Label(self.forget\_password\_window,text="New Password",font=("times new roman",15)).place(x=20,y=160)

txt\_new\_pass=Entry(self.forget\_password\_window,textvariable=self.var\_new\_pass,font=("times new roman",15),bg="lightyellow").place(x=20,y=190,width=250,height=30)

lbl\_con\_pass=Label(self.forget\_password\_window,text="Confirm Password",font=("times new roman",15)).place(x=20,y=225)

txt\_con\_pass=Entry(self.forget\_password\_window,textvariable=self.var\_con\_pass,show="\*",font=("times new roman",15),bg="lightyellow").place(x=20,y=255,width=250,height=30)

self.btn\_update=Button(self.forget\_password\_window,text="UPDATE",command=self.update\_pass,font=("times new roman",15),state=DISABLED,bg="forestgreen",fg="azure")

self.btn\_update.place(x=150,y=300,width=100,height=50)

except Exception as ex:

messagebox.showerror("Error",f"Error due to : {str(ex)}",parent=self.root)

#messagebox.showinfo("Forgot Password","You have selected Forgot Password")

def sign\_up(self):

con=sqlite3.connect(database=r'ims.db')

cur=con.cursor()

try:

self.forget\_password\_window=Toplevel(self.root)

self.forget\_password\_window.title("SIGN UP")

self.forget\_password\_window.geometry("1000x500+100+60")

self.forget\_password\_window.config(bg="white")

self.forget\_password\_window.focus\_force()

#===Employee Entry=====

self.var\_new\_emp\_id=StringVar()

self.var\_new\_gender=StringVar()

self.var\_new\_contact=StringVar()

self.var\_new\_name=StringVar()

self.var\_new\_dob=StringVar()

self.var\_new\_doj=StringVar()

self.var\_new\_email=StringVar()

self.var\_new\_pass=StringVar()

self.var\_new\_utype=StringVar()

self.var\_new\_salary=StringVar()

self.var\_new\_address=StringVar()

title=Label(self.forget\_password\_window,text="Employee Entry Details",font=("goudy old style",15),bg="#0f4d7d",fg="white").place(x=10,y=10,width=1000)

#====content=======

#===row1====

lbl\_new\_empid=Label(self.forget\_password\_window,text="Emp ID",font=("goudy old style",15),bg="white").place(x=50,y=50)

lbl\_new\_name=Label(self.forget\_password\_window,text="Name",font=("goudy old style",15),bg="white").place(x=500,y=50)

txt\_new\_empid=Entry(self.forget\_password\_window,textvariable=self.var\_new\_emp\_id,font=("goudy old style",15),bg="lightyellow").place(x=150,y=50,width=250)

txt\_new\_name=Entry(self.forget\_password\_window,textvariable=self.var\_new\_name,font=("goudy old style",15),bg="lightyellow").place(x=600,y=50,width=250)

#===row2====

lbl\_new\_contact=Label(self.forget\_password\_window,text="Contact",font=("goudy old style",15),bg="white").place(x=50,y=100)

lbl\_new\_email=Label(self.forget\_password\_window,text="Email",font=("goudy old style",15),bg="white").place(x=500,y=100)

txt\_new\_contact=Entry(self.forget\_password\_window,textvariable=self.var\_new\_contact,font=("goudy old style",15),bg="lightyellow").place(x=150,y=100,width=200)

txt\_new\_email=Entry(self.forget\_password\_window,textvariable=self.var\_new\_email,font=("goudy old style",15),bg="lightyellow").place(x=600,y=100,width=200)

#===row3===

lbl\_new\_dob=Label(self.forget\_password\_window,text="D.O.B",font=("goudy old style",15),bg="white").place(x=50,y=150)

lbl\_new\_doj=Label(self.forget\_password\_window,text="D.O.J",font=("goudy old style",15),bg="white").place(x=500,y=150)

txt\_new\_dob=Entry(self.forget\_password\_window,textvariable=self.var\_new\_dob,font=("goudy old style",15),bg="lightyellow").place(x=150,y=150,width=200)

txt\_new\_doj=Entry(self.forget\_password\_window,textvariable=self.var\_new\_doj,font=("goudy old style",15),bg="lightyellow").place(x=600,y=150,width=200)

#===row4===

lbl\_new\_gender=Label(self.forget\_password\_window,text="Gender",font=("goudy old style",15),bg="white").place(x=50,y=200)

lbl\_new\_utype=Label(self.forget\_password\_window,text="User Type",font=("goudy old style",15),bg="white").place(x=500,y=200)

cmb\_new\_gender=ttk.Combobox(self.forget\_password\_window,textvariable=self.var\_new\_gender,values=("Select","Male","Female","Other"),state='readonly',justify=CENTER,font=("goudy old style",15))

cmb\_new\_gender.place(x=150,y=200,width=200)

cmb\_new\_gender.current(0)

cmb\_new\_utype=ttk.Combobox(self.forget\_password\_window,textvariable=self.var\_new\_utype,values=("Select","Admin","Employee"),state='readonly',justify=CENTER,font=("goudy old style",15))

cmb\_new\_utype.place(x=600,y=200,width=200)

cmb\_new\_utype.current(0)

#===row5====

lbl\_new\_salary=Label(self.forget\_password\_window,text="Salary",font=("goudy old style",15),bg="white").place(x=50,y=250)

lbl\_new\_pass=Label(self.forget\_password\_window,text="Password",font=("goudy old style",15),bg="white").place(x=500,y=250)

txt\_new\_pass=Entry(self.forget\_password\_window,textvariable=self.var\_new\_pass,font=("goudy old style",15),bg="lightyellow").place(x=150,y=250,width=200)

txt\_new\_salary=Entry(self.forget\_password\_window,textvariable=self.var\_new\_salary,font=("goudy old style",15),bg="lightyellow").place(x=600,y=250,width=200)

#===row6====

lbl\_new\_address=Label(self.forget\_password\_window,text="Address",font=("goudy old style",15),bg="white").place(x=50,y=300)

self.new\_txt\_address=Text(self.forget\_password\_window,font=("goudy old style",15),bg="lightyellow")

self.new\_txt\_address.place(x=150,y=300,width=300,height=100)

#===buttons====

btn\_new\_add=Button(self.forget\_password\_window,text="Add User",command=self.new\_add,font=("goudy old style",15),bg="#2196f3",fg="white",cursor="hand2").place(x=50,y=450,width=200,height=28)

except Exception as ex:

messagebox.showerror("Error",f"Error due to : {str(ex)}",parent=self.root)

def new\_add(self):

con=sqlite3.connect(database=r'ims.db')

cur=con.cursor()

try:

if self.var\_new\_emp\_id.get()=="": #or self.var\_name.get()=="" or self.var\_email.get()=="" or self.var\_gender.get()=="" or self.var\_contact.get()=="" or self.var\_dob.get()=="" or self.var\_doj.get()=="" or self.var\_pass.get()=="" or self.var\_utype.get()=="" or self.var\_address.get()=="" or self.var\_salary.get()=="":

messagebox.showerror("Error","All Fields must be required",parent=self.forget\_password\_window)

else:

cur.execute("Select \* from employee where eid=?",(self.var\_new\_emp\_id.get(),))

row=cur.fetchone()

if row!=None:

messagebox.showerror("Error","This Employee ID is Alreary Assigned, try different",parent=self.root)

else:

cur.execute("Insert into employee (eid,name,email,gender,contact,dob,doj,pass,utype,address,salary) values(?,?,?,?,?,?,?,?,?,?,?)",(

self.var\_new\_emp\_id.get(),

self.var\_new\_name.get(),

self.var\_new\_email.get(),

self.var\_new\_gender.get(),

self.var\_new\_contact.get(),

self.var\_new\_dob.get(),

self.var\_new\_doj.get(),

self.var\_new\_salary.get(),

self.var\_new\_utype.get(),

self.new\_txt\_address.get('1.0',END),

self.var\_new\_pass.get(),

#self.var\_address.get()

))

con.commit()

messagebox.showinfo("Succes","Employee Added Successfully",parent=self.forget\_password\_window)

messagebox.showinfo("Success","Sign Up Successful")

self.forget\_password\_window.destroy()

except Exception as ex:

messagebox.showerror("Error",f"Error due to : {str(ex)}",parent=self.forget\_password\_window)

# def send\_email(self,to\_):

s=smtplib.SMTP('smtp.gmail.com',587)

# s.starttls()

# email\_=email\_pass.email\_

# pass\_=email\_pass.pass\_

# s.login(email\_,pass\_)

self.otp=int(time.strftime("%H%S%M"))+int(time.strftime("%M%H%S"))

#print(self.otp)

subj='IMS-Reset Password OTP'

msg=f'Dear Sir/Madam\n\nYour Reset OTP is {str(self.otp)}.\n\nWith Regard,\nIMS Team'

msg="Subject:{}\n\n{}".format(subj,msg)

# s.sendmail(email\_,to\_,msg)

chk=s.ehlo()

if chk[0]==250:

return 'Success'

else:

return 'Failure'

# def update\_pass(self):

# if self.var\_new\_pass.get()=="" or self.var\_con\_pass.get()=="":

# messagebox.showerror("Error","Password is Required",parent=self.forget\_password\_window)

# elif self.var\_new\_pass.get()!=self.var\_con\_pass.get():

# messagebox.showerror("Error","New Password & Confirm Password should be same",parent=self.forget\_password\_window)

# else:

# con=sqlite3.connect(database=r'ims.db')

# cur=con.cursor()

# try:

# cur.execute("Update employee set pass=? where eid=?",(self.var\_new\_pass.get(),self.var\_emp\_id.get()))

# con.commit()

# messagebox.showinfo("Success","Password Canged Successfully")

# except Exception as ex:

# messagebox.showerror("Error",f"Error due to : {str(ex)}",parent=self.root)

def validate\_otp(self):

if int(self.otp)==int(self.var\_otp.get()):

self.btn\_update.config(state=NORMAL)

self.btn\_reset.config(state=DISABLED)

else:

messagebox.showerror("Error","Invalid OTP\tTry Again",parent=self.forget\_password\_window)

#if \_\_name\_\_=="\_\_main\_\_":

root=Tk()

obj=Login\_System(root)

root.mainloop()

1. **Dashboard file**

from tkinter import \*

import tkinter as tk

from PIL import Image,ImageTk

import sqlite3

import tkinter.messagebox

import datetime

import time

import math

import os

import tempfile

import random

from employee import employeeclass

from category import categoryClass

from supplier import supplierClass

date = datetime.datetime.now().date()

class homepage:

def \_\_init\_\_(self,root):

self.root=root

# heading

self.heading = Label(text="SK SALES CORPORATIONS", font="arial 40 bold",anchor="w",fg="white", bg="grey30")

self.heading.place(x=0, y=0, relwidth=1,height=70)

self.log\_out = Button(text='Logout',width=30, height=2, bg='red',fg="black",

font=("arial 18 bold"),command=self.logout,cursor="hand2")

self.log\_out.place(x=1150, y=10,width=150,height=50)

# clock

self.lbl\_clock = Label(text="Welcome to SK SALES CORPORATIONS\t\t\t\t\t\t\t\t\t\t\t Date: "+ str(date) +"", font=("times new roman",15),fg="black", bg="lightgrey")

self.lbl\_clock.place(x=0, y=70, relwidth=1,height=30)

#left menu

# leftmenu = Frame(root,bd=2,relief=RIDGE)

# leftmenu.place(x=0,y=102,width=200,height=450)

self.menulogo = Image.open("image/icon.png")

self.menulogo = self.menulogo.resize((220,130),Image.LANCZOS)

self.menulogo=ImageTk.PhotoImage(self.menulogo)

leftmenu=Frame(self.root,bd=2,relief=RIDGE,bg="white")

leftmenu.place(x=0,y=100,width=220,height=595)

lbl\_menulogo=Label(leftmenu,bg="white",image=self.menulogo)

lbl\_menulogo.pack(side=TOP,fill=X)

self.lbl\_menu = Label(leftmenu,text='Menu',width=30, height=2,fg="white",font=("times new roman",20),bg="maroon").pack(side=TOP,fill=X)

btn\_employee = Button(leftmenu,text='Employee',width=30,command=self.employee,height=1,fg="black",font=("times new roman",20,"bold"),bg="white",bd=3,cursor="hand2").pack(side=TOP,fill=X)

btn\_supplier = Button(leftmenu,text='Supplier',width=30,command=self.supplier, height=1,fg="black",font=("times new roman",20,"bold"),bg="white",bd=3,cursor="hand2").pack(side=TOP,fill=X)

btn\_category = Button(leftmenu,text='Category',width=30,command=self.category, height=1,fg="black",font=("times new roman",20,"bold"),bg="white",bd=3,cursor="hand2").pack(side=TOP,fill=X)

btn\_ad\_products = Button(leftmenu,text='Add Products',width=30,command=self.adding, height=1,fg="black",font=("times new roman",20,"bold"),bg="white",bd=3,cursor="hand2").pack(side=TOP,fill=X)

btn\_up\_products = Button(leftmenu,text='Update Product',width=30,command=self.updating, height=1,fg="black",font=("times new roman",20,"bold"),bg="white",bd=3,cursor="hand2").pack(side=TOP,fill=X)

btn\_sales = Button(leftmenu,text='Sales',width=30,command=self.open\_bill\_page, height=1,fg="black",font=("times new roman",20,"bold"),bg="white",bd=3,cursor="hand2").pack(side=TOP,fill=X)

btn\_exit = Button(leftmenu,text='Exit',width=30,command=self.exit, height=1,fg="black",font=("times new roman",20,"bold"),bg="white",bd=3,cursor="hand2").pack(side=TOP,fill=X)

#content

self.lbl\_employee = Label(root,text="Total Employee \n[ 0 ]",bg="steelblue",fg="white", bd=5, relief=RIDGE,font=("arial",20,"bold"))

self.lbl\_employee.place(x=300,y=150,width=300,height=150)

self.lbl\_supplier = Label(root,text="Total Supplier\n [ 0 ]",bg="yellow",fg="black", bd=5, relief=RIDGE,font=("arial",20,"bold"))

self.lbl\_supplier.place(x=650,y=150,width=300,height=150)

self.lbl\_add\_pd = Label(root,text="Total Products\n [ 0 ]",bg="green",fg="white", bd=5, relief=RIDGE,font=("arial",20,"bold"))

self.lbl\_add\_pd.place(x=1000,y=150,width=300,height=150)

self.lbl\_category = Label(root,text="Total Employee \n[ 0 ]",bg="violet",fg="white", bd=5, relief=RIDGE,font=("arial",20,"bold"))

self.lbl\_category.place(x=300,y=340,width=300,height=150)

self.update\_content()

#Update content

def update\_content(self):

conn = sqlite3.connect("C:\store management\Database\store.db")

cur=conn.cursor()

try:

#===Employee====

cur.execute("Select \* from employee")

employee=cur.fetchall()

#print(len(employee))

self.lbl\_employee.config(text=f"Total Employee \n[{str(len(employee))}]")

#===Supplier===

cur.execute("select \* from supplier")

supplier=cur.fetchall()

self.lbl\_supplier.config(text=f"Total Supplier \n[{str(len(supplier))}]")

#===Category===

cur.execute("select \* from category")

category=cur.fetchall()

self.lbl\_category.config(text=f"Total Category \n[{str(len(category))}]")

#===Products====

cur.execute("select \* from inventory")

inventory=cur.fetchall()

self.lbl\_add\_pd.config(text=f"Total Products \n[{str(len(inventory))}]")

except Exception as ex:

tkinter.messagebox.showerror("Error",f"Error due to : {str(ex)}",parent=self.root)

def employee(self):

self.new\_win=Toplevel(self.root)

self.new\_obj=employeeclass(self.new\_win)

def supplier(self):

self.new\_win=Toplevel(self.root)

self.new\_obj=supplierClass(self.new\_win)

def category(self):

self.new\_win=Toplevel(self.root)

self.new\_obj=categoryClass(self.new\_win)

def adding(self):

# self.master.destroy()

os.system("python add\_to\_db.py")

def updating(self):

# self.master.destroy()

os.system("python update.py")

def open\_bill\_page(self):

# self.master.destroy()

os.system("python main\_file.py")

def exit(self):

self.root.destroy()

def logout(self):

root.destroy()

os.system("python login.py")

if \_\_name\_\_ == "\_\_main\_\_":

root = tk.Tk()

b = homepage(root)

root.geometry("1366x700+0+0")

root.title("Home Page")

# root.config(bg="white")

root.mainloop()

1. **Add to Database file**

#import all modules

from tkinter import \*

import tkinter as tk

import sqlite3

import tkinter.messagebox

from tkinter import ttk,messagebox

conn = sqlite3.connect("C:\store management\Database\store.db")

c = conn.cursor()

result = c.execute("SELECT Max(id) from inventory")

for r in result:

id = r[0]

class Database:

def \_\_init\_\_(self,master,\*args,\*\*kwargs):

self.master = master

self.heading = Label(master,text="Add Products", font=('arial 35 bold'),fg='black')

self.heading.place(x=550,y=0)

# self.btn\_clear = Button(master, text="return", width=18, height=2, bg='red',fg='white',command=self.clear\_all)

# self.btn\_clear.place(x=0, y=10)

#label and entries for the window

self.category=Label(master,text="Enter Category",font=("arial 18 bold")).place(x=0,y=50)

# self.name\_l = Label(master, text="Enter the Product Name", font=("arial 18 bold"))

# self.name\_l.place(x=0, y=100)

self.sr\_no\_l = Label(master, text="Enter the Serial Number", font=("arial 18 bold"))

self.sr\_no\_l.place(x=0, y=100)

self.stock\_l = Label(master, text="Enter Stock In Meter", font=("arial 18 bold"))

self.stock\_l.place(x=0, y=150)

self.cp\_l = Label(master, text="Enter Cost price", font=("arial 18 bold"))

self.cp\_l.place(x=0, y=200)

self.sp\_l = Label(master, text="Enter Selling price", font=("arial 18 bold"))

self.sp\_l.place(x=0, y=250)

self.vendor\_l = Label(master, text="Enter the Vendor Name", font=("arial 18 bold"))

self.vendor\_l.place(x=0, y=300)

self.vendor\_phone\_l = Label(master, text="Enter the Vendor Number", font=("arial 18 bold"))

self.vendor\_phone\_l.place(x=0, y=350)

self.id\_l = Label(master, text="Enter ID", font=("arial 18 bold"))

self.id\_l.place(x=0, y=400)

self.status=Label(master,text="Status",font=("arial 18 bold"))

self.status.place(x=0,y=450)

#entries for the labels

self.category\_e=ttk.Combobox(master, width=24,values=("Select","Rexine","Velvet","Carpet"),state='readonly',justify=CENTER,font=("arial 18"))

self.category\_e.place(x=380,y=55)

self.category\_e.current(0)

# self.name\_e = Entry(master,width=25,font=("arial 18 bold"))

# self.name\_e.place(x=380,y=100)

self.sr\_no\_e = Entry(master,width=25,font=("arial 18 bold"))

self.sr\_no\_e.place(x=380,y=100)

self.stock\_e = Entry(master,width=25,font=("arial 18 bold"))

self.stock\_e.place(x=380,y=150)

self.cp\_e = Entry(master,width=25,font=("arial 18 bold"))

self.cp\_e.place(x=380,y=200)

self.sp\_e = Entry(master,width=25,font=("arial 18 bold"))

self.sp\_e.place(x=380,y=250)

self.vendor\_e = Entry(master,width=25,font=("arial 18 bold"))

self.vendor\_e.place(x=380,y=300)

self.vendor\_phone\_e = Entry(master,width=25,font=("arial 18 bold"))

self.vendor\_phone\_e.place(x=380,y=350)

self.id\_e = Entry(master,width=25,font=("arial 18 bold"))

self.id\_e.place(x=380,y=400)

self.status\_e=ttk.Combobox(master,width=24,values=("Select","Active","Inactive"),state='readonly',justify=CENTER,font=("arial 18"))

self.status\_e.place(x=380,y=450)

self.status\_e.current(0)

#button to add to the database

self.btn\_add = Button(master,text="Add to Database",width=25,height=2,bg="black",fg="white",command=self.get\_items)

self.btn\_add.place(x=550,y=520)

self.btn\_clear = Button(master, text="Clear All Fields", width=18, height=2, bg='red',fg='white',command=self.clear\_all)

self.btn\_clear.place(x=380, y=520)

#text box for logs

self.tBox = Text(master,width=60,height=21)

self.tBox.place(x=750,y=100)

self.tBox.insert(END, "ID has reached upto: " + str(id))

def get\_items(self,\*args,\*\*kwargs):

#get from Entries

self.category = self.category\_e.get()

# self.name = self.name\_e.get()

self.sr\_no = self.sr\_no\_e.get()

self.stock = self.stock\_e.get()

self.cp = self.cp\_e.get()

self.sp = self.sp\_e.get()

self.vendor = self.vendor\_e.get()

self.vendor\_phone = self.vendor\_phone\_e.get()

self.status = self.status\_e.get()

#dynamic entries

self.totalcp = float(self.cp) \* float(self.stock) if self.cp and self.stock else 0.0

try:

self.totalsp = float(self.sp) \* float(self.stock)

except ValueError:

self.totalsp = 0.0

self.assumed\_profit = float(self.totalsp) - float(self.totalcp)

if self.category == ''or self.sr\_no == '' or self.stock == '' or self.cp == '' or self.sp == '':

tkinter.messagebox.showinfo("Error","please fill all the entries")

else:

sql = "INSERT INTO inventory (category, sr\_no, stock, cp, sp, totalcp, totalsp, assumed\_profit, vendor, vendor\_phone,status) VALUES(?,?,?,?,?,?,?,?,?,?,?)"

c.execute(sql,(self.category,self.sr\_no,self.stock,self.cp,self.sp,self.totalcp,self.totalsp,self.assumed\_profit,self.vendor,self.vendor\_phone,self.status))

conn.commit()

# textBox insert

self.tBox.insert(END,"\n \nInserted " + str(self.category) + " Into the database with code " + str(self.id\_e.get()))

tkinter.messagebox.showinfo("Success","Sucessfully added to the database")

def clear\_all(self,\*args,\*\*kwargs):

self.category\_e.delete(0, END)

# self.name\_e.delete(0, END)

self.sr\_no\_e.delete(0, END)

self.stock\_e.delete(0, END)

self.cp\_e.delete(0, END)

self.sp\_e.delete(0, END)

self.vendor\_e.delete(0, END)

self.vendor\_phone\_e.delete(0, END)

self.id\_e.delete(0, END)

self.status\_e.delete(0,END)

root=Tk()

b = Database(root)

root.geometry("1366x768+0+100")

root.title("Add to the database")

root.mainloop()

**4.Update Database file**

#import all modules

from tkinter import \*

import sqlite3

import tkinter.messagebox

from tkinter import ttk,messagebox

conn = sqlite3.connect("C:\store management\Database\store.db")

c = conn.cursor()

result = c.execute("SELECT Max(id) from inventory")

for r in result:

id = r[0]

class Database:

def \_\_init\_\_(self,master,\*args,\*\*kwargs):

self.master = master

self.heading = Label(master,text="Update Products", font=('arial 40 bold'),fg='black')

self.heading.place(x=400,y=0)

#label and Entry for id

self.id\_le = Label(master,text="Enter id", font=("arial 18 bold"))

self.id\_le.place(x=0,y=70)

self.id\_leb = Entry(master, font=("arial 18 bold"), width=10)

self.id\_leb.place(x=380, y=70)

self.btn\_search = Button(master, text="Search", width=15, height=2, bg="orange", command=self.search)

self.btn\_search.place(x=550,y=68)

#label and entries for the window

# self.name\_l = Label(master, text="Enter the Product Name", font=("arial 18 bold"))

# self.name\_l.place(x=0, y=120)

self.category=Label(master,text="Enter Category",font=("arial 18 bold")).place(x=0, y=120)

self.sr\_no\_l = Label(master, text="Enter the Serial Number", font=("arial 18 bold"))

self.sr\_no\_l.place(x=0, y=170)

self.stock\_l = Label(master, text="Enter Stock In Meter", font=("arial 18 bold"))

self.stock\_l.place(x=0, y=220)

self.cp\_l = Label(master, text="Enter Cost price", font=("arial 18 bold"))

self.cp\_l.place(x=0, y=270)

self.sp\_l = Label(master, text="Enter Selling price", font=("arial 18 bold"))

self.sp\_l.place(x=0, y=320)

self.totalcp\_l = Label(master, text="Enter Total cost price", font=("arial 18 bold"))

self.totalcp\_l.place(x=0, y=370)

self.totalsp\_l = Label(master, text="Enter Total Selling price", font=("arial 18 bold"))

self.totalsp\_l.place(x=0, y=420)

self.vendor\_l = Label(master, text="Enter the Vendor Name", font=("arial 18 bold"))

self.vendor\_l.place(x=0, y=470)

self.vendor\_phone\_l = Label(master, text="Enter the Vendor Number", font=("arial 18 bold"))

self.vendor\_phone\_l.place(x=0, y=520)

self.status=Label(master,text="Status",font=("arial 18 bold"))

self.status.place(x=0,y=570)

#entries for the labels

# self.name\_e = Entry(master,width=25,font=("arial 18 bold"))

# self.name\_e.place(x=380,y=120)

self.category\_e=ttk.Combobox(master, width=24,values=("Select","Rexine","Velvet","Carpet"),state='readonly',justify=CENTER,font=("arial 18"))

self.category\_e.place(x=380,y=120)

self.category\_e.current(0)

self.sr\_no\_e = Entry(master,width=25,font=("arial 18 bold"))

self.sr\_no\_e.place(x=380,y=170)

self.stock\_e = Entry(master,width=25,font=("arial 18 bold"))

self.stock\_e.place(x=380,y=220)

self.cp\_e = Entry(master,width=25,font=("arial 18 bold"))

self.cp\_e.place(x=380,y=270)

self.sp\_e = Entry(master,width=25,font=("arial 18 bold"))

self.sp\_e.place(x=380,y=320)

self.totalcp\_e = Entry(master,width=25,font=("arial 18 bold"))

self.totalcp\_e.place(x=380,y=370)

self.totalsp\_e = Entry(master,width=25,font=("arial 18 bold"))

self.totalsp\_e.place(x=380,y=420)

self.vendor\_e = Entry(master,width=25,font=("arial 18 bold"))

self.vendor\_e.place(x=380,y=470)

self.vendor\_phone\_e = Entry(master,width=25,font=("arial 18 bold"))

self.vendor\_phone\_e.place(x=380,y=520) self.status\_e=ttk.Combobox(master,width=24,values=("Select","Active","Inactive"),state='readonly',justify=CENTER,font=("arial 18"))

self.status\_e.place(x=380,y=570)

self.status\_e.current(0)

#button to add to the database

self.btn\_add = Button(master, text="Update Database",width=25,height=2,bg="steel blue",fg="white", command=self.update)

self.btn\_add.place(x=520,y=620)

#text box for logs

self.tBox = Text(master,width=60,height=21,font=("goudy old style", 12))

self.tBox.place(x=750,y=70)

self.tBox.insert(END,"\n FOR REXINE SEARCH ID FROM 0-100."+"\n\n FOR VELVET SEARCH ID FROM 101-200."+"\n\n FOR CARPET SEARCH ID FROM 201-250.")

def search(self, \*args, \*\*kwargs):

sql = "SELECT \* FROM inventory WHERE id=?"

result = c.execute(sql,(self.id\_leb.get(), ))

for r in result:

self.n1 = r[1] #category

self.n2 = r[2] #sr\_no

self.n3 = r[3] #stock

self.n4 = r[4] #cp

self.n5 = r[5] #sp

self.n6 = r[6] #totalcp

self.n7 = r[7] #totalsp

self.n8 = r[8] #assumed\_profit

self.n9 = r[9] #vendor

self.n10 = r[10] #vendor\_phone

self.n11 = r=[11] #status

conn.commit()

#insert into the entries to update

self.category\_e.delete(0, END)

self.category\_e.insert(0, str(self.n1))

self.sr\_no\_e.delete(0, END)

self.sr\_no\_e.insert(0, str(self.n2))

self.stock\_e.delete(0, END)

self.stock\_e.insert(0, str(self.n3))

self.cp\_e.delete(0, END)

self.cp\_e.insert(0, str(self.n4))

self.sp\_e.delete(0, END)

self.sp\_e.insert(0, str(self.n5))

self.vendor\_e.delete(0, END)

self.vendor\_e.insert(0, str(self.n9))

self.vendor\_phone\_e.delete(0, END)

self.vendor\_phone\_e.insert(0, str(self.n10))

self.totalcp\_e.delete(0, END)

self.totalcp\_e.insert(0, str(self.n6))

self.totalsp\_e.delete(0, END)

self.totalsp\_e.insert(0, str(self.n7))

self.status\_e.delete(0, END)

self.status\_e.insert(0, str(self.n11))

def update(self, \*args, \*\*kwargs):

#get all the update values

self.u1 = self.category\_e.get()

self.u2 = self.sr\_no\_e.get()

self.u3 = self.stock\_e.get()

self.u4 = self.cp\_e.get()

self.u5 = self.sp\_e.get()

self.u6 = self.totalcp\_e.get()

self.u7 = self.totalsp\_e.get()

self.u8 = self.vendor\_e.get()

self.u9 = self.vendor\_phone\_e.get()

self.u10 = self.status\_e.get()

query = "UPDATE inventory SET category=?, sr\_no=?, stock=?, cp=?, sp=?, totalcp=?, totalsp=?, vendor=?, vendor\_phone=?, status=? WHERE id=?"

c.execute(query, (self.u1, self.u2, self.u3, self.u4, self.u5, self.u6, self.u7, self.u8, self.u9,self.u10, self.id\_leb.get()))

conn.commit()

tkinter.messagebox.showinfo("Success", "Updated Database Successfully")

root=Tk()

b = Database(root)

root.geometry("1366x768+0+30")

root.title("Update the database")

root.mainloop()

1. **Main file**

#import all modules

from tkinter import \*

import sqlite3

import tkinter.messagebox

import datetime

import math

import os

import random

conn = sqlite3.connect("C:\store management\Database\store.db")

c = conn.cursor()#todays date

date = datetime.datetime.now().date()

#temporary lists like sessions

products\_list = []

product\_price = []

product\_quantity = []

product\_id = []

# list for labels

labels\_list = []

class Application:

def \_\_init\_\_(self, master, \*args, \*\*kwargs):

self.master = master

#Frames

self.left= Frame(master,width=720,height=780,bg="white")

self.left.pack(side=LEFT)

self.right= Frame(master,width=646,height=780,bg="lightgrey")

self.right.pack(side=RIGHT)

#components

self.heading = Label(self.left, text="SK SALES CORPORATION",font="arial 40 bold", bg="white",fg="black" )

self.heading.place(x=0,y=0) self.date\_l = Label(self.right, text="Today's date: " + str(date), font=("arial 16 bold"), bg='lightgrey',fg='black')

self.date\_l.place(x=0,y=0)

#table invoice

self.tsr\_no = Label(self.right,text="Serial Number",font=('arial 18 bold'), bg='lightgrey',fg='black')

self.tsr\_no.place(x=0,y=60)

self.tquantity = Label(self.right,text="Quantity",font=('arial 18 bold'), bg='lightgrey',fg='black')

self.tquantity.place(x=300,y=60)

self.tamount = Label(self.right,text="Amount",font=('arial 18 bold'), bg='lightgrey',fg='black')

self.tamount.place(x=500,y=60)

#enter stuff

self.enterid = Label(self.left,text="Enter product's ID", font=('arial 17 bold'),bg='white')

self.enterid.place(x=0,y=80) self.enteride = Entry(self.left, width=25,font=('arial 18 bold'), bg='lightgrey')

self.enteride.place(x=210, y=80)

self.enteride.focus()

#button

self.search\_btn = Button(self.left, text='Search', width=22, height=2,bg='black',fg="white", command=self.ajax)

self.search\_btn.place(x=370,y=120) #fill it later by the ajax

self.ser\_no = Label(self.left, text="", font=('arial 27 bold'),bg='white' )

self.ser\_no.place(x=0,y=220) self.pprice = Label(self.left, text="", font=('arial 27 bold'),bg='white')

self.pprice.place(x=0,y=270) #total label

self.total\_l = Label(self.right,text="",font=('arial 37 bold'),bg="lightgrey",fg='black')

self.total\_l.place(x=0,y=600)

def ajax(self,\*args,\*\*kwargs):

self.get\_id = self.enteride.get()

#get the sr\_no info with that id and fill it in the label above

query = "SELECT \* FROM inventory WHERE id=?"

result = c.execute(query, (self.get\_id, ))

for self.r in result:

self.get\_id = self.r[0]

self.get\_sr\_no = self.r[2]

self.get\_price = self.r[5]

self.get\_stock = self.r[3]

self.ser\_no.configure(text="Product sr\_no: " + str(self.get\_sr\_no))

self.pprice.configure(text="Price: Rs." + str(self.get\_price))

#create the quantity and discount table

self.quantity\_l = Label(self.left, text="Enter Quantity", font=('arial 18 bold'), bg='white')

self.quantity\_l.place(x=0,y=370)

self.quantity\_e = Entry(self.left, width=25, font=('arial 18 bold'), bg='lightgrey')

self.quantity\_e.place(x=180,y=370)

self.quantity\_e.focus() self.discount\_l = Label(self.left, text="Enter discount", font=('arial 18 bold'), bg='white')

self.discount\_l.place(x=0,y=410)

self.discount\_e = Entry(self.left, width=25, font=('arial 18 bold'), bg='lightgrey')

self.discount\_e.place(x=180,y=410)

self.discount\_e.insert(END,0)

#add to cart button

self.add\_to\_cart\_btn = Button(self.left, text='Add to cart', width=22, height=2,bg='black',fg="white",command=self.add\_to\_cart)

self.add\_to\_cart\_btn.place(x=350,y=450)

#generate bill and change

self.change\_l = Label(self.left, text='Given Amount', font=('arial 18 bold'),bg='white')

self.change\_l.place(x=0,y=550)

self.change\_e = Entry(self.left, width=25, font=('arial 18 bold'),bg='lightgrey')

self.change\_e.place(x=190,y=550)

#button change

self.change\_btn = Button(self.left, text='Calculate Change', width=22, height=2,bg='black',fg="white",command=self.change\_func)

self.change\_btn.place(x=350,y=590)

#Genarate bill button

self.bill\_btn = Button(self.left, text='Genarate bill', width=100, height=2,bg='black',fg='white',command=self.generate\_bill)

self.bill\_btn.place(x=0,y=640)

def add\_to\_cart(self,\*args,\*\*kwargs):

#get the quantitiy value from the database

self.quantity\_value = int(self.quantity\_e.get())

if self.quantity\_value > int(self.get\_stock):

tkinter.messagebox.showinfo("Error","Not that many product in our inventory..'AVAILABLE SOON'")

else:

#calculate the price

self.final\_price = (float(self.quantity\_value) \* float(self.get\_price)) - (float(self.discount\_e.get()))

products\_list.append(self.get\_sr\_no)

product\_price.append(self.final\_price)

product\_quantity.append(self.quantity\_value)

product\_id.append(self.get\_id)

self.x\_index = 0

self.y\_index = 100

self.counter = 0

for self.p in products\_list:

self.tempsr\_no = Label(self.right, text=str(products\_list[self.counter]),font=('arial 18 bold'),bg='lightgrey',fg='black')

self.tempsr\_no.place(x=0,y=self.y\_index)

labels\_list.append(self.tempsr\_no)

self.tempqt = Label(self.right, text=str(product\_quantity[self.counter]),font=('arial 18 bold'),bg='lightgrey',fg='black')

self.tempqt.place(x=300,y=self.y\_index)

labels\_list.append(self.tempqt)

self.tempprice = Label(self.right, text=str(product\_price[self.counter]),font=('arial 18 bold'),bg='lightgrey',fg='black')

self.tempprice.place(x=500,y=self.y\_index)

labels\_list.append(self.tempprice)

self.y\_index += 40

self.counter += 1 #total configure

self.total\_l.configure(text="Total: " + str(sum(product\_price)))

#delete

self.quantity\_l.place\_forget()

self.quantity\_e.place\_forget()

self.discount\_l.place\_forget()

self.discount\_e.place\_forget()

self.ser\_no.configure(text='')

self.pprice.configure(text='')

self.add\_to\_cart\_btn.destroy()

def change\_func(self,\*args,\*\*kwargs):

#get the amount given by the customer and the amount genarated by the computer

self.amount\_given = float(self.change\_e.get())

self.our\_total = float(sum(product\_price))

self.to\_give = self.amount\_given - self.our\_total

#label change

self.c\_amount = Label(self.left, text="Change: Rs. " + str(self.to\_give),font=('arial 18 bold'),fg='red',bg='white')

self.c\_amount.place(x=0,y=600) def generate\_bill(self, \*args, \*\*kwargs):

#creating the bill before updating to the database

directory = "C:/store management/Invoice/"+str(date)+ "/"

if not os.path.exists(directory):

os.makedirs(directory)

#templates for the bill

company = "\t\t\t\tSK Sales Corporation Pvt. Ltd.\n"

address = "\t\t\t\tSakinaka, Mumbai, 400072, Maharahtra.\n"

phone = "\t\t\t\t9819848323"

dt = "\t\t\t\t" + str(date) +"\n"

sample = "\t\t\t\tInvoice"

table\_header = "\n\n\t\t\t---------------------------------------------\n\t\t\tSN.\t\tSR\_NO\t\tQty\t\tAmount\n\n\t\t\t----------------------------------------------"

final = company + address + phone + dt + sample + "\n" + table\_header

#open a file to write

file\_name = str(directory)+str(random.randrange(5000, 10000)) + ".rtf"

f = open(file\_name, 'w')

f.write(final)

#fill dynamics

r = 1

i = 0

for t in products\_list:

f.write("\n\t\t\t" + str(r) + "\t\t" + str(products\_list[i]) + "\t\t" + str(product\_quantity[i]) + "\t\t" + str(product\_price[i]))

i += 1

r += 1

f.write("\n\n\n\t\t\t\t\t\t\tTotal: Rs. " + str(sum(product\_price)))

f.write("\n\t\t\t\t\t\t\tThanks for buying product ")

f.write("\n\t\t\t---------------------------------------------")

os.startfile(file\_name,"print")

f.close()

#decrease the stock

self.x = 0

initial = "SELECT \* FROM inventory WHERE id =?"

result = c.execute(initial, (product\_id[self.x], ))

for i in products\_list:

for r in result:

self.old\_stock = r[2]

self.new\_stock = int(self.old\_stock) - int(product\_quantity[self.x])

# update the stock

sql = "UPDATE inventory SET stock=? WHERE id = ?"

c.execute(sql,(self.new\_stock, product\_id[self.x]))

conn.commit()

# insert into the transaction

sql2 = "INSERT INTO transactions (product\_name, quantity, amount, date) VALUES (?,?,?,?)"

c.execute(sql2, (products\_list[self.x], product\_quantity[self.x], product\_price[self.x], date))

conn.commit()

self.x += 1

for a in labels\_list:

a.destroy()

del(products\_list[:])

del(product\_id[:])

del(product\_quantity[:])

del(product\_price[:])

self.total\_l.configure(text="")

self.c\_amount.configure(text="")

self.change\_e.delete(0,END)

self.enteride.delete(0,END)

tkinter.messagebox.showinfo("success","Done Everything smoothly")

root=Tk()

b = Application(root)

root.geometry("1366x768+0+20")

# root.title("Add to the database")

root.mainloop()

1. **Employee Page**

from tkinter import \*

import tkinter as tk

import sqlite3

import tkinter.messagebox

import datetime

import math

import os

from tkinter import \*

from PIL import Image,ImageTk #pip install pillow

from tkinter import ttk,messagebox

import sqlite3

class employeeclass:

def \_\_init\_\_(self,root):

self.root=root

self.root.geometry("1100x500+220+130")

self.root.title("Inventory Management System")

self.root.config(bg="#fafafa")

self.root.focus\_force()

#==============================

# All Variables======

self.var\_searchby=StringVar()

self.var\_searchtxt=StringVar()

self.var\_emp\_id=StringVar()

self.var\_gender=StringVar()

self.var\_contact=StringVar()

self.var\_name=StringVar()

self.var\_dob=StringVar()

self.var\_doj=StringVar()

self.var\_pass=StringVar()

self.var\_utype=StringVar()

#===searchFrame=====

# SearchFrame=LabelFrame(self.root,text="Search Employee",font=("goudy old style",12,"bold"),bd=2,relief=RIDGE,bg="#fafafa")

# SearchFrame.place(x=250,y=20,width=600,height=70)

#===options===

# cmb\_search=ttk.Combobox(SearchFrame,textvariable=self.var\_searchby,values=("Select","Name","Contact"),state='readonly',justify=CENTER,font=("goudy old style",15))

# cmb\_search.place(x=10,y=10,width=180)

# cmb\_search.current(0)

# txt\_search=Entry(SearchFrame,textvariable=self.var\_searchtxt,font=("goudy old style",15),bg="lightgrey").place(x=200,y=10)

# btn\_search=Button(SearchFrame,text="Search",command=self.search,font=("goudy old style",15),bg="steelblue",fg="white",cursor="hand2").place(x=410,y=9,width=150,height=30)

#===title====

title=Label(self.root,text="Employee Details",font=("goudy old style",15),bg="black",fg="white").place(x=50,y=10,width=1000)

#====content=======

#===row1====

lbl\_empid=Label(self.root,text="Emp ID",font=("goudy old style",15),bg="#fafafa").place(x=50,y=50)

lbl\_gender=Label(self.root,text="Gender",font=("goudy old style",15),bg="#fafafa").place(x=350,y=50)

lbl\_contact=Label(self.root,text="Contact",font=("goudy old style",15),bg="#fafafa").place(x=750,y=50)

txt\_empid=Entry(self.root,textvariable=self.var\_emp\_id,font=("goudy old style",15),bg="lightgrey").place(x=150,y=50,width=180) cmb\_gender=ttk.Combobox(self.root,textvariable=self.var\_gender,values=("Select","Male","Female","Other"),state='readonly',justify=CENTER,font=("goudy old style",15))

cmb\_gender.place(x=500,y=50,width=180)

cmb\_gender.current(0)

txt\_contact=Entry(self.root,textvariable=self.var\_contact,font=("goudy old style",15),bg="lightgrey").place(x=850,y=50,width=180)

#===row2====

lbl\_name=Label(self.root,text="Name",font=("goudy old style",15),bg="#fafafa").place(x=50,y=100)

lbl\_dob=Label(self.root,text="D.O.B",font=("goudy old style",15),bg="#fafafa").place(x=350,y=100)

lbl\_doj=Label(self.root,text="D.O.J",font=("goudy old style",15),bg="#fafafa").place(x=750,y=100)

txt\_name=Entry(self.root,textvariable=self.var\_name,font=("goudy old style",15),bg="lightgrey").place(x=150,y=100,width=180)

txt\_dob=Entry(self.root,textvariable=self.var\_dob,font=("goudy old style",15),bg="lightgrey").place(x=500,y=100,width=180)

txt\_doj=Entry(self.root,textvariable=self.var\_doj,font=("goudy old style",15),bg="lightgrey").place(x=850,y=100,width=180)

#===row3====

lbl\_pass=Label(self.root,text="Password",font=("goudy old style",15),bg="#fafafa").place(x=50,y=150)

lbl\_utype=Label(self.root,text="User Type",font=("goudy old style",15),bg="#fafafa").place(x=350,y=150)

txt\_pass=Entry(self.root,textvariable=self.var\_pass,font=("goudy old style",15),bg="lightgrey").place(x=150,y=150,width=180) cmb\_utype=ttk.Combobox(self.root,textvariable=self.var\_utype,values=("Select","Admin","Employee"),state='readonly',justify=CENTER,font=("goudy old style",15))

cmb\_utype.place(x=500,y=150,width=180)

cmb\_utype.current(0)

#===buttons====

btn\_add=Button(self.root,text="Save",command=self.add,font=("goudy old style",15),bg="violet",fg="white",cursor="hand2").place(x=300,y=250,width=110,height=40)

btn\_update=Button(self.root,text="Update",command=self.update,font=("goudy old style",15),bg="teal",fg="white",cursor="hand2").place(x=420,y=250,width=110,height=40)

btn\_delete=Button(self.root,text="Delete",command=self.delete,font=("goudy old style",15),bg="coral",fg="white",cursor="hand2").place(x=540,y=250,width=110,height=40)

btn\_clear=Button(self.root,text="Clear",command=self.clear,font=("goudy old style",15),bg="red",fg="white",cursor="hand2").place(x=660,y=250,width=110,height=40)

#====Employee Details===

emp\_frame=Frame(self.root,bd=3,relief=RIDGE)

emp\_frame.place(x=0,y=300,relwidth=1,height=200)

scrolly=Scrollbar(emp\_frame,orient=VERTICAL)

scrollx=Scrollbar(emp\_frame,orient=HORIZONTAL) self.EmployeeTable=ttk.Treeview(emp\_frame,columns=("eid","name","gender","contact","dob","doj","pass","u\_type"),yscrollcommand=scrolly.set,xscrollcommand=scrollx.set)

scrollx.pack(side=BOTTOM,fill=X)

scrolly.pack(side=RIGHT,fill=Y)

scrollx.config(command=self.EmployeeTable.xview)

scrolly.config(command=self.EmployeeTable.yview)

self.EmployeeTable.heading("eid",text="Employee ID")

self.EmployeeTable.heading("name",text="Name")

self.EmployeeTable.heading("gender",text="Gender")

self.EmployeeTable.heading("contact",text="Contact Number")

self.EmployeeTable.heading("dob",text="D.O.B")

self.EmployeeTable.heading("doj",text="D.O.J")

self.EmployeeTable.heading("pass",text="Password")

self.EmployeeTable.heading("u\_type",text="User Type")

self.EmployeeTable["show"]="headings"

self.EmployeeTable.pack(fill=BOTH,expand=1)

self.EmployeeTable.column("eid",width=90)

self.EmployeeTable.column("name",width=200)

self.EmployeeTable.column("gender",width=100)

self.EmployeeTable.column("contact",width=100)

self.EmployeeTable.column("dob",width=100)

self.EmployeeTable.column("doj",width=100)

self.EmployeeTable.column("pass",width=100)

self.EmployeeTable.column("u\_type",width=100)

self.EmployeeTable.pack(fill=BOTH,expand=1)

self.EmployeeTable.bind("<ButtonRelease-1>",self.get\_data)

self.show()

#===============================================================================

def add(self):

conn=sqlite3.connect("C:\store management\Database\store.db")

cur=conn.cursor()

try:

if self.var\_emp\_id.get()=="": #or self.var\_name.get()=="" or self.var\_email.get()=="" or self.var\_gender.get()=="" or self.var\_contact.get()=="" or self.var\_dob.get()=="" or self.var\_doj.get()=="" or self.var\_pass.get()=="" or self.var\_utype.get()=="" or self.var\_address.get()=="" or self.var\_salary.get()=="":

messagebox.showerror("Error","All Fields must be required",parent=self.root)

else:

cur.execute("Select \* from employee where eid=?",(self.var\_emp\_id.get(),))

row=cur.fetchone()

if row!=None:

messagebox.showerror("Error","This Employee ID is Alreary Assigned, try different",parent=self.root)

else:

cur.execute("Insert into employee (eid,name,gender,contact,dob,doj,pass,u\_type) values(?,?,?,?,?,?,?,?)",(

self.var\_emp\_id.get(),

self.var\_name.get(),

self.var\_gender.get(),

self.var\_contact.get(),

self.var\_dob.get(),

self.var\_doj.get(),

self.var\_pass.get(),

self.var\_utype.get(),

))

conn.commit()

messagebox.showinfo("Succes","Employee Added Successfully",parent=self.root)

self.show()

except Exception as ex:

messagebox.showerror("Error",f"Error due to : {str(ex)}",parent=self.root)

def show(self):

conn=sqlite3.connect("C:\store management\Database\store.db")

cur=conn.cursor()

try:

cur.execute("Select \* from employee")

rows=cur.fetchall()

self.EmployeeTable.delete(\*self.EmployeeTable.get\_children())

for row in rows:

self.EmployeeTable.insert('',END,values=row)

except Exception as ex:

messagebox.showerror("Error",f"Error due to : {str(ex)}",parent=self.root)

def get\_data(self,ev):

f=self.EmployeeTable.focus()

content=(self.EmployeeTable.item(f))

row=content['values']

#print(row)

self.var\_emp\_id.set(row[0])

self.var\_name.set(row[1])

self.var\_gender.set(row[2])

self.var\_contact.set(row[3])

self.var\_dob.set(row[4])

self.var\_doj.set(row[5])

self.var\_pass.set(row[6])

self.var\_utype.set(row[7])

def update(self):

conn=sqlite3.connect("C:\store management\Database\store.db")

cur=conn.cursor()

try:

if self.var\_emp\_id.get()=="": #or self.var\_name.get()=="" or self.var\_email.get()=="" or self.var\_gender.get()=="" or self.var\_contact.get()=="" or self.var\_dob.get()=="" or self.var\_doj.get()=="" or self.var\_pass.get()=="" or self.var\_utype.get()=="" or self.var\_address.get()=="" or self.var\_salary.get()=="":

messagebox.showerror("Error","All Fields must be required",parent=self.root)

else:

cur.execute("Select \* from employee where eid=?",(self.var\_emp\_id.get(),))

row=cur.fetchone()

if row==None:

messagebox.showerror("Error","Invalid Employee ID",parent=self.root)

else:

cur.execute("Update employee set name=?,gender=?,contact=?,dob=?,doj=?,pass=?,u\_type=? where eid=?",(

self.var\_name.get(),

self.var\_gender.get(),

self.var\_contact.get(),

self.var\_dob.get(),

self.var\_doj.get(),

self.var\_pass.get(),

self.var\_utype.get(),

self.var\_emp\_id.get()

))

conn.commit()

messagebox.showinfo("Succes","Employee Updated Successfully",parent=self.root)

self.show()

except Exception as ex:

messagebox.showerror("Error",f"Error due to : {str(ex)}",parent=self.root)

def delete(self):

conn=sqlite3.connect("C:\store management\Database\store.db")

cur=conn.cursor()

try:

if self.var\_emp\_id.get()=="": #or self.var\_name.get()=="" or self.var\_email.get()=="" or self.var\_gender.get()=="" or self.var\_contact.get()=="" or self.var\_dob.get()=="" or self.var\_doj.get()=="" or self.var\_pass.get()=="" or self.var\_utype.get()=="" or self.var\_address.get()=="" or self.var\_salary.get()=="":

messagebox.showerror("Error","All Fields must be required",parent=self.root)

else:

cur.execute("Select \* from employee where eid=?",(self.var\_emp\_id.get(),))

row=cur.fetchone()

if row==None:

messagebox.showerror("Error","Invalid Employee ID",parent=self.root)

else:

op=messagebox.askyesno("Confirm","Do you really want to delete?",parent=self.root)

if op==True:

cur.execute("delete from employee where eid=?",(self.var\_emp\_id.get(),))

conn.commit()

messagebox.showinfo("Delete","Employee Deleted Successfully",parent=self.root)

self.clear()

except Exception as ex:

messagebox.showerror("Error",f"Error due to : {str(ex)}",parent=self.root)

def clear(self):

self.var\_emp\_id.set("")

self.var\_name.set("")

self.var\_gender.set("Select")

self.var\_contact.set("")

self.var\_dob.set("")

self.var\_doj.set("")

self.var\_pass.set("")

self.var\_utype.set("Select")

self.var\_searchtxt.set("")

self.var\_searchby.set("Select")

self.show()

def search(self):

conn=sqlite3.connect("C:\store management\Database\store.db")

cur=conn.cursor()

try:

if self.var\_searchby.get()=="Select":

messagebox.showerror("Error","Select Search By Option",parent=self.root)

elif self.var\_searchtxt.get()=="":

messagebox.showerror("Error","Search Input is Required",parent=self.root)

else:

cur.execute("Select \* from employee where "+self.var\_searchby.get()+" LIKE '%"+self.var\_searchtxt.get()+"%'")

rows=cur.fetchall()

if len(rows)!=0:

self.EmployeeTable.delete(\*self.EmployeeTable.get\_children())

for row in rows:

self.EmployeeTable.insert('',END,values=row)

else:

messagebox.showerror("Error","No Record Found!!!",parent=self.root)

except Exception as ex:

messagebox.showerror("Error",f"Error due to : {str(ex)}",parent=self.root)

if \_\_name\_\_=="\_\_main\_\_":

root=Tk()

screen\_width = root.winfo\_screenwidth()

screen\_height = root.winfo\_screenheight()

root.resizable(screen\_width,screen\_height)

obj=employeeclass(root)

root.mainloop()

1. **Supplier page.**

from tkinter import \*

from PIL import Image,ImageTk #pip install pillow

from tkinter import ttk,messagebox

import sqlite3

class supplierClass:

def \_\_init\_\_(self,root):

self.root=root

self.root.geometry("1100x500+220+130")

self.root.title("Inventory Management System")

self.root.config(bg="#fafafa")

self.root.focus\_force()

#==============================

# All Variables======

self.var\_searchtxt=StringVar()

# self.var\_date=StringVar()

self.var\_supid=StringVar()

self.var\_company=StringVar()

self.var\_name=StringVar()

self.var\_contact=StringVar()

# self.var\_cat=StringVar()

# self.cat\_list=[]

# self.fetch\_cat()

# self.var\_desc=StringVar()

#===searchFrame=====

SearchFrame=LabelFrame(self.root,text="Search Supplier",font=("goudy old style",12),bd=2,relief=RIDGE,bg="#fafafa")

SearchFrame.place(x=470,y=60,width=600,height=90)

#===options===

lbl\_search=Label(SearchFrame,text="Search By Supplier ID",font=("goudy old style",15),bg="#fafafa")

lbl\_search.place(x=10,y=10,width=180)

txt\_search=Entry(SearchFrame,textvariable=self.var\_searchtxt,font=("goudy old style",15),bg="lightgrey").place(x=200,y=10,width=210) btn\_search=Button(SearchFrame,text="Search",command=self.search,font=("goudy old style",15),bg="steelblue",fg="white",cursor="hand2").place(x=430,y=10,width=150,height=30)

#===title====

title=Label(self.root,text="Supplier Details",font=("goudy old style",20,"bold"),bg="black",fg="white").place(x=50,y=10,width=1000,height=40)

#====content=======

#===row1====

# lbl\_date=Label(self.root,text="Date",font=("goudy old style bold",18),bg="#fafafa").place(x=50,y=100)

# txt\_date=Entry(self.root,textvariable=self.var\_date,font=("goudy old style bold",18),bg="lightgrey").place(x=220,y=100,width=210)

#===row2====

lbl\_supid=Label(self.root,text="Supplier ID",font=("goudy old style bold",18),bg="#fafafa").place(x=50,y=150)

txt\_supid=Entry(self.root,textvariable=self.var\_supid,font=("goudy old style bold",18),bg="lightgrey").place(x=220,y=150,width=210)

#===row3====

lbl\_company=Label(self.root,text="Company",font=("goudy old style bold",18),bg="#fafafa").place(x=50,y=200)

txt\_company=Entry(self.root,textvariable=self.var\_company,font=("goudy old style bold",18),bg="lightgrey").place(x=220,y=200,width=210)

#===row4====

lbl\_name=Label(self.root,text="Supplier Name",font=("goudy old style bold",18),bg="#fafafa").place(x=50,y=250)

txt\_name=Entry(self.root,textvariable=self.var\_name,font=("goudy old style bold",18),bg="lightgrey").place(x=220,y=250,width=210)

#===row5====

lbl\_contact=Label(self.root,text="Contact",font=("goudy old style bold",18),bg="#fafafa").place(x=50,y=300)

txt\_contact=Entry(self.root,textvariable=self.var\_contact,font=("goudy old style bold",18),bg="lightgrey").place(x=220,y=300,width=210)

#===buttons====

btn\_add=Button(self.root,text="Save",command=self.add,font=("goudy old style",15),bg="violet",fg="white",cursor="hand2").place(x=20,y=450,width=130,height=40)

btn\_update=Button(self.root,text="Update",command=self.update,font=("goudy old style",15),bg="teal",fg="white",cursor="hand2").place(x=160,y=450,width=130,height=40)

btn\_delete=Button(self.root,text="Delete",command=self.delete,font=("goudy old style",15),bg="coral",fg="white",cursor="hand2").place(x=300,y=450,width=130,height=40)

btn\_clear=Button(self.root,text="Clear",command=self.clear,font=("goudy old style",15),bg="red",fg="white",cursor="hand2").place(x=440,y=450,width=130,height=40)

# btn\_sup\_apmc=Button(self.root,text="APMC Supplier",command=self.sup\_apmc,font=("goudy old style",15,"bold"),bg="white",cursor="hand2").place(x=10,y=450,width=300,height=30) #image=self.icon\_side

#====Supplier Details===

sup\_frame=Frame(self.root,bd=3,relief=RIDGE)

sup\_frame.place(x=580,y=180,width=490,height=290)

scrolly=Scrollbar(sup\_frame,orient=VERTICAL)

scrollx=Scrollbar(sup\_frame,orient=HORIZONTAL) self.SupplierTable=ttk.Treeview(sup\_frame,columns=("supid","company","name","contact"),yscrollcommand=scrolly.set,xscrollcommand=scrollx.set)

scrollx.pack(side=BOTTOM,fill=X)

scrolly.pack(side=RIGHT,fill=Y)

scrollx.config(command=self.SupplierTable.xview)

scrolly.config(command=self.SupplierTable.yview)

# self.var\_date.set("")

# self.SupplierTable.heading("date",text="Date")

self.SupplierTable.heading("supid",text="Supplier ID")

self.SupplierTable.heading("company",text="Company")

self.SupplierTable.heading("name",text="Name")

self.SupplierTable.heading("contact",text="Contact")

self.SupplierTable["show"]="headings"

self.SupplierTable.pack(fill=BOTH,expand=1)

# self.SupplierTable.column("date",width=80)

self.SupplierTable.column("supid",width=90)

self.SupplierTable.column("company",width=100)

self.SupplierTable.column("name",width=200)

self.SupplierTable.column("contact",width=100)

self.SupplierTable.pack(fill=BOTH,expand=1)

self.SupplierTable.bind("<ButtonRelease-1>",self.get\_data)

self.show()

#===============================================================================

def sup\_apmc(self):

self.new\_win=Toplevel(self.root)

# self.new\_obj=sup\_apmcClass(self.new\_win)

def add(self):

conn=sqlite3.connect("C:\store management\Database\store.db")

cur=conn.cursor()

try:

if self.var\_supid.get()=="": #or self.var\_name.get()=="" or self.var\_email.get()=="" or self.var\_gender.get()=="" or self.var\_contact.get()=="" or self.var\_dob.get()=="" or self.var\_doj.get()=="" or self.var\_pass.get()=="" or self.var\_utype.get()=="" or self.var\_desc.get()=="" or self.var\_salary.get()=="":

messagebox.showerror("Error","All Fields must be required",parent=self.root)

else:

cur.execute("Select \* from supplier where supid=?",(self.var\_supid.get(),))

row=cur.fetchone()

if row!=None:

messagebox.showerror("Error","This Supplier ID is Alreary Assigned, try different",parent=self.root)

else:

cur.execute("Insert into supplier (supid,company,name,contact) values(?,?,?,?)",(

# self.var\_date.get(),

self.var\_supid.get(),

self.var\_company.get(),

self.var\_name.get(),

self.var\_contact.get(),

))

conn.commit()

messagebox.showinfo("Succes","Supplier Added Successfully",parent=self.root)

self.show()

except Exception as ex:

messagebox.showerror("Error",f"Error due to : {str(ex)}",parent=self.root)

def show(self):

conn=sqlite3.connect("C:\store management\Database\store.db")

cur=conn.cursor()

try:

cur.execute("Select \* from supplier")

rows=cur.fetchall()

self.SupplierTable.delete(\*self.SupplierTable.get\_children())

for row in rows:

self.SupplierTable.insert('',END,values=row)

except Exception as ex:

messagebox.showerror("Error",f"Error due to : {str(ex)}",parent=self.root)

def get\_data(self,ev):

f=self.SupplierTable.focus()

content=(self.SupplierTable.item(f))

row=content['values']

#print(row)

# self.var\_date.set(row[0])

self.var\_supid.set(row[0])

self.var\_company.set(row[1])

self.var\_name.set(row[2])

self.var\_contact.set(row[3])

def update(self):

conn=sqlite3.connect("C:\store management\Database\store.db")

cur=conn.cursor()

try:

if self.var\_supid.get()=="": #or self.var\_name.get()=="" or self.var\_email.get()=="" or self.var\_gender.get()=="" or self.var\_contact.get()=="" or self.var\_dob.get()=="" or self.var\_doj.get()=="" or self.var\_pass.get()=="" or self.var\_utype.get()=="" or self.var\_desc.get()=="" or self.var\_salary.get()=="":

messagebox.showerror("Error","All Fields must be required",parent=self.root)

else:

cur.execute("Select \* from supplier where supid=?",(self.var\_supid.get(),))

row=cur.fetchone()

if row==None:

messagebox.showerror("Error","Invalid Supplier ID",parent=self.root)

else:

cur.execute("Update supplier set company=?,name=?,contact=? where supid=?",(

# self.var\_date.get(),

self.var\_company.get(),

self.var\_name.get(),

self.var\_contact.get(),

self.var\_supid.get()

))

conn.commit()

messagebox.showinfo("Succes","Supplier Updated Successfully",parent=self.root)

self.show()

except Exception as ex:

messagebox.showerror("Error",f"Error due to : {str(ex)}",parent=self.root)

def delete(self):

conn=sqlite3.connect("C:\store management\Database\store.db")

cur=conn.cursor()

try:

if self.var\_supid.get()=="": #or self.var\_name.get()=="" or self.var\_email.get()=="" or self.var\_gender.get()=="" or self.var\_contact.get()=="" or self.var\_dob.get()=="" or self.var\_doj.get()=="" or self.var\_pass.get()=="" or self.var\_utype.get()=="" or self.var\_desc.get()=="" or self.var\_salary.get()=="":

messagebox.showerror("Error","All Fields must be required",parent=self.root)

else:

cur.execute("Select \* from supplier where supid=?",(self.var\_supid.get(),))

row=cur.fetchone()

if row==None:

messagebox.showerror("Error","Invalid Supplier ID",parent=self.root)

else:

op=messagebox.askyesno("Confirm","Do you really want to delete?",parent=self.root)

if op==True:

cur.execute("delete from supplier where supid=?",(self.var\_supid.get(),))

conn.commit()

messagebox.showinfo("Delete","Supplier Deleted Successfully",parent=self.root)

self.clear()

except Exception as ex:

messagebox.showerror("Error",f"Error due to : {str(ex)}",parent=self.root)

def clear(self):

# self.var\_date.set("")

self.var\_supid.set("")

self.var\_company.set("")

self.var\_name.set("")

self.var\_contact.set("")

self.var\_searchtxt.set("")

self.show()

def search(self):

conn=sqlite3.connect("C:\store management\Database\store.db")

cur=conn.cursor()

try:

if self.var\_searchtxt.get()=="":

messagebox.showerror("Error","Supplier ID is Required",parent=self.root)

else:

cur.execute("Select \* from supplier where supid=?",(self.var\_searchtxt.get(),))

row=cur.fetchone()

if row!=None:

self.SupplierTable.delete(\*self.SupplierTable.get\_children())

self.SupplierTable.insert('',END,values=row)

else:

messagebox.showerror("Error","No Record Found!!!",parent=self.root)

except Exception as ex:

messagebox.showerror("Error",f"Error due to : {str(ex)}",parent=self.root)

if \_\_name\_\_=="\_\_main\_\_":

root=Tk()

#root.resizable(False,False)

obj=supplierClass(root)

root.mainloop()

**7. Category Page:**

from tkinter import \*

from PIL import Image,ImageTk #pip install pillow

from tkinter import ttk,messagebox

import sqlite3

class categoryClass:

def \_\_init\_\_(self,root):

self.root=root

self.root.geometry("1100x550+220+130")

self.root.title("Inventory Management System")

self.root.config(bg="white")

self.root.focus\_force()

#=============================

#####Variables=====

self.var\_cat\_id=StringVar()

self.var\_main\_name=StringVar()

self.var\_sub\_name=StringVar()

#======Title======

lbl\_title=Label(self.root,text="Manage Product Category",font=("goudy old style",30),bg="#184a45",fg="white",bd=3,relief=RIDGE).pack(side=TOP,fill=X,padx=10,pady=20)

#======Entry Frame=====

entry\_frame=Frame(self.root,bd=3,relief=RIDGE,bg="white")

entry\_frame.place(x=30,y=80,width=510,height=450)

lbl\_cid=Label(entry\_frame,text=" Category ID",font=("goudy old style",20),bg="white").place(x=10,y=10)

lbl\_m\_name=Label(entry\_frame,text="Main Category Name",font=("goudy old style",20),bg="white").place(x=10,y=50)

# lbl\_s\_name=Label(entry\_frame,text="Sub Category Name",font=("goudy old style",20),bg="white").place(x=10,y=100)

lbl\_cid=Entry(entry\_frame,textvariable=self.var\_cat\_id,font=("goudy old style",20),bg="lightyellow").place(x=250,y=10,width=150) #cmb\_m\_name=ttk.Combobox(entry\_frame,textvariable=self.var\_main\_name,values=("Select","Admin","Employee"),state='readonly',justify=CENTER,font=("goudy old style",20))

#cmb\_m\_name.place(x=250,y=50,width=200)

#cmb\_m\_name.current(0) #cmb\_s\_name=ttk.Combobox(entry\_frame,textvariable=self.var\_sub\_name,values=("Select","Admin",input()),state='readonly',justify=CENTER,font=("goudy old style",20))

#cmb\_s\_name.place(x=250,y=100,width=200)

#cmb\_s\_name.current(0)

lbl\_m\_name=Entry(entry\_frame,textvariable=self.var\_main\_name,font=("goudy old style",20),bg="lightyellow").place(x=250,y=50,width=150)

# lbl\_s\_name=Entry(entry\_frame,textvariable=self.var\_sub\_name,font=("goudy old style",20),bg="lightyellow").place(x=250,y=100,width=150)

btn\_add=Button(entry\_frame,text="Add",command=self.add,font=("goudy old style",15),bg="#4caf50",fg="white",cursor="hand2").place(x=250,y=150,width=250,height=50)

btn\_delete=Button(entry\_frame,text="Delete",command=self.delete,font=("goudy old style",15),bg="#ff0000",fg="white",cursor="hand2").place(x=250,y=220,width=150,height=30)

# btn\_cat\_apmc=Button(entry\_frame,text="APMC Category",command=self.cat\_apmc,font=("goudy old style",15,"bold"),bg="white",cursor="hand2").place(x=10,y=150,width=200,height=40)

#====Category Details===

cat\_frame=Frame(self.root,bd=3,relief=RIDGE)

cat\_frame.place(x=570,y=80,width=490,height=450)

scrolly=Scrollbar(cat\_frame,orient=VERTICAL)

scrollx=Scrollbar(cat\_frame,orient=HORIZONTAL)

self.CategoryTable=ttk.Treeview(cat\_frame,columns=("cid","m\_name"))

scrollx.pack(side=BOTTOM,fill=X)

scrolly.pack(side=RIGHT,fill=Y)

scrollx.config(command=self.CategoryTable.xview)

scrolly.config(command=self.CategoryTable.yview)

self.CategoryTable.heading("cid",text="Category ID")

self.CategoryTable.heading("m\_name",text="Main Category")

# self.CategoryTable.heading("s\_name",text="Sub Category")

self.CategoryTable["show"]="headings"

self.CategoryTable.column("cid",width=80)

self.CategoryTable.column("m\_name",width=100)

# self.CategoryTable.column("s\_name",width=100)

self.CategoryTable.pack(fill=BOTH,expand=1)

self.CategoryTable.bind("<ButtonRelease-1>",self.get\_data)

self.show()

#===images======

#self.im1=Image.open("images/cat.jpg")

#self.im1=self.im1.resize((500,220),Image.ANTIALIAS)

#self.im1=ImageTk.PhotoImage(self.im1)

#self.lbl\_im1=Label(self.root,image=self.im1,bd=2,relief=RAISED)

#self.lbl\_im1.place(x=40,y=270)

#self.im2=Image.open("images/category.jpg")

#self.im2=self.im2.resize((500,220),Image.ANTIALIAS)

#self.im2=ImageTk.PhotoImage(self.im2)

#self.lbl\_im2=Label(self.root,image=self.im2,bd=2,relief=RAISED)

#self.lbl\_im2.place(x=580,y=270)

#=======Functions===========

def cat\_apmc(self):

self.new\_win=Toplevel(self.root)

# self.new\_obj=cat\_apmcClass(self.new\_win)

def add(self):

conn = sqlite3.connect("C:\store management\Database\store.db")

cur=conn.cursor()

try:

if self.var\_cat\_id.get()=="": #or self.var\_name.get()=="" or self.var\_email.get()=="" or self.var\_gender.get()=="" or self.var\_contact.get()=="" or self.var\_dob.get()=="" or self.var\_doj.get()=="" or self.var\_pass.get()=="" or self.var\_utype.get()=="" or self.var\_desc.get()=="" or self.var\_salary.get()=="":

messagebox.showerror("Error","All Fields must be required",parent=self.root)

else:

cur.execute("Select \* from category where cid=?",(self.var\_cat\_id.get(),))

row=cur.fetchone()

if row!=None:

messagebox.showerror("Error","This Category ID is Alreary Assigned, try different",parent=self.root)

else:

cur.execute("Insert into category(cid,m\_name) values(?,?)",(

self.var\_cat\_id.get(),

self.var\_main\_name.get()

#self.var\_sub\_name.get()

))

conn.commit()

messagebox.showinfo("Succes","Category Added Successfully",parent=self.root)

self.show()

except Exception as ex:

messagebox.showerror("Error",f"Error due to : {str(ex)}",parent=self.root)

def show(self):

conn = sqlite3.connect("C:\store management\Database\store.db")

cur=conn.cursor()

try:

cur.execute("Select \* from category")

rows=cur.fetchall()

self.CategoryTable.delete(\*self.CategoryTable.get\_children())

for row in rows:

self.CategoryTable.insert('',END,values=row)

except Exception as ex:

messagebox.showerror("Error",f"Error due to : {str(ex)}",parent=self.root)

def get\_data(self,ev):

f=self.CategoryTable.focus()

content=(self.CategoryTable.item(f))

row=content['values']

#print(row)

self.var\_cat\_id.set(row[0])

self.var\_main\_name.set(row[1])

def delete(self):

conn = sqlite3.connect("C:\store management\Database\store.db")

cur=conn.cursor()

try:

if self.var\_cat\_id.get()=="": #or self.var\_name.get()=="" or self.var\_email.get()=="" or self.var\_gender.get()=="" or self.var\_contact.get()=="" or self.var\_dob.get()=="" or self.var\_doj.get()=="" or self.var\_pass.get()=="" or self.var\_utype.get()=="" or self.var\_desc.get()=="" or self.var\_salary.get()=="":

messagebox.showerror("Error","please select category from the list",parent=self.root)

else:

cur.execute("Select \* from category where cid=?",(self.var\_cat\_id.get(),))

row=cur.fetchone()

if row==None:

messagebox.showerror("Error","Error, please try again",parent=self.root)

else:

op=messagebox.askyesno("Confirm","Do you really want to delete?",parent=self.root)

if op==True:

cur.execute("delete from category where cid=?",(self.var\_cat\_id.get(),))

conn.commit()

messagebox.showinfo("Delete","Category Deleted Successfully",parent=self.root)

self.show()

self.var\_cat\_id.set("")

self.var\_main\_name.set("")

#self.var\_sub\_name.set("")

except Exception as ex:

messagebox.showerror("Error",f"Error due to : {str(ex)}",parent=self.root)

if \_\_name\_\_=="\_\_main\_\_":

root=Tk()

#root.resizable(False,False)

obj=categoryClass(root)

root.mainloop()

**6.4 Testing Approach**

We used different testing approach to test our application like unit testing. Usability testing and Security testing. Testing the Rexine Store Management System (RSMS) is a critical phase to ensure its functionality, reliability, and security. A well-planned testing approach helps identify issues, validate requirements, and ensure the system works as intended.

**6.4.1 Unit Testing**

Unit Testing is a type of software testing where individual units or components of a software are tested. The purpose is to validate that each unit of the software code performs as expected. Unit Testing is done during the development (coding phase) of an application to Reduces the Defects in the Newly developed features or reduces bugs when changing the existing functionality.

• Reduces Cost of Testing as defects are captured in very early phase.

• Improves design and allows better refactoring of code.

• Unit Tests, when integrated with build gives the quality of the build as well.

• Objective: Test individual components, modules, or functions in isolation.

• Focus Areas: Business logic, database interactions, user interface elements.

•Tools: Unit testing frameworks for the programming language used (e.g., unittest in Python).

**6.4.2 Usability Testing**

The Rexine Store Management System (RSMS) is essential to ensure that the system is user-friendly, intuitive, and efficient for the store employees who will be using it regularly.In this testing I ensure that there no spelling or grammatical errors. I also ensure that no dark coloured themed is used, so that the layout UI (User interface) Components are visible to user clearly. I also performed the navigation testing. I checked all the buttons, menu and links that they work properly on the click event and redirect to the correct page. I also ensure that all the options on header, footer left/right navigation are consistent throughout the applications

. **6.4.3 Security Testing**

Security testing attempts to verify that protection mechanism built into a system will protect it from improper penetration. It also aims at verifying below principal:

**1.Confidentiality:** I ensure that the data provided by the user during the registration is kept confidential and only the authorized person can access it.

**2.Integrity:** The term data integrity refers to the accuracy and consistency of data. I ensure that the user provide right type of data in the data field. If user enter the password in email id field it will show an error message. Data integrity will prevent user from making these mistakes.

**3.Authentication:** Authentication is the process of verifying the identity of a person. I ensure that the user enter the correct username and password while doing login.

**4.Authorization:** Authorization is the process of giving someone the ability to access a resource. The SQL Server database can only be access by our group members as it contains the user information and can access only by providing the valid credentials.

**6.4.4 Positive** **Testing**

We are checking whether our application behaves as expected with positive input. We are testing the application by giving the valid data. For example: In login module when provided the valid data it should redirect the user to home page.

**Positive Testing:**

Product Addition: Verify that new products can be added with all required details (name, price, quantity, etc.) and they appear correctly in the inventory.

**Inventory Update**: Test if the system accurately updates the inventory when products are sold or restocked.

**Sales Process:** Confirm that sales transactions are processed correctly, including calculating the total amount, applying discounts if applicable, and generating invoices.

**Customer Management:** Validate that customer information can be added, modified, and accessed as needed.

**Supplier Management**: Ensure that supplier details are properly stored and products can be ordered from suppliers.

**Reporting:** Verify the generation of various reports such as sales reports, inventory status, and profit/loss statements.

**User Authentication:** Confirm that only authorized users can access the system and perform necessary operations.

**Search Functionality:** Test the search functionality to ensure users can find products, customers, or sales transactions efficiently.

**Data Backup and Recovery:** Verify that the system allows for data backup and that data can be restored successfully if needed.

**6.4.5 Negative Testing**

We are checking whether our application behaves as expected with negative input. We are testing the application by giving the invalid data. For example: In login module when provided the invalid data it should display an error message. Goal is to check the stability of the application against incorrect data.

**Negative Testing:**

**Invalid Product Addition:** Test adding a product with missing or invalid information to check if the system handles errors appropriately.

**Out-of-Stock Sales**: Attempt to sell products that are out of stock and verify the system prevents such transactions.

**Incorrect Pricing:** Test if the system prevents the sale of products with incorrect or negative prices.

**Unauthorized Access:** Try accessing the system with invalid credentials to ensure unauthorized users are denied access.

**Concurrency Testing:** Simulate multiple users accessing and updating data simultaneously to check for issues related to concurrent access.

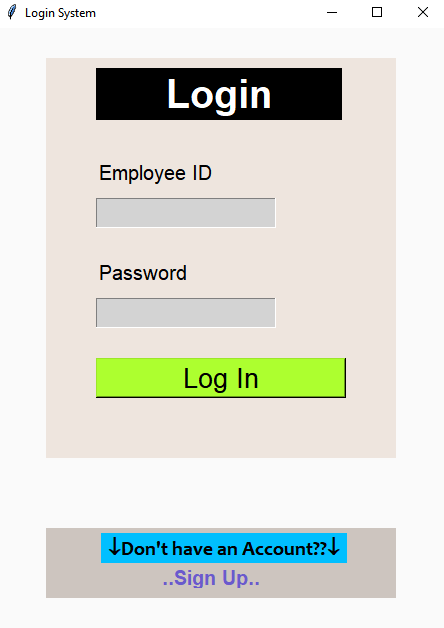
**Boundary Testing**: Test the system's behavior with extreme values, such as very large or very small quantities or prices.

**Network Failure:** Simulate network failures during transactions to ensure the system can handle such interruptions and recover gracefully.

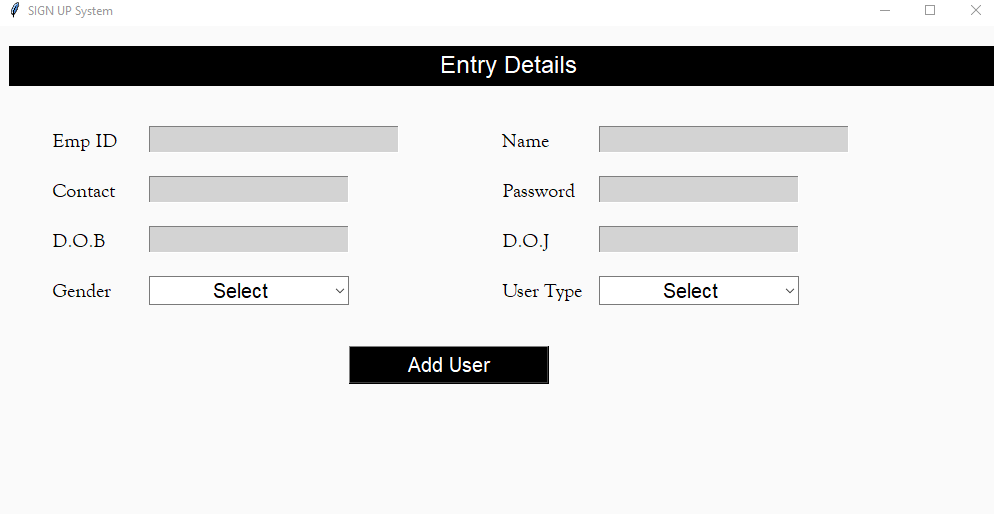
**Performance Testing**: Load the system with a large number of simultaneous requests to evaluate its performance under stress.

**Chapter 7. Results**

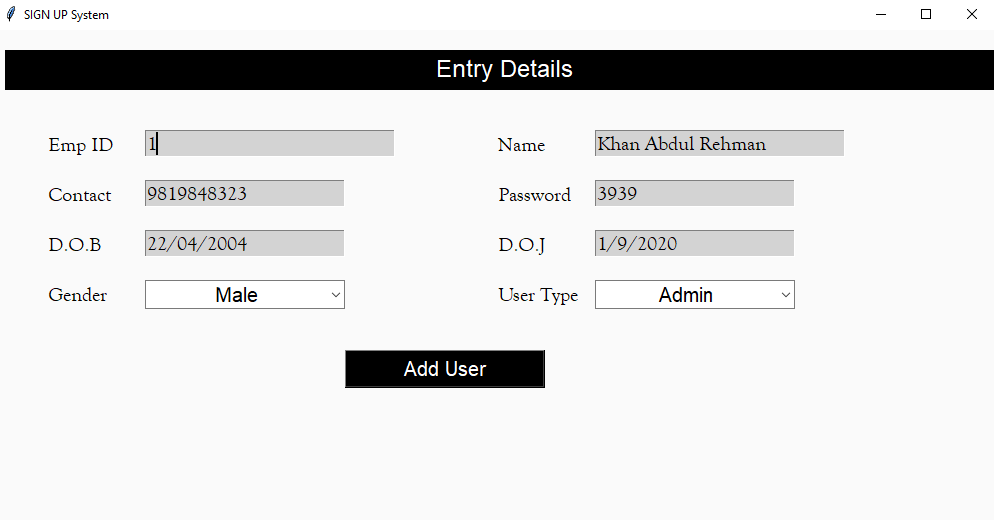
**LOGIN PAGE**

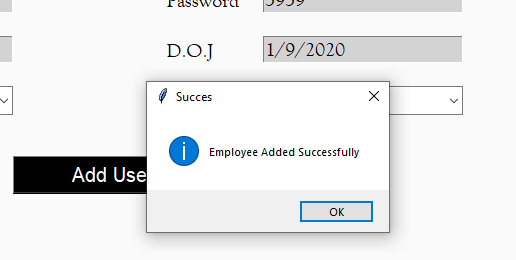
****

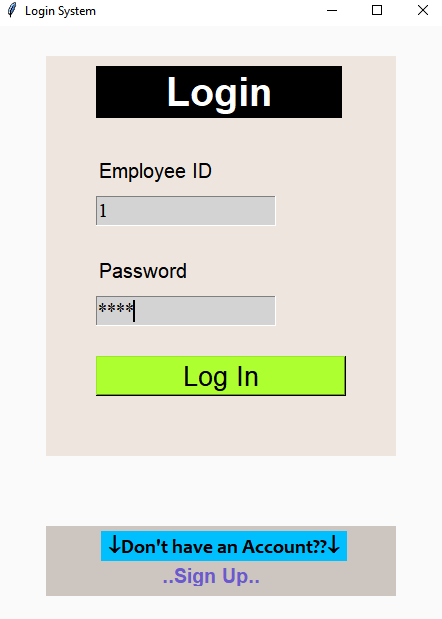
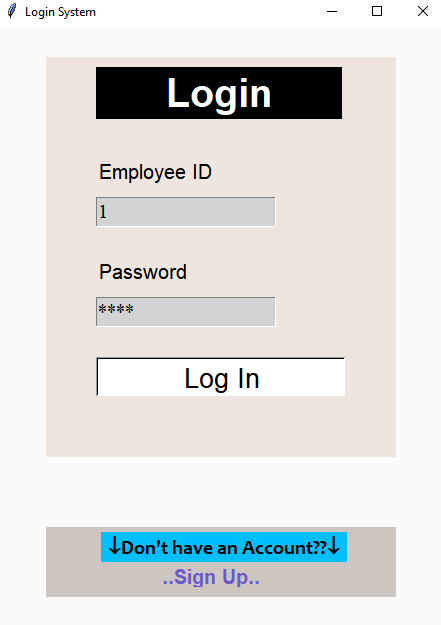
**SIGN UP PAGE**

****

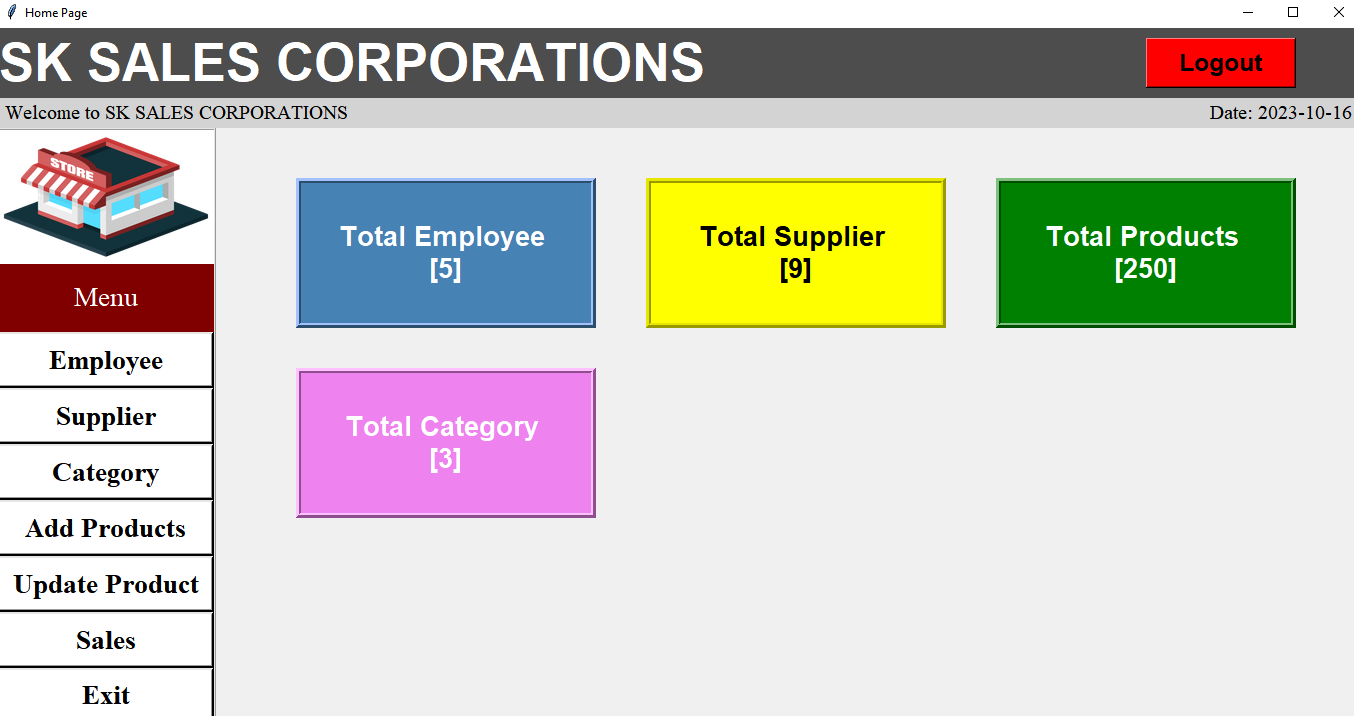
**ADMIN SIGN UP AND LOGIN**

****

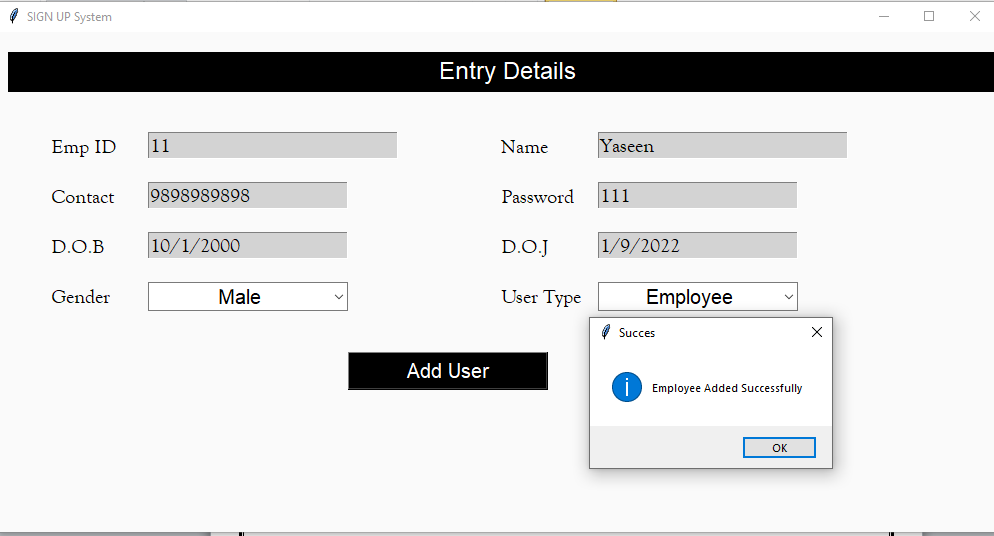
****

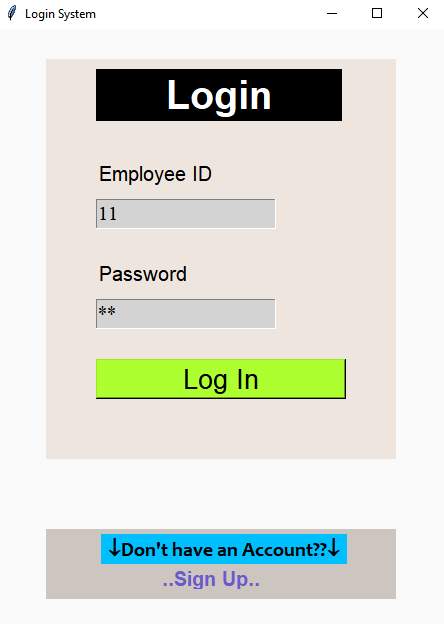
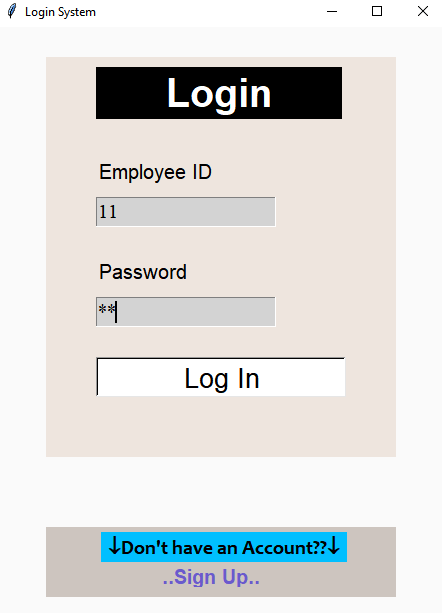
** **

**After Admin login:**

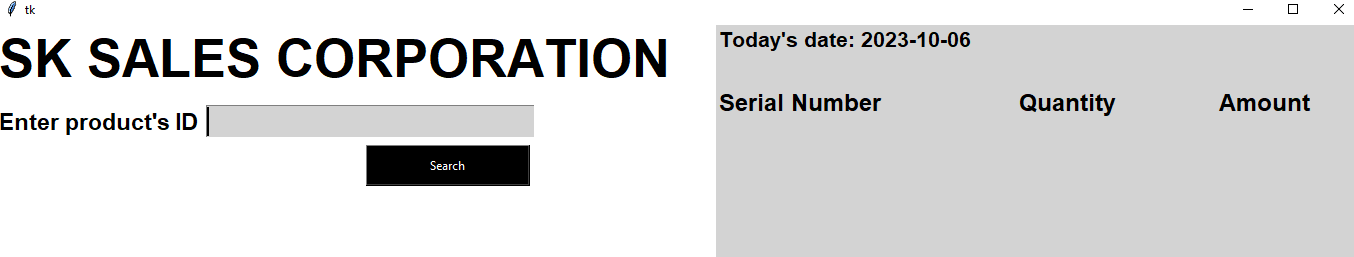


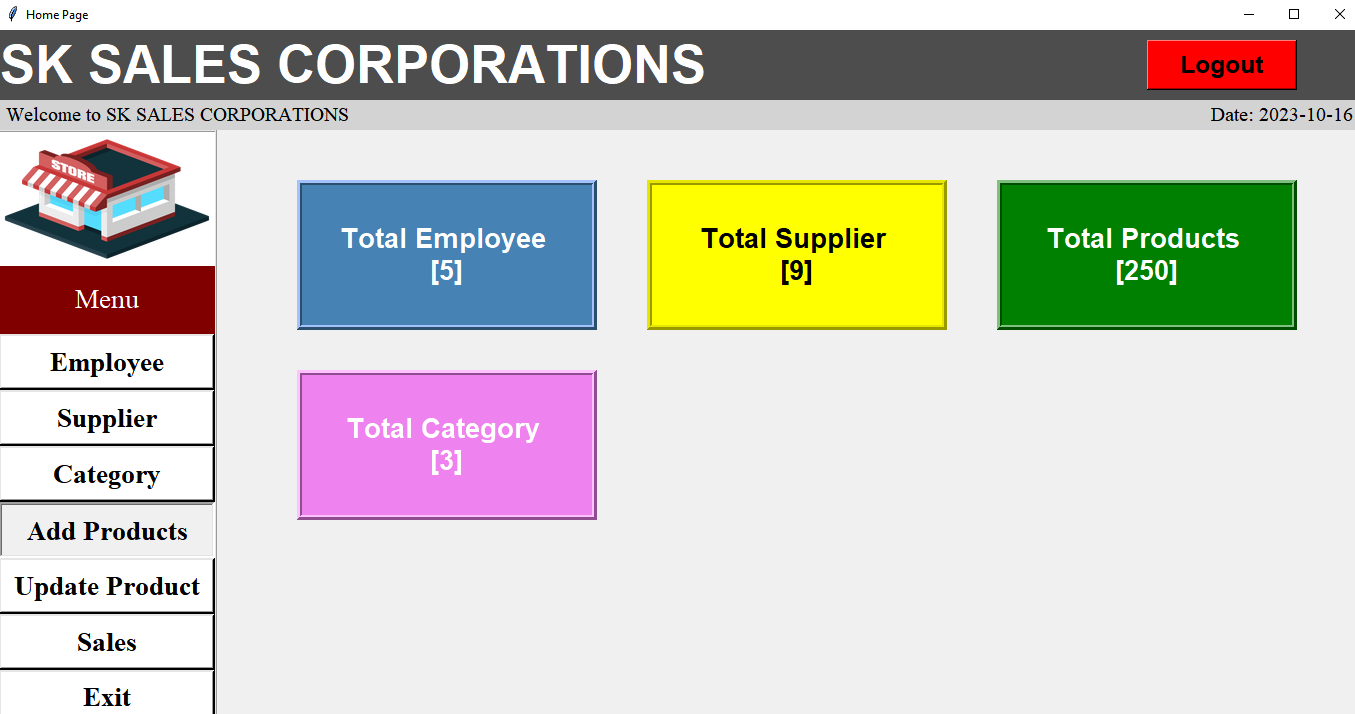
**EMPLOYEE SIGN UP AND LOGIN**

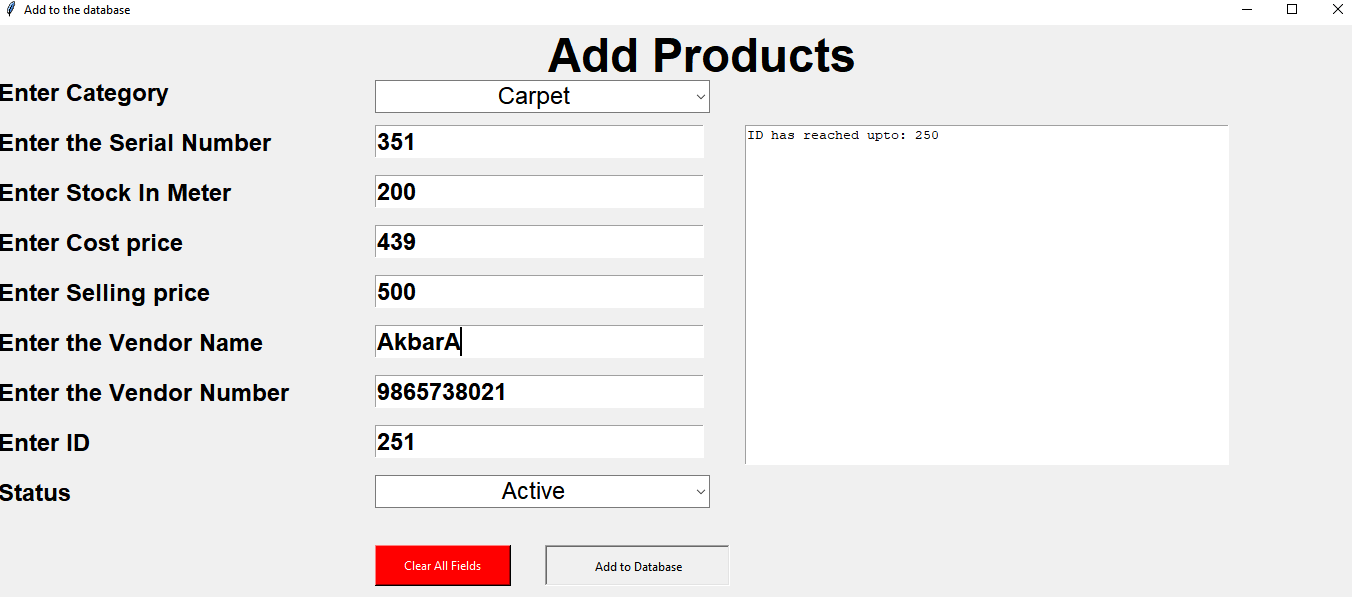
****

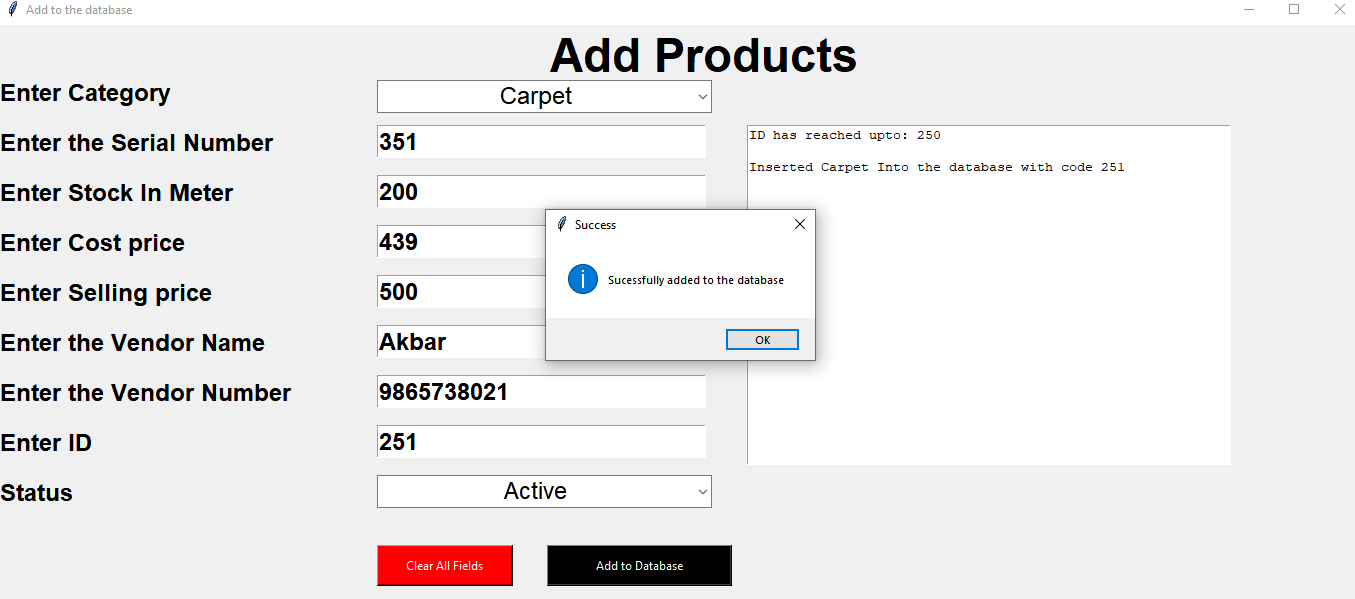
** **

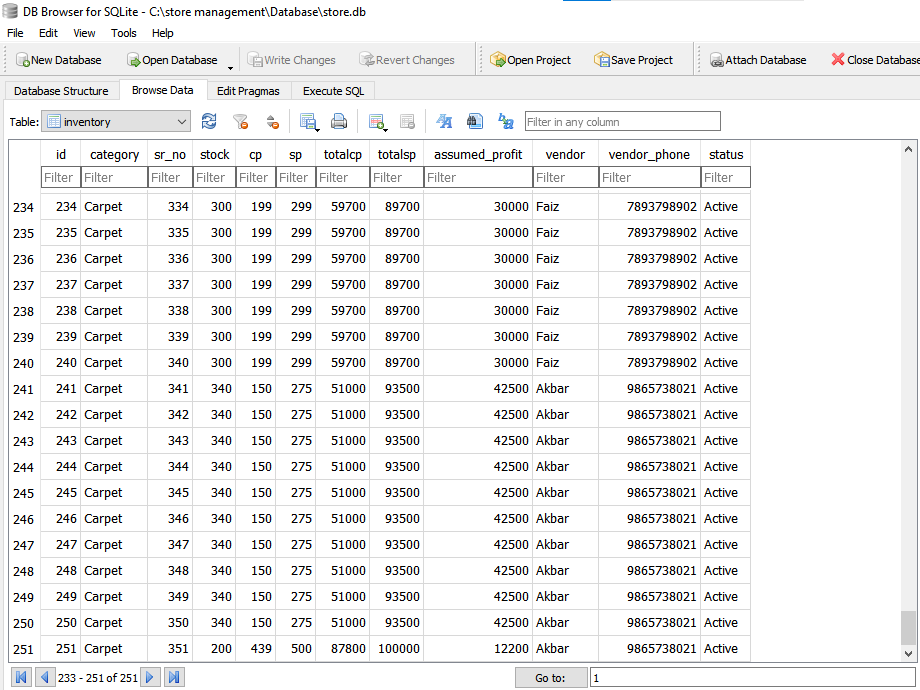
**After Employee Login:**

****

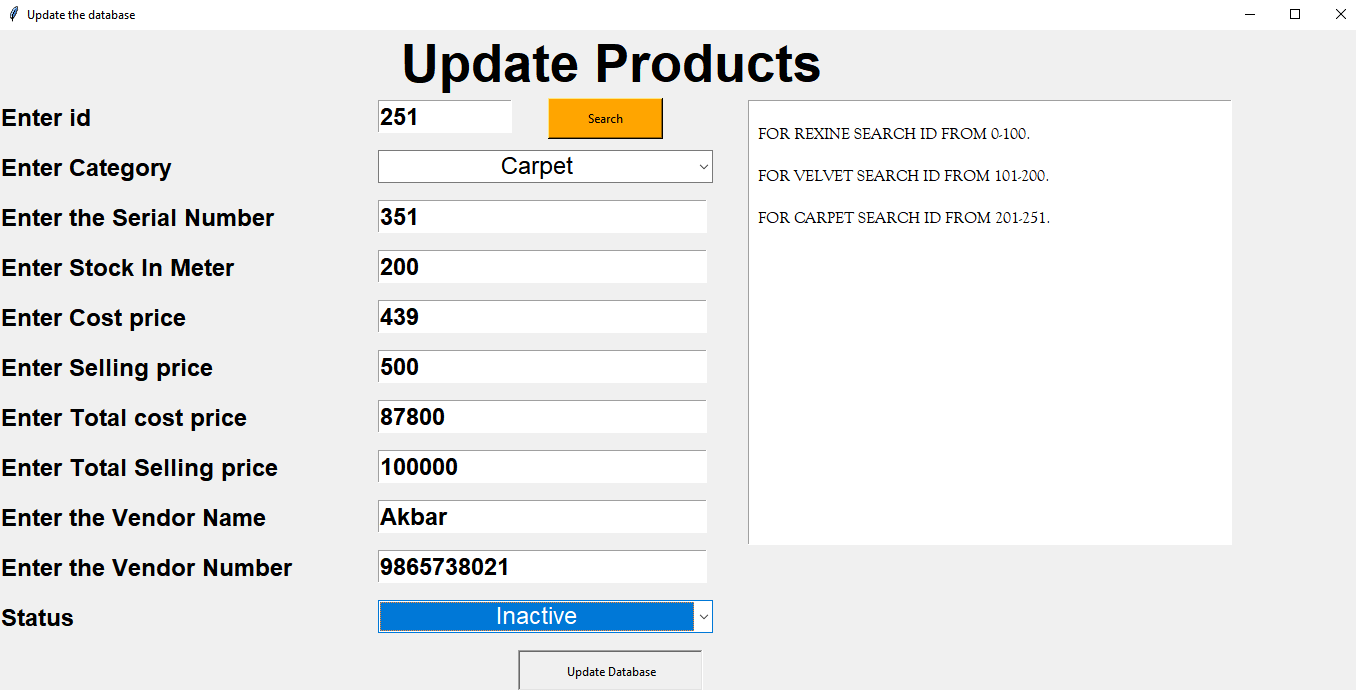
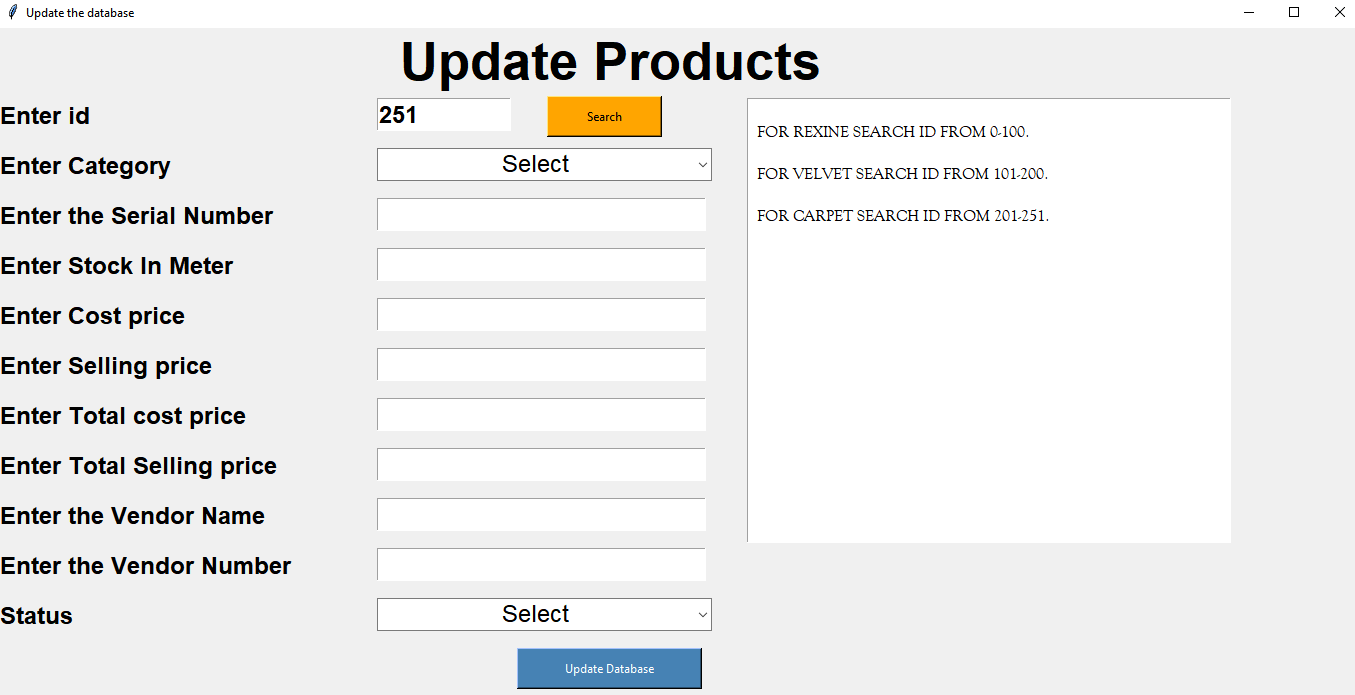
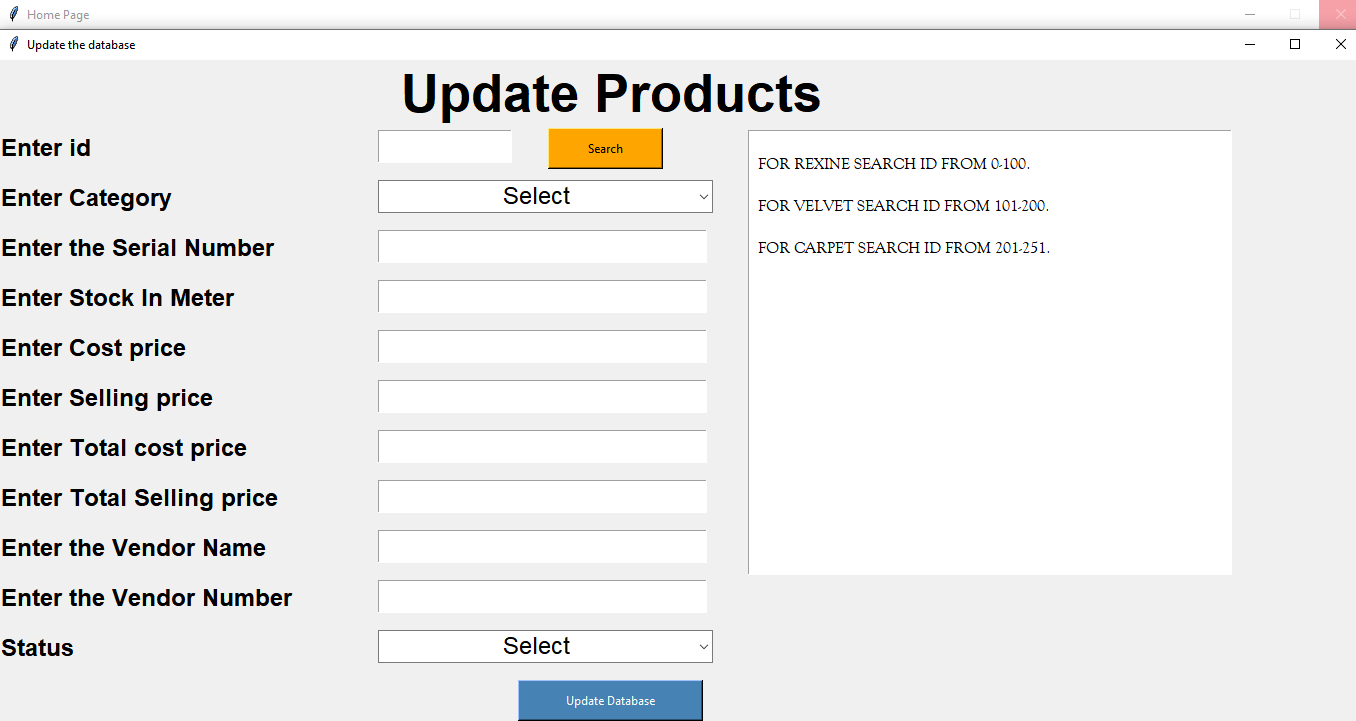
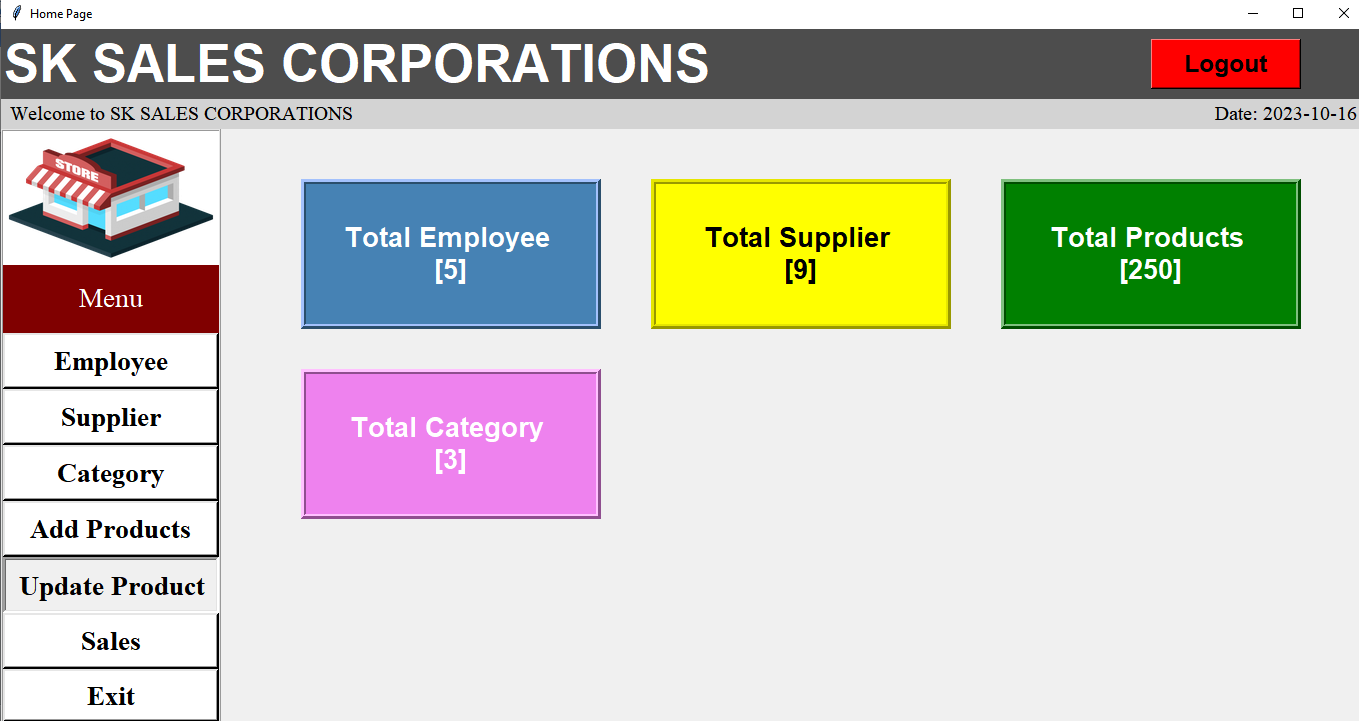
**Add to a Product to store**

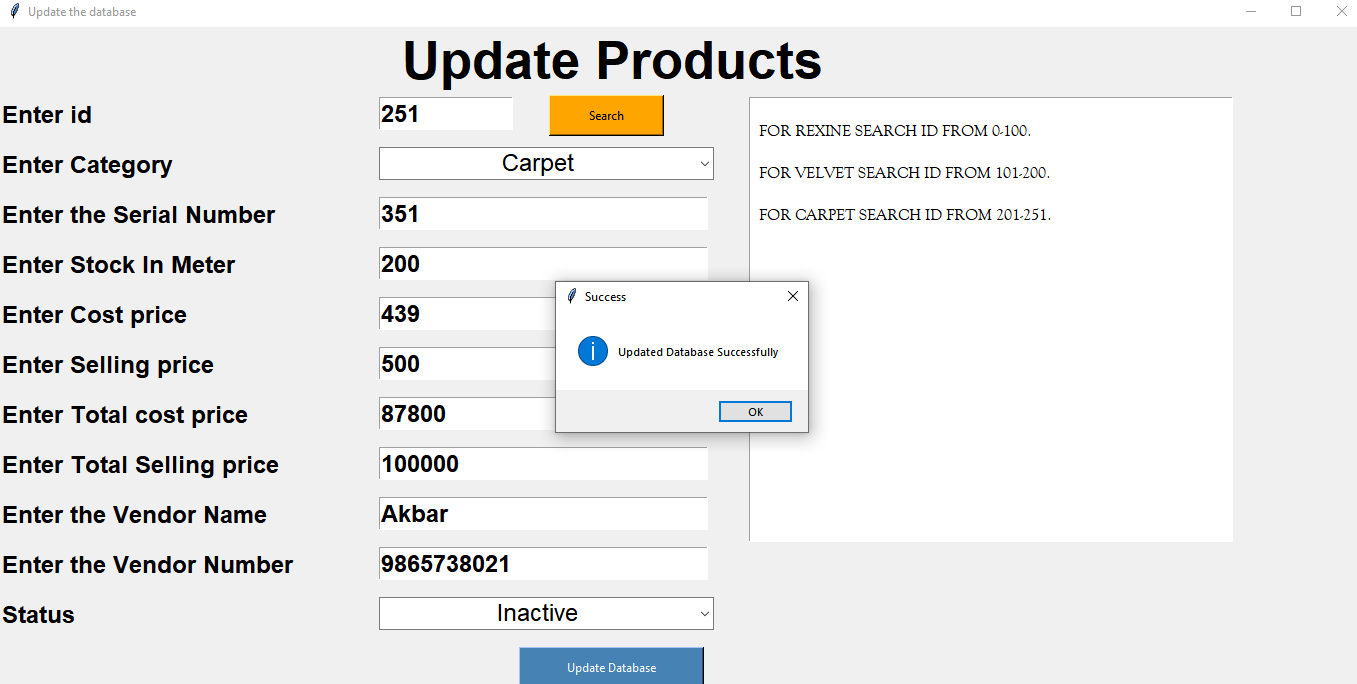




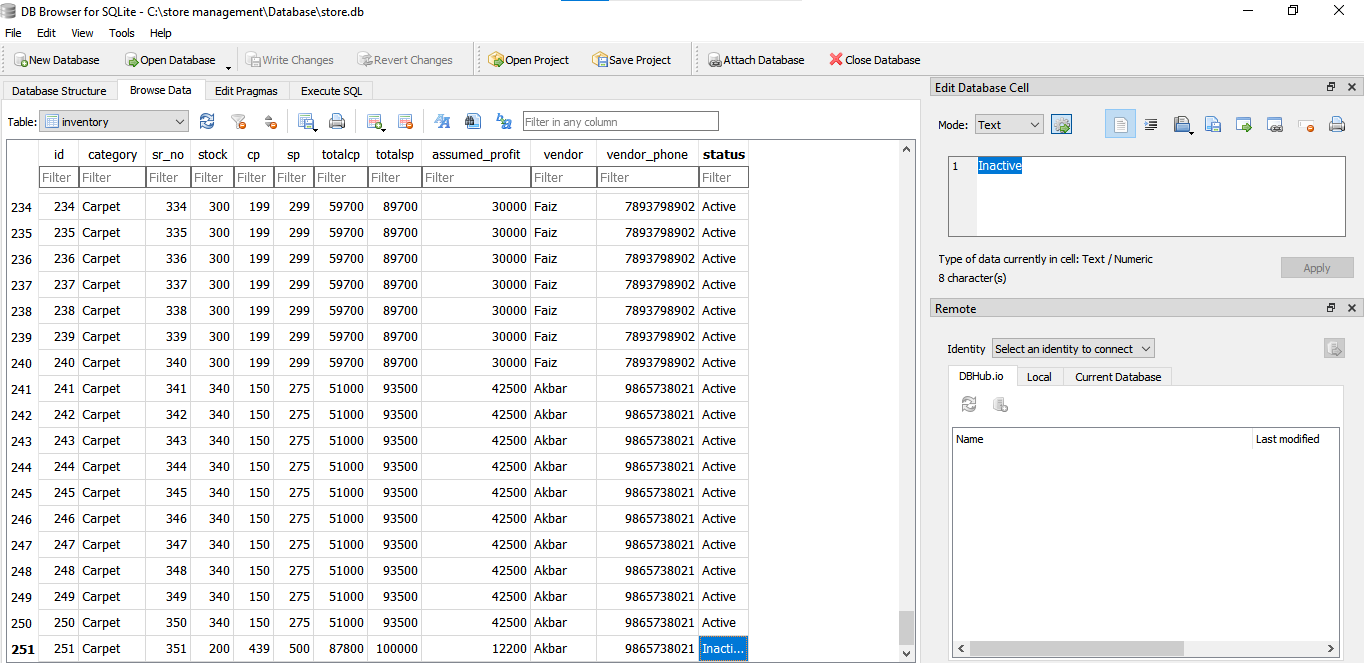


**UPDATE TO DATABASE**

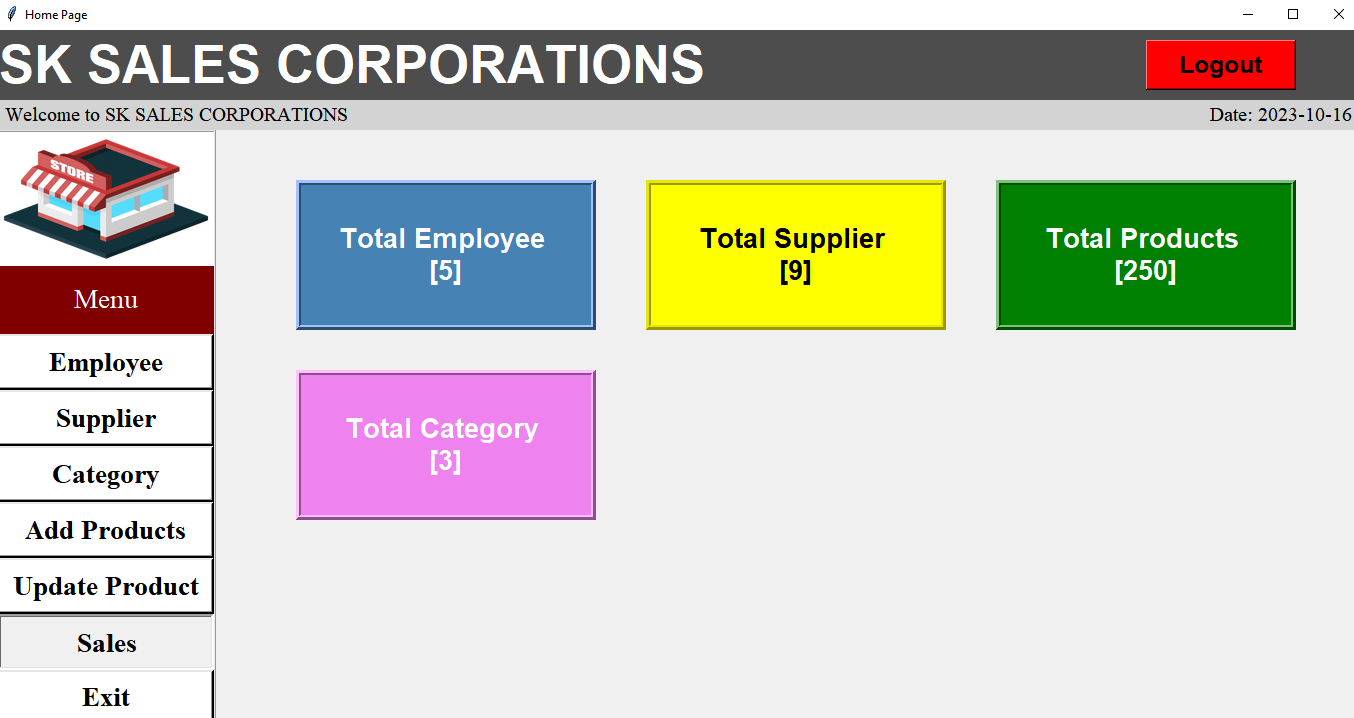
****

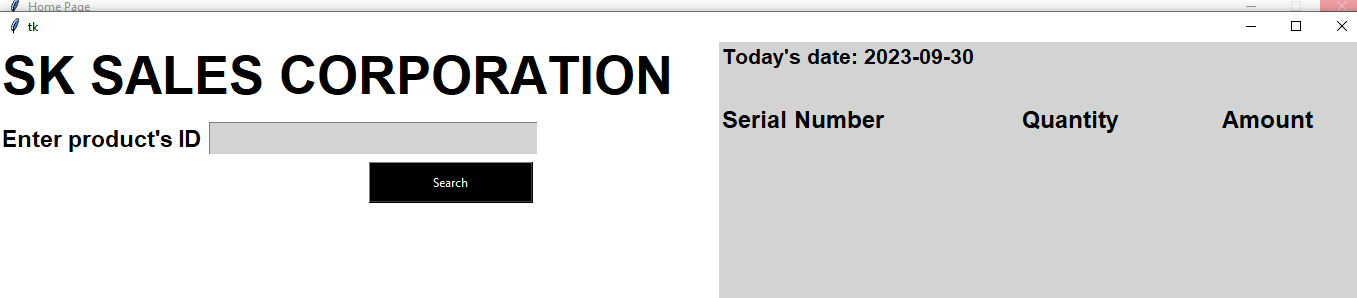
****

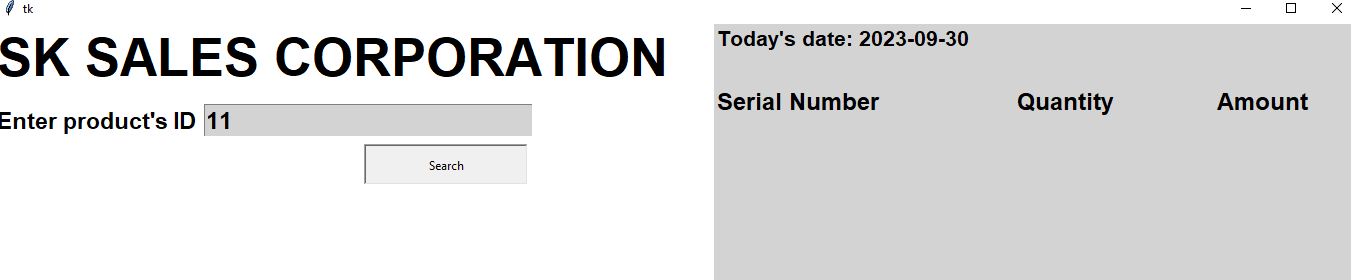
**Status changed of ID : 251. Active to Inactive**

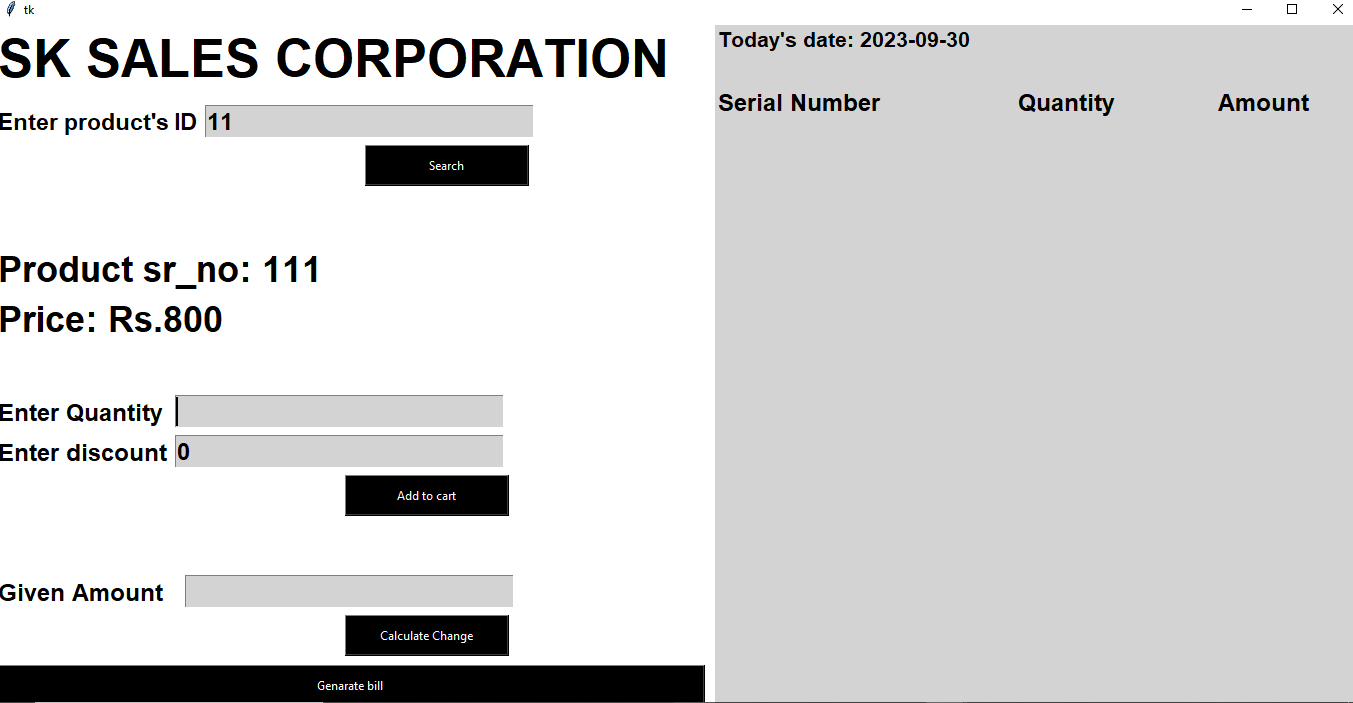
****

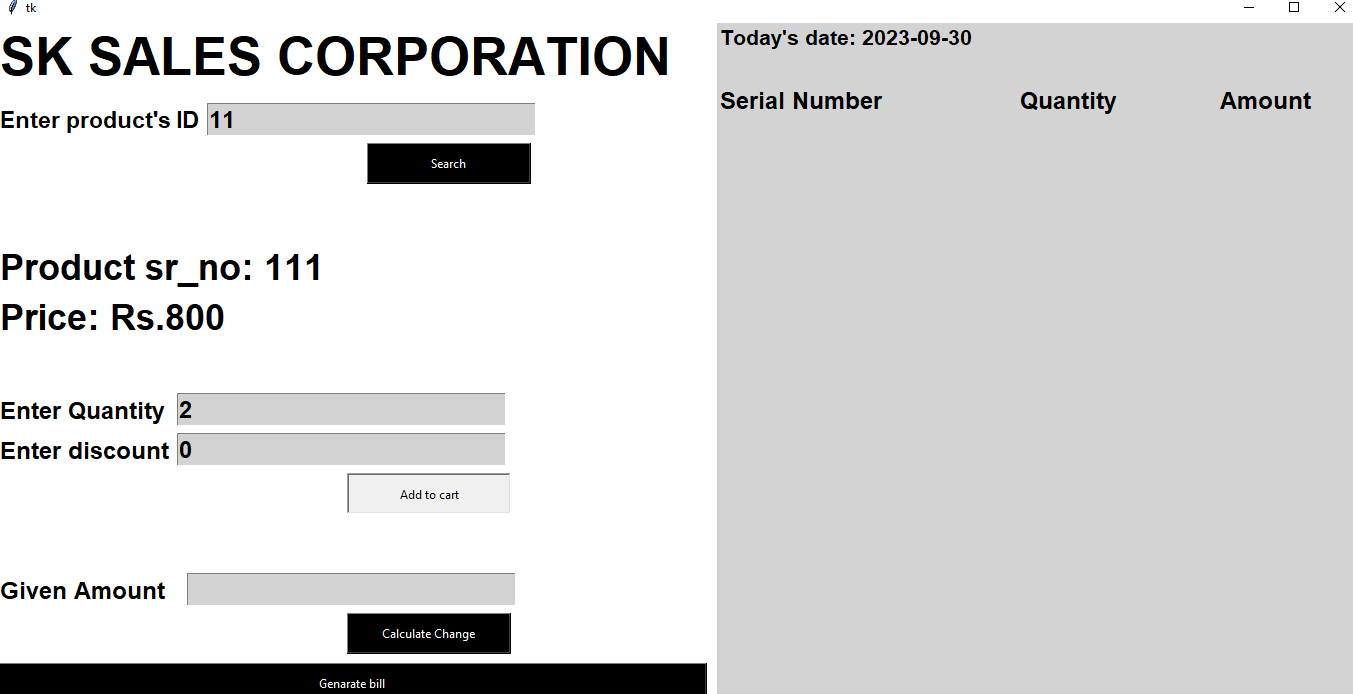
**Sales Page:**

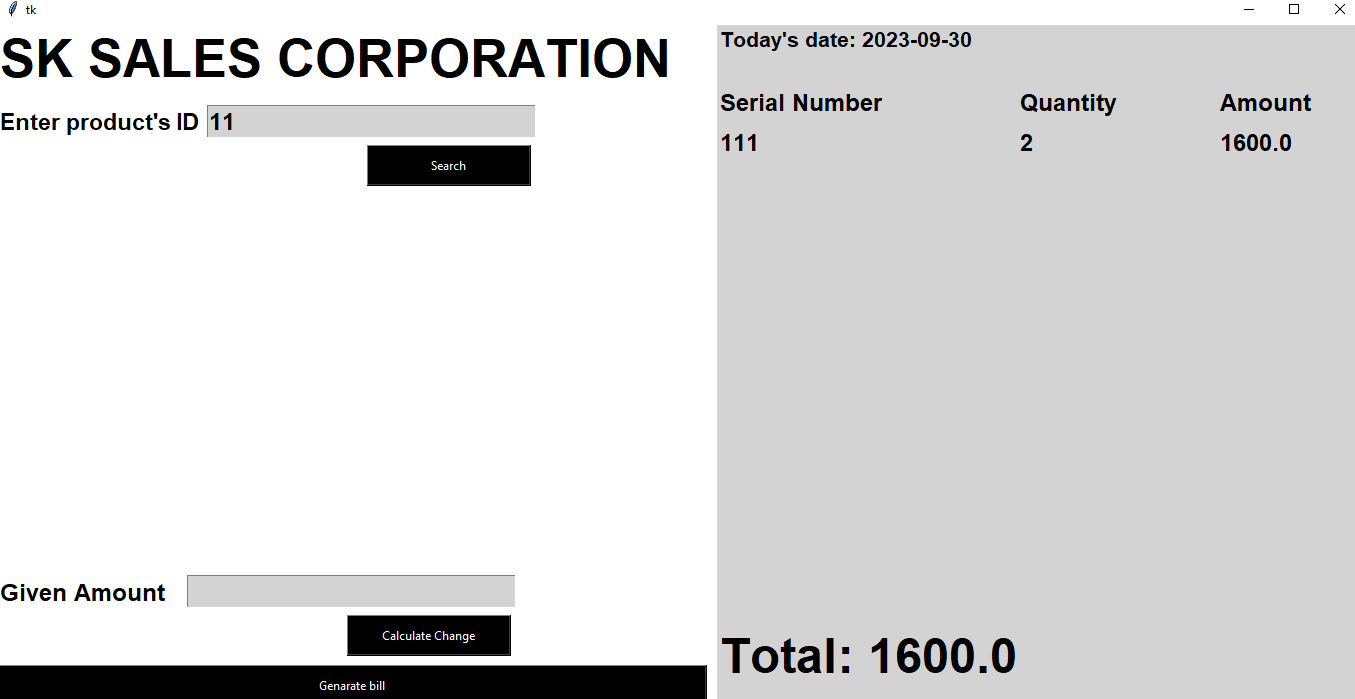


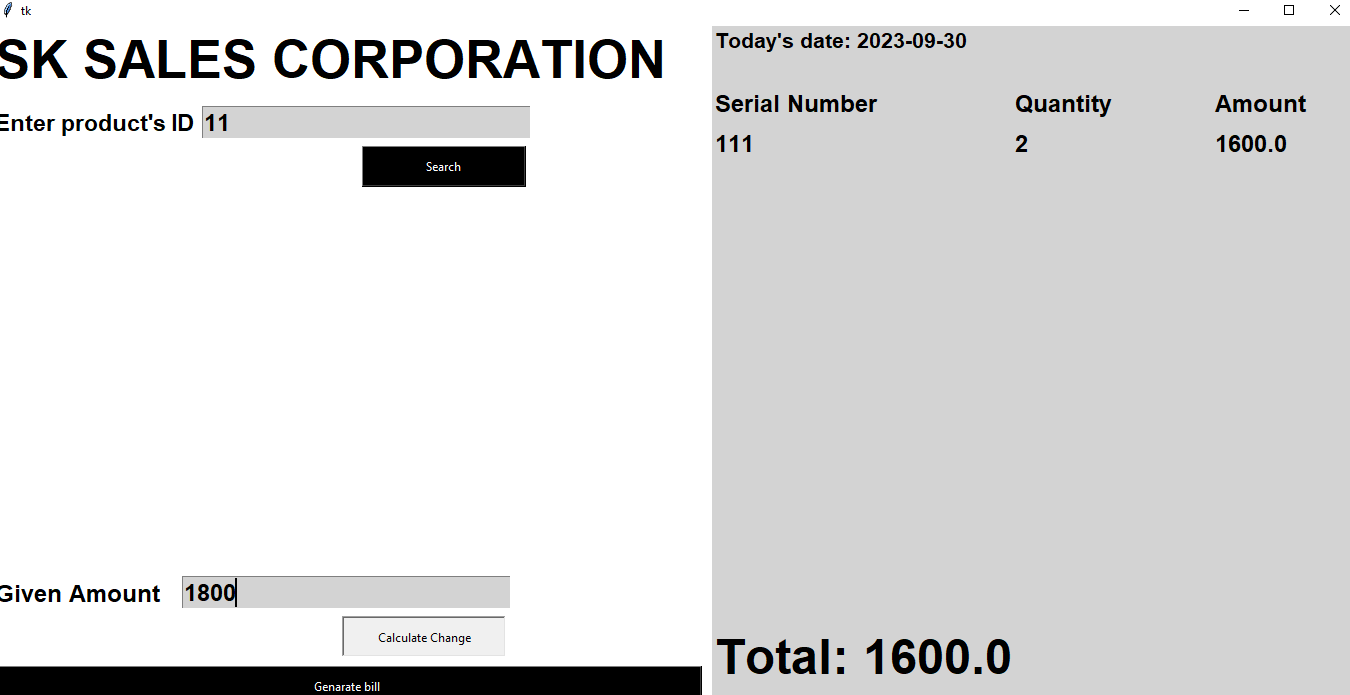


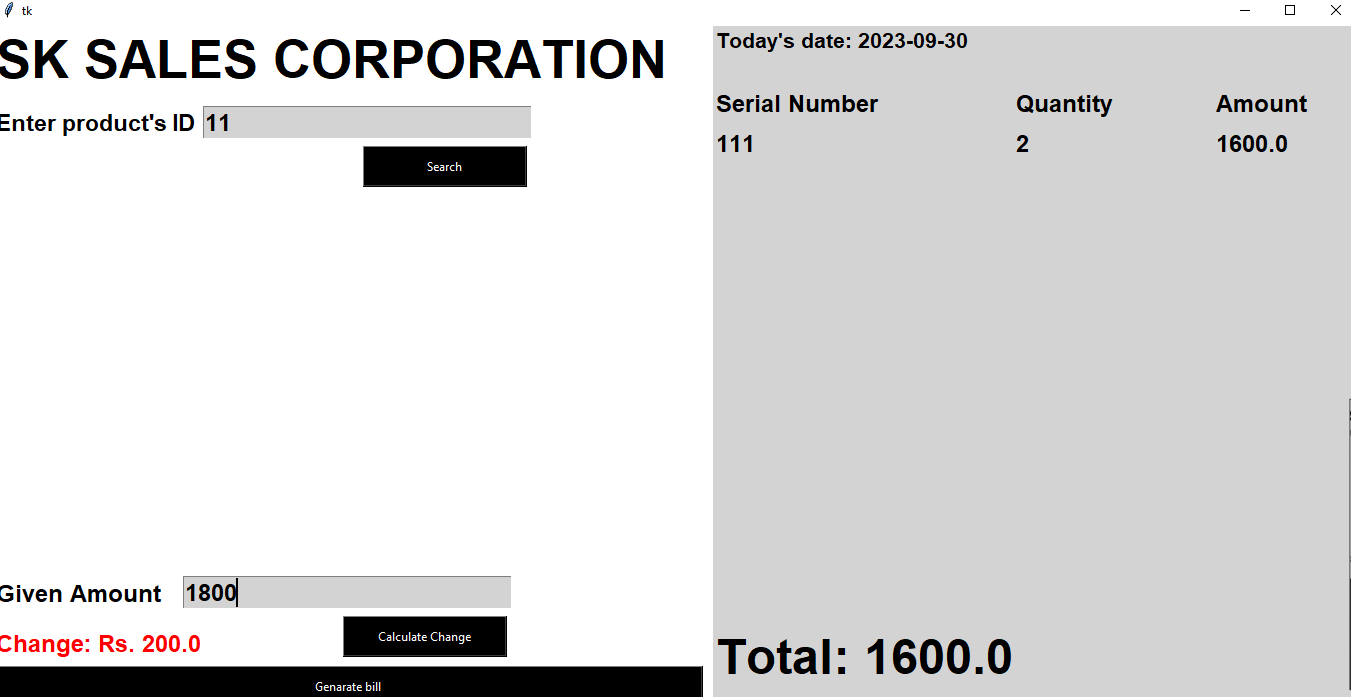


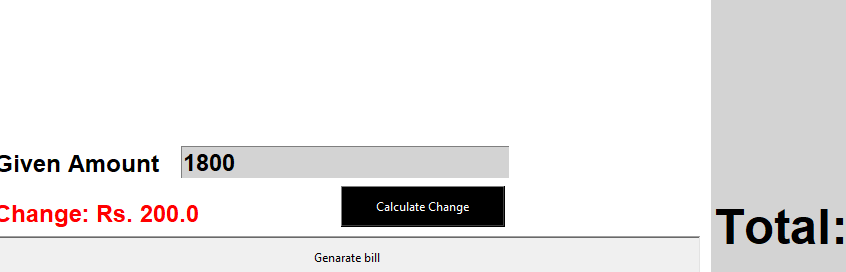


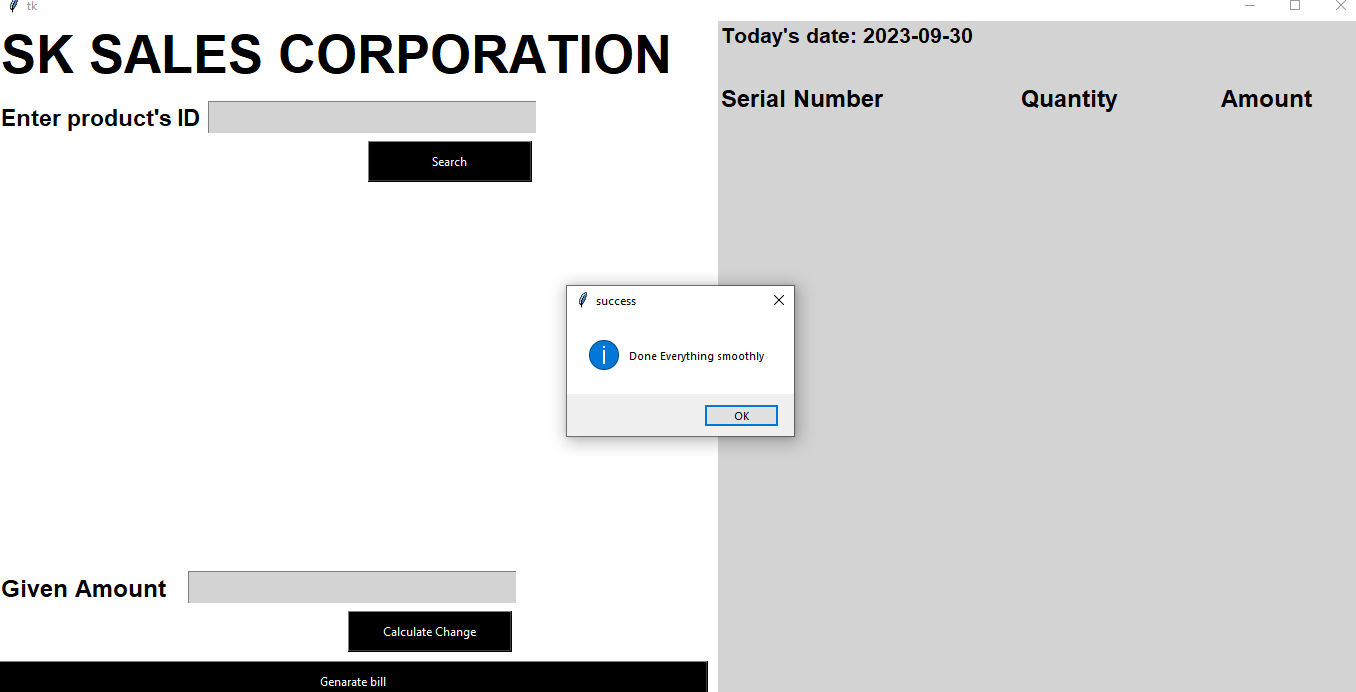


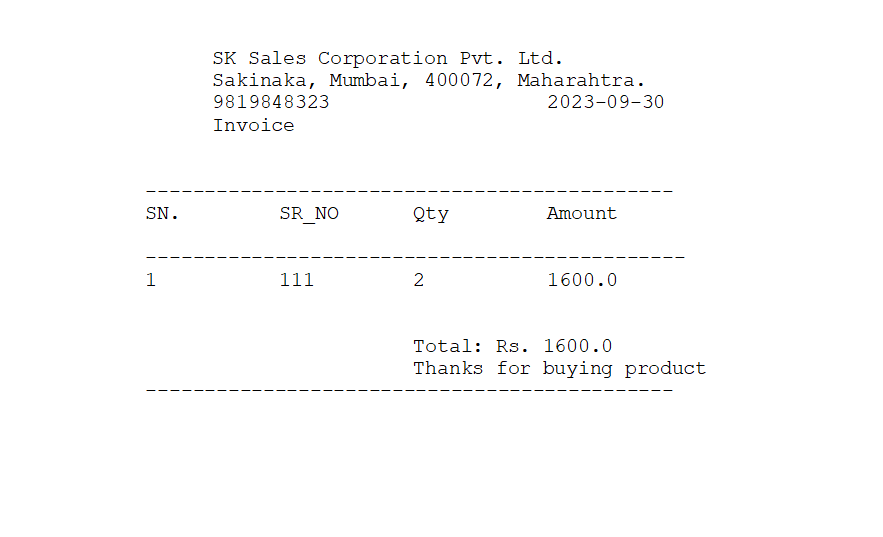






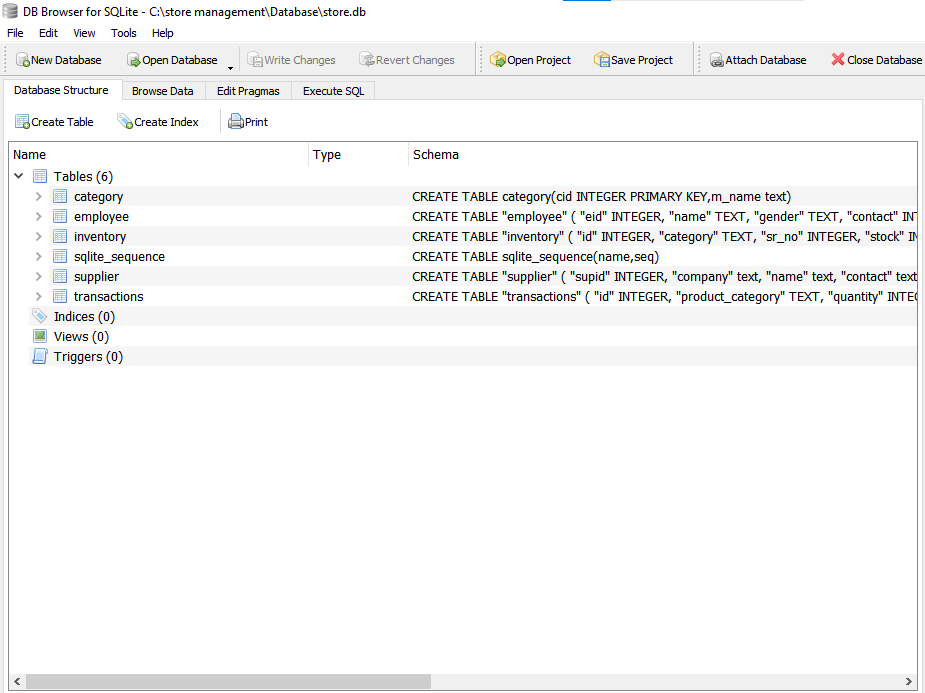




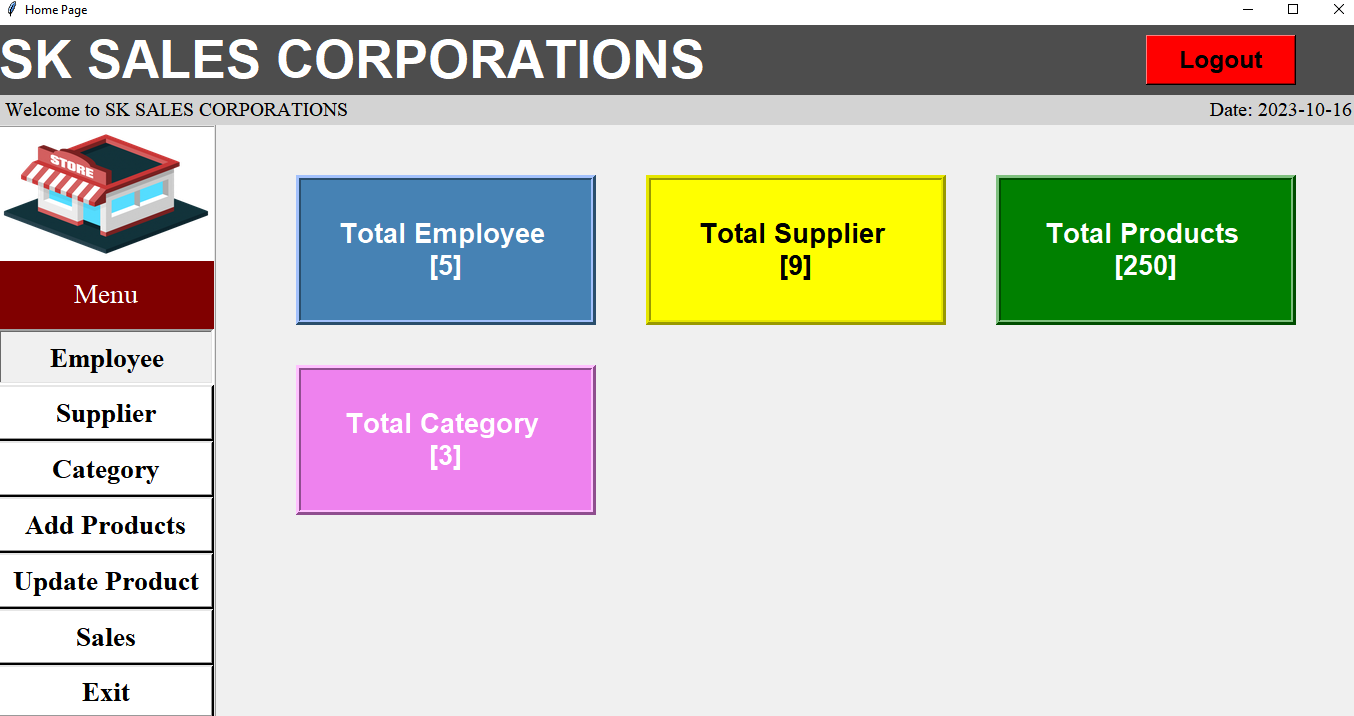


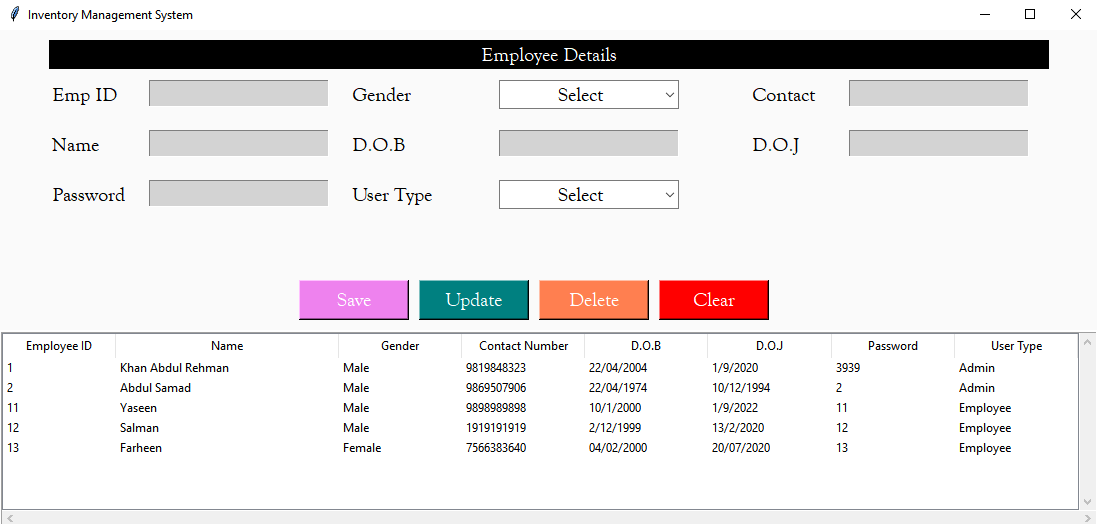
**DATABASE FOLDER**

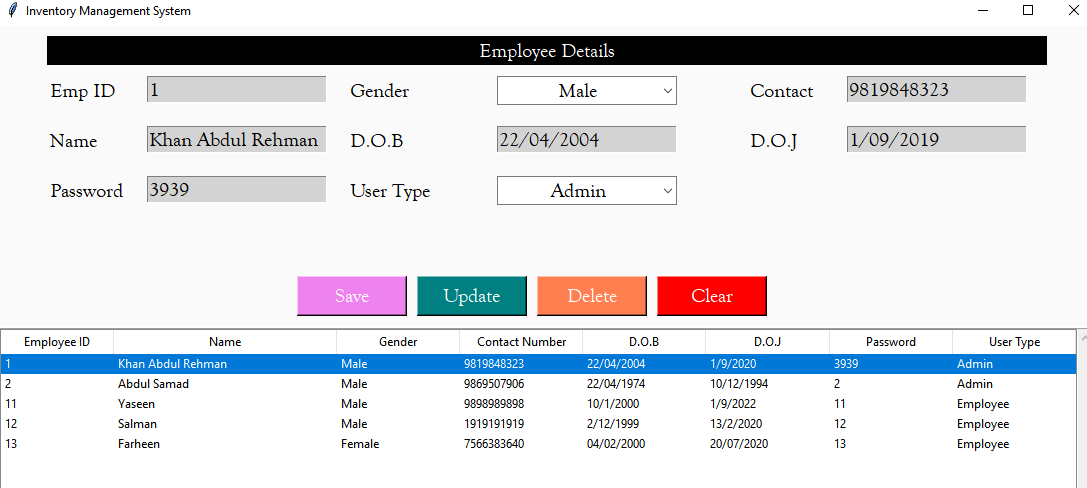
Database Structure



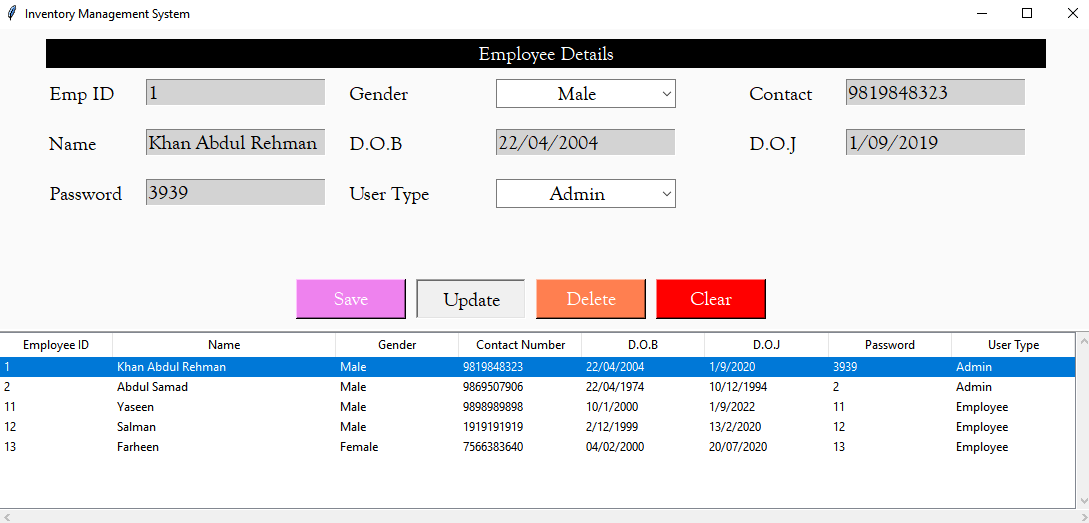
**Employee Page:**

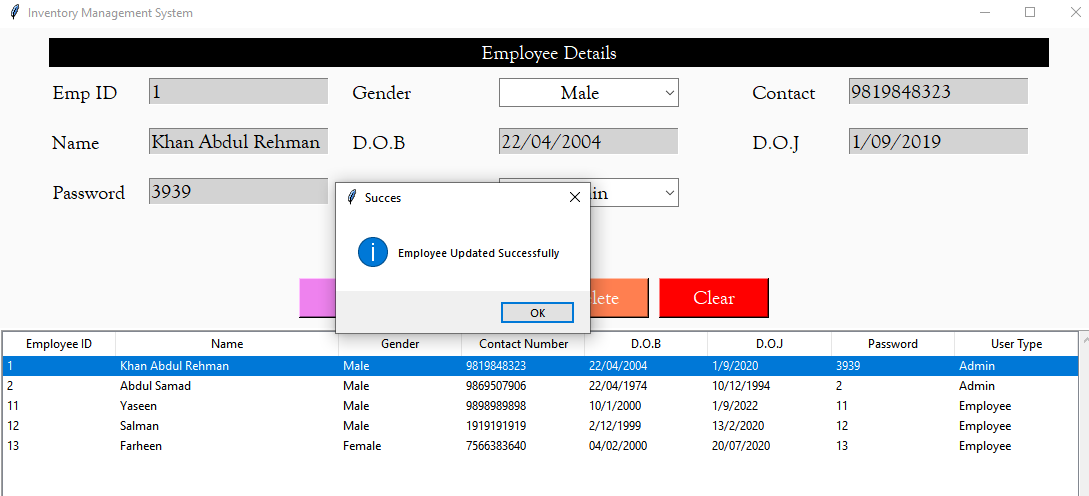
****



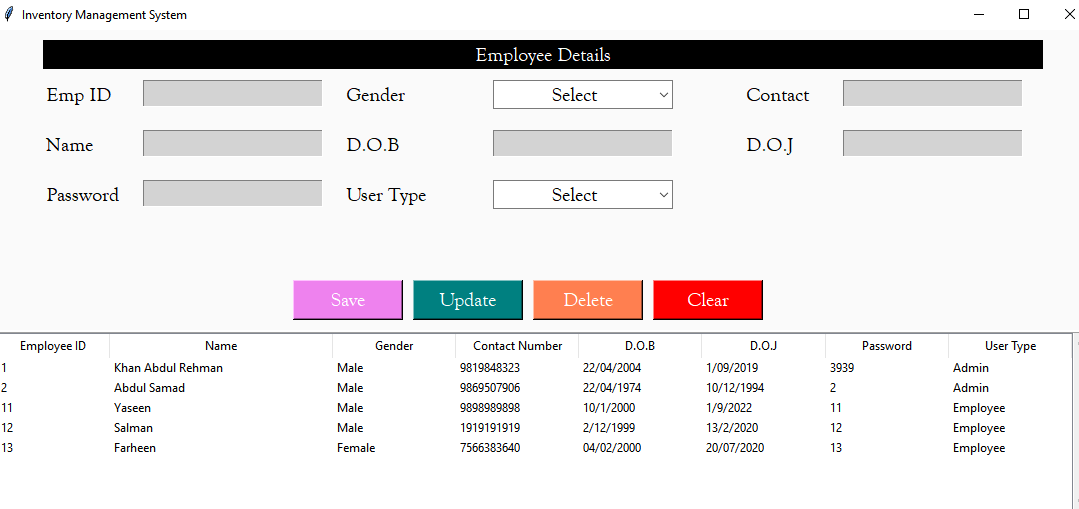


**After updating the joining year of admin:**

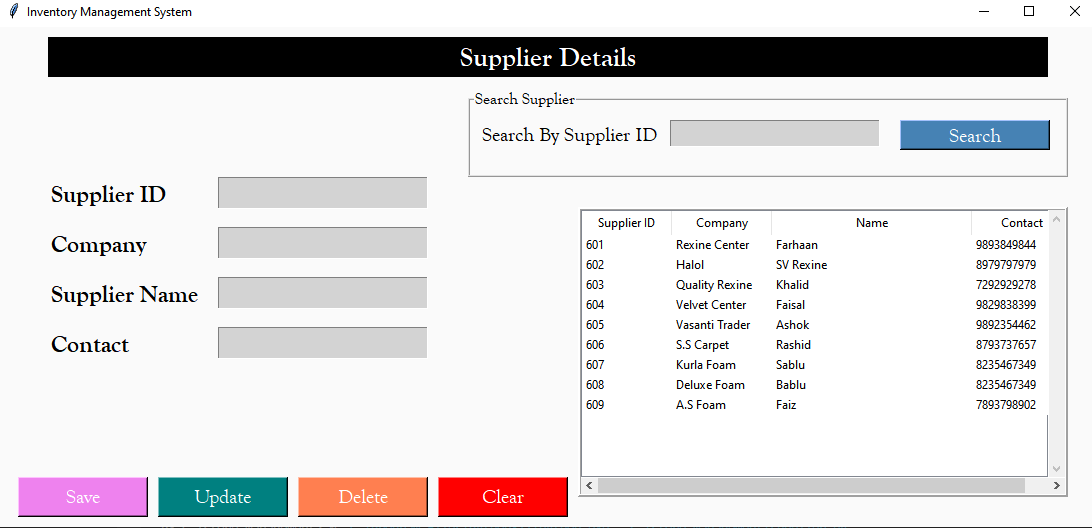




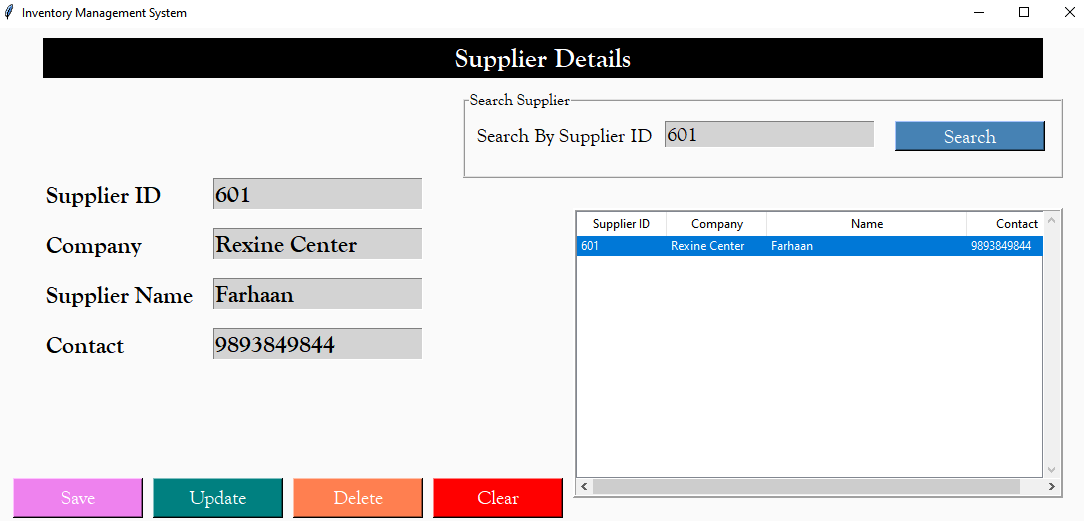
**After Clicking on clear it will clear all field:**

****

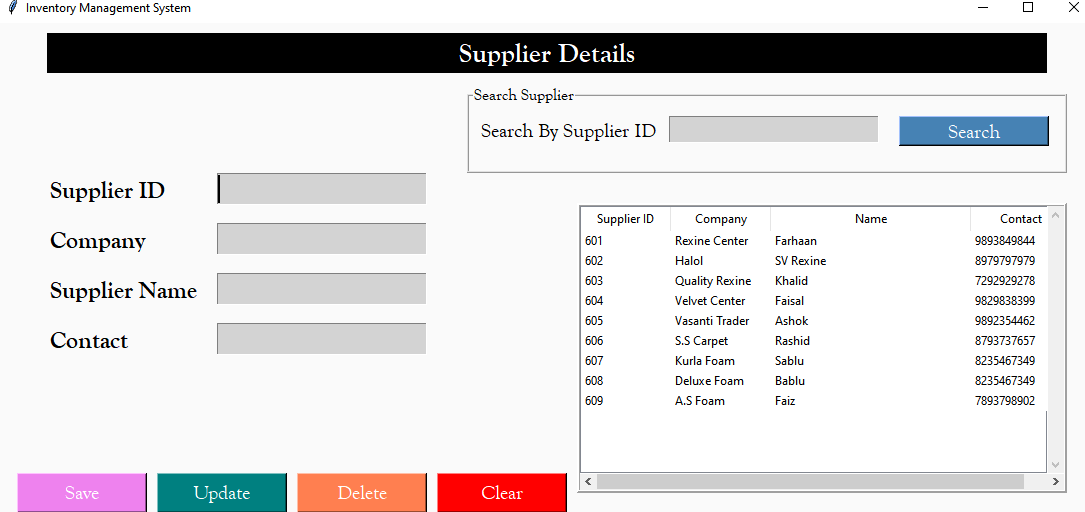
**Supplier page:**

****

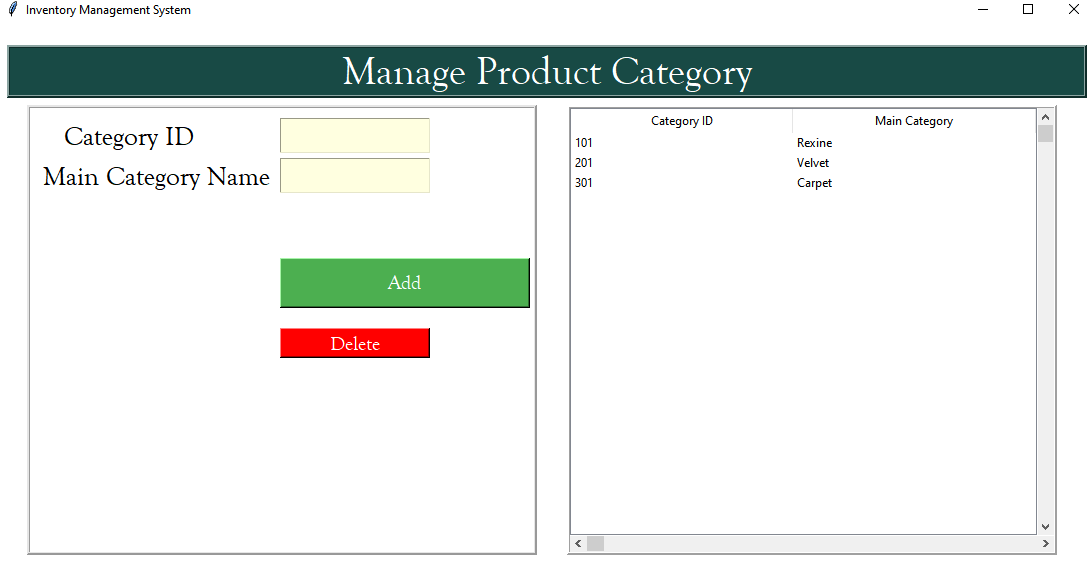
**Search by Supplier ID:**

****

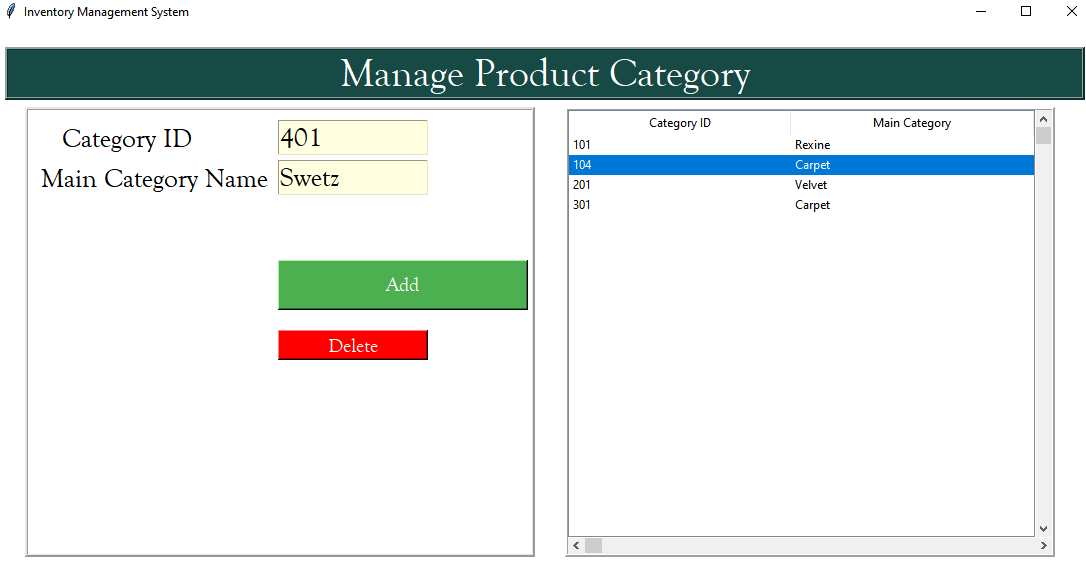
**After clicking Clear:**

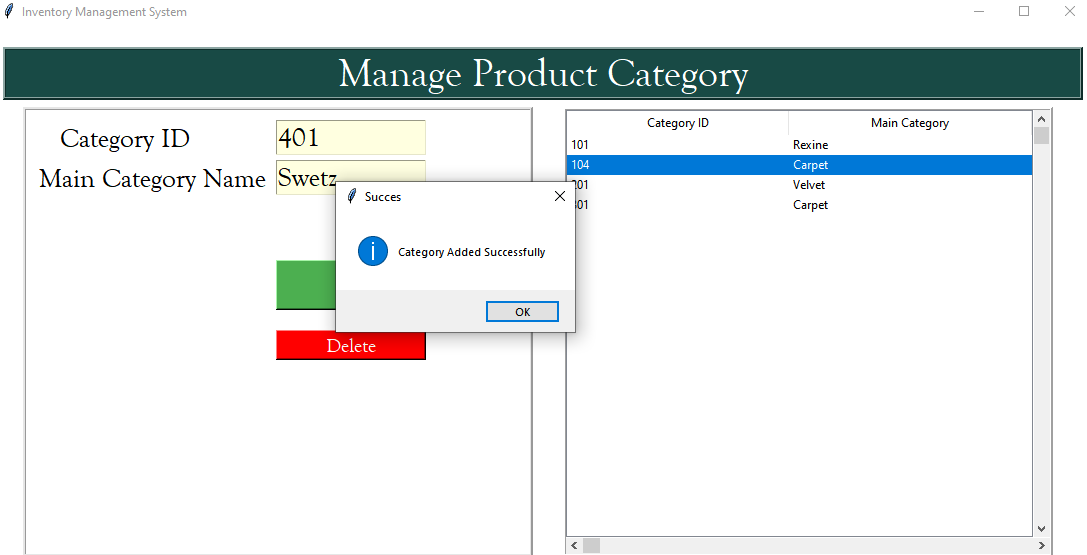
****

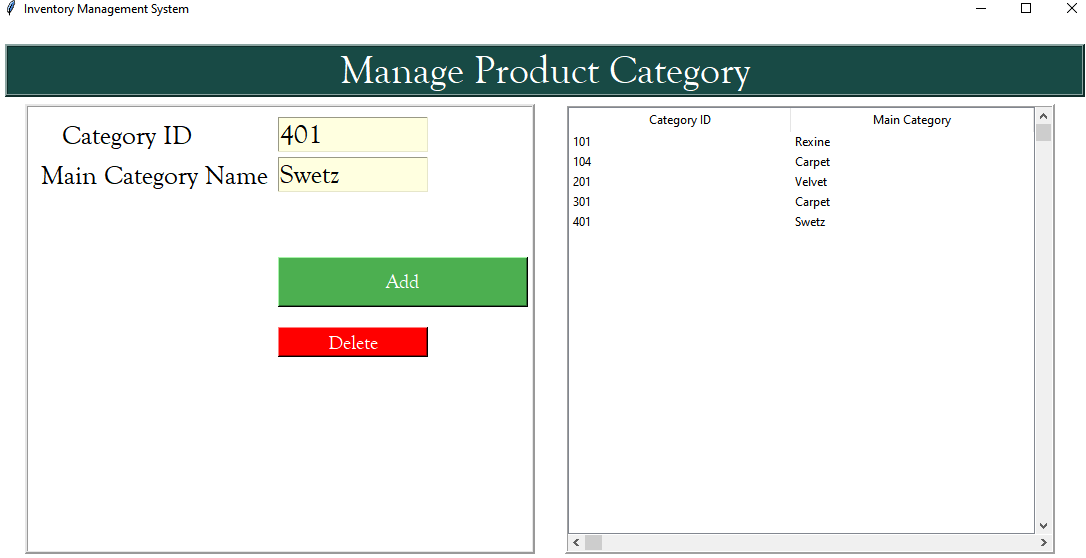
**Category Page:**

****

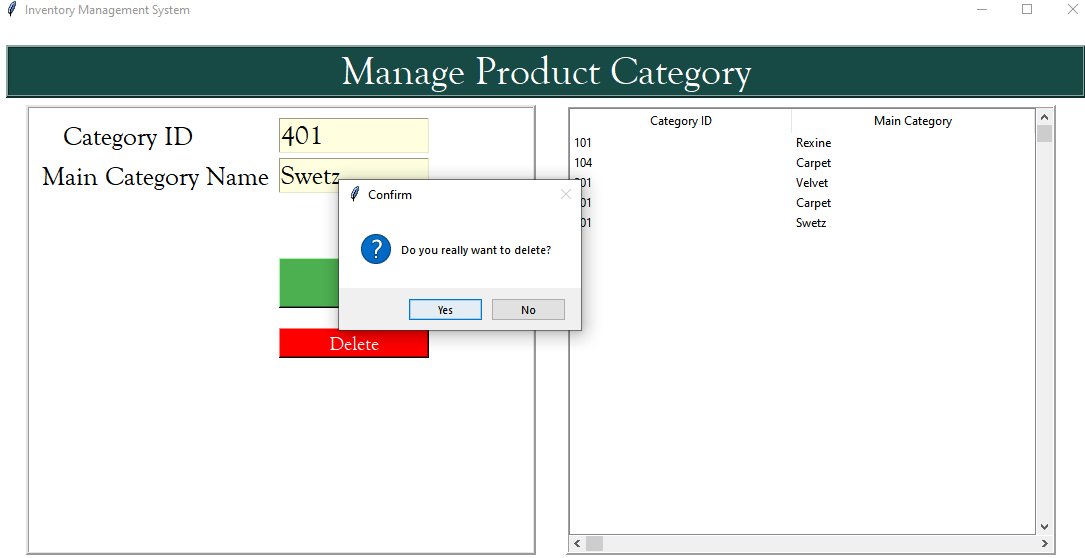
**Adding Category id by 401:**

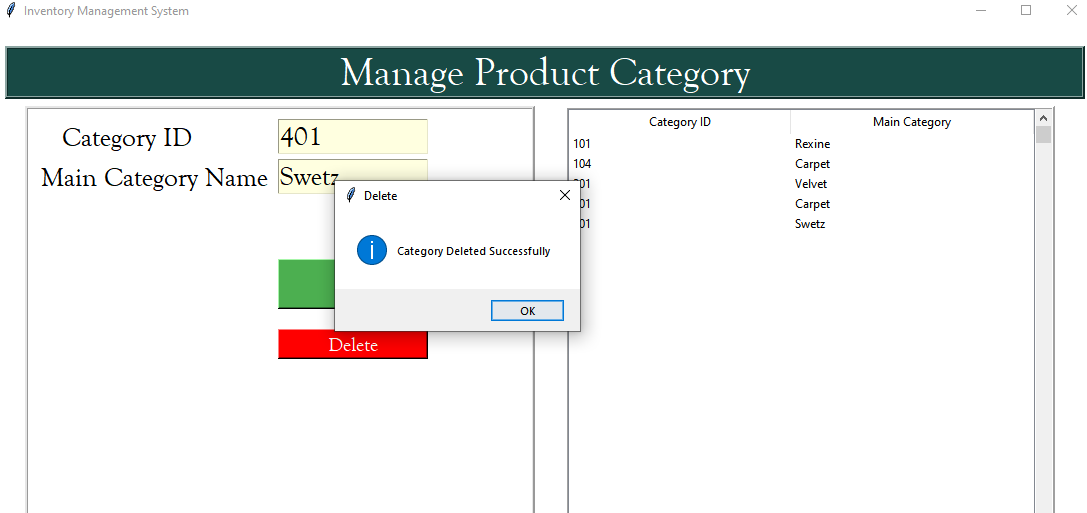
****

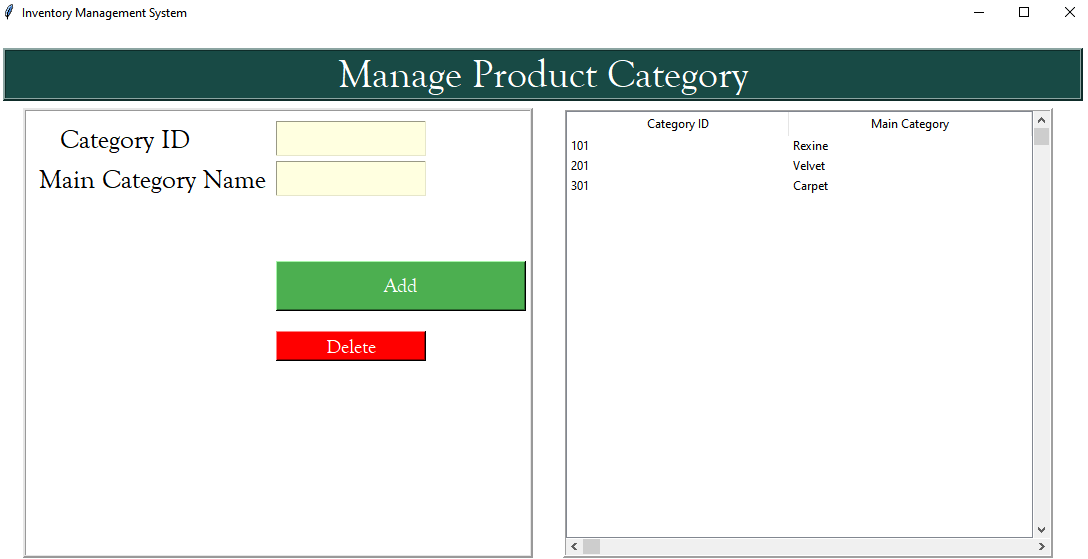
****

****

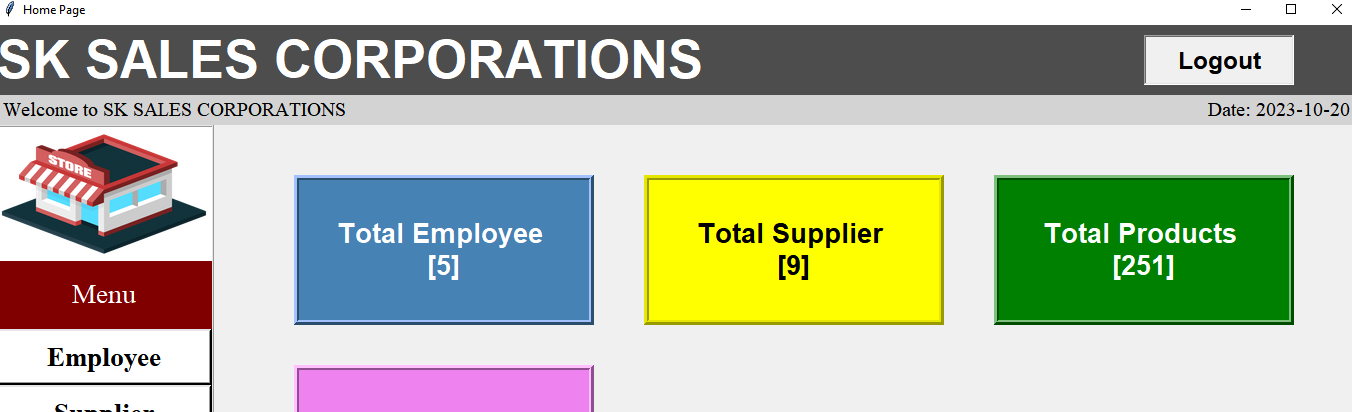
**For Deleting ID 401 Clicking On Delete button:**

****

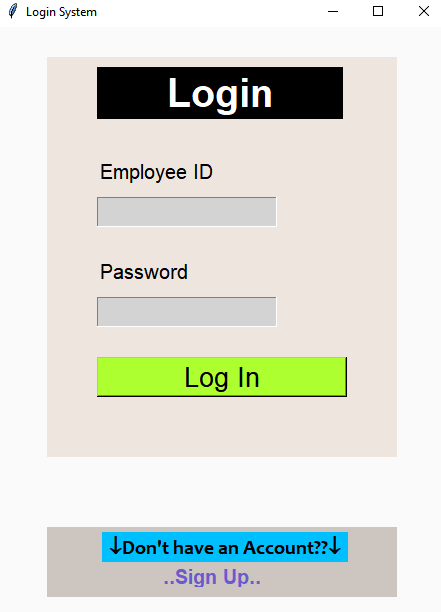
****

****

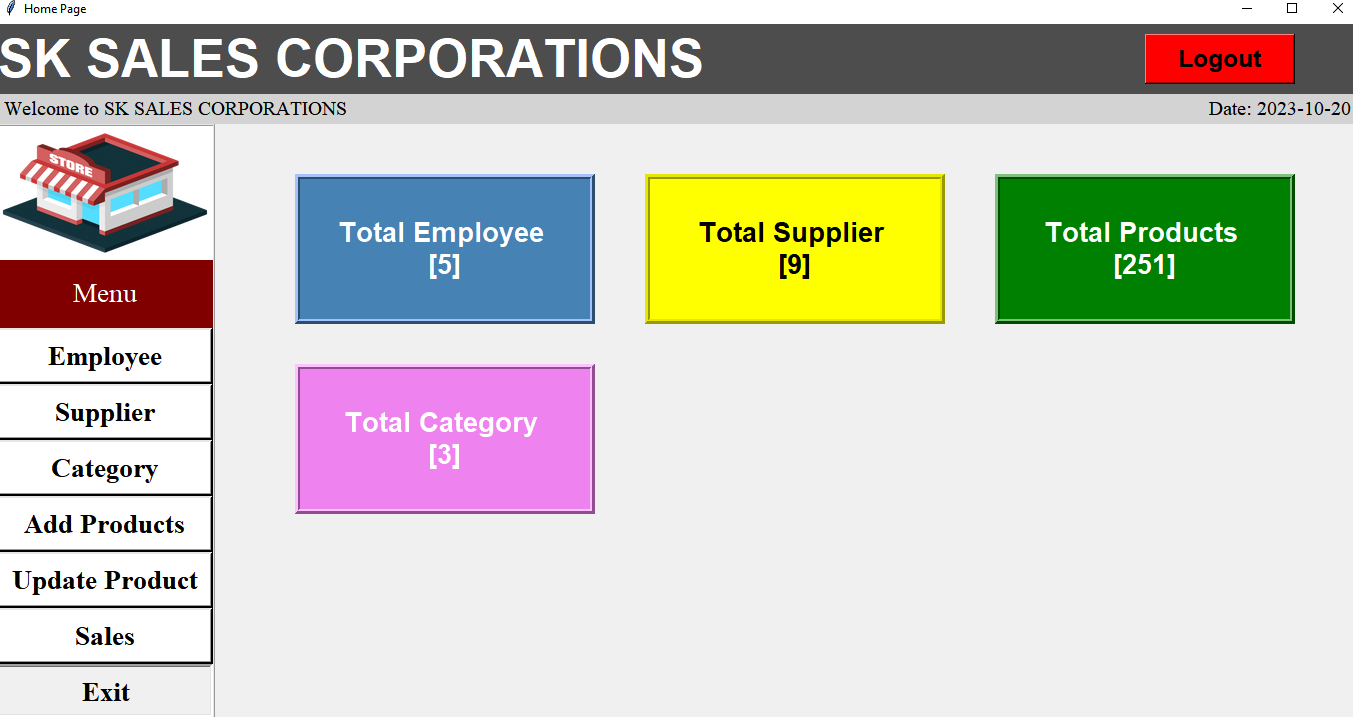
**For Logout Click On Logout Button:**

****

**Then We Will Go To Login page:**

****

**While Clicking Exit it will close the root**

****

**Chapter 8. Conclusion and Future Scope**

**8.1 Conclusion**

the Rexine Store Management System plays a pivotal role in optimizing the operations of a rexine store. Through efficient management of inventory, sales, customers, and suppliers, the system ensures smooth and streamlined business processes. It offers a range of functionalities, including product management, sales processing, customer relationship management, supplier interactions, and robust reporting capabilities.

By conducting positive testing, we have ascertained that the system performs well under normal conditions. It can accurately handle tasks such as product addition, inventory updates, sales transactions, and generating various reports. Moreover, the system demonstrates excellent user authentication, ensuring that only authorized personnel can access and manipulate sensitive data. Additionally, the implementation of features like search functionality and data backup and recovery enhances user experience and data security.

Furthermore, negative testing has been instrumental in identifying potential vulnerabilities and weaknesses within the system. By subjecting the system to various adverse conditions, such as invalid data inputs, out-of-stock scenarios, unauthorized access attempts, and network failures, we have tested its resilience. The Rexine Store Management System has proven its robustness by appropriately handling errors, preventing unauthorized access, ensuring data integrity, and recovering gracefully from unexpected interruptions.

In essence, the Rexine Store Management System not only enhances the efficiency and accuracy of day-to-day operations but also provides a secure and reliable platform for managing critical business processes. Its positive features ensure a seamless user experience, while the successful handling of negative scenarios showcases its reliability and resilience in challenging situations. As a comprehensive solution, the system stands as an invaluable asset for any rexine store, facilitating effective management and contributing significantly to the overall success of the business.

**8.1.1 Advantages**

Rexine Store Management System (RSMS) can offer various advantages to businesses in the rexine industry.

**Efficient Inventory Management:**

RSMS helps in tracking rexine rolls, sheets, and related products efficiently. It maintains accurate stock levels and reduces the chances of overstocking or stockouts.

**Streamlined Sales Process:**

The system simplifies the sales process, allowing for easy product searches, quick order processing, and generating invoices. It enhances the speed and accuracy of transactions.

**Effective Supplier Management:**

RSMS enables businesses to manage interactions with suppliers, track orders, and maintain a smooth supply chain. This ensures a steady supply of rexine materials.

**Enhanced Security:**

RSMS allows role-based access control, ensuring that only authorized personnel can access sensitive information. This improves data security and confidentiality.

**Cost Savings:**

By automating various tasks, businesses can save costs related to labor, paperwork, and operational inefficiencies. It leads to overall cost savings in the long run.

**8.1.2 Limitations**

Rexine Store Management System (RSMS) offers numerous advantages, it also has certain limitations and challenges. Here are some common limitations associated with RSMS:

**Complexity**: RSMS can be complex, especially if it's highly customized or integrated with other systems. Complexity might lead to difficulties in implementation, maintenance, and user adoption.

**Data Security Concerns:** Storing sensitive customer data, sales records, and inventory details electronically poses security risks. If not properly secured, this data can be vulnerable to hacking, data breaches, or unauthorized access.

**Dependency on Technology:** RSMS relies heavily on technology. System downtimes, software bugs, or hardware failures can disrupt operations and cause inconvenience to customers.

**Data Loss:** Despite regular backups, there is always a risk of data loss due to unforeseen circumstances such as natural disasters, hardware failures, or software glitches. Regular backup procedures are necessary to mitigate this risk.

**8.2. Future Enhancement**

The future enhancement of a Rexine Store Management System (RSMS) can involve implementing advanced features and technologies to further streamline operations, improve customer experience, and stay ahead in the competitive market. Here are some potential future enhancements for an RSMS:

**Mobile Application:** Develop a mobile app version of the RSMS to allow customers to browse products, place orders, and track their purchases from their smartphones. This enhances customer convenience and expands the store's reach.

**E-commerce Integration:** Integrate the RSMS with e-commerce platforms to enable online sales. This integration can attract a broader customer base and provide 24/7 access to products, boosting sales and revenue.

**Customer Relationship Management (CRM):** Enhance the CRM capabilities of the system. Implement features like personalized product recommendations, customer feedback analysis, and loyalty programs to strengthen customer relationships and improve customer retention.

**Chatbots and Virtual Assistants:** Implement chatbots or virtual assistants on the website or mobile app. These AI-driven assistants can handle customer inquiries, provide product information, and assist in the purchasing process, enhancing customer support.

**Chapter 9. References**

**9.1 References**

• <https://youtu.be/yQSEXcf6s2I?si=lydBceeMMW54U6lu>

• <https://www.w3schools.in/python/gui-programming>

• <https://www.javatpoint.com/python-tkinter>

• <https://www.geeksforgeeks.org/python-tkinter-tutorial/>

• <https://www.geeksforgeeks.org/python-tkinter-label/?ref=lbp>

• <https://www.geeksforgeeks.org/python-tkinter-message/?ref=lbp>