**SUPERMARKET MANAGEMENT SYSTEM**

A MINI PROJECT REPORT

Submitted by

RUDRAPRIYAN N 220701232

REUBEN ABRAHAM GEORGE 220701223

# In partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE

RAJALAKSHMI ENGINEERING COLLEGE (AUTONOMOUS)

THANDALAM

CHENNAI-602105

2023-24

**BONAFIDE CERTIFICATE**

Certified that this project report “**SUPER MARKET MANAGEMENT SYSTEM**” is the bonafide work of “**RUDRAPRIYAN N (220701232), REUBEN ABRAHAM GEORGE (220701223)**” who carried out the project work under my supervision.

**Submitted for the Practical Examination held on** \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**SIGNATURE**

**Mrs.K.Mahesmeena**

**Assistant Professor,**

**Computer Science and Engineering,**

**Rajalakshmi Engineering College,**

**(Autonomous),**

**Thandalam.**

**Chennai – 602 105**

**ABSTRACT**

The Supermarket Management System (SMS) is designed to streamline and automate the key operations of a supermarket, focusing on product management, category management, billing, and administrative oversight. The product management module allows administrators to efficiently manage the supermarket's inventory by adding, viewing, updating, and deleting products, complete with details such as product ID, name, rate, and category. Category management supports the creation and maintenance of product categories, ensuring an organized and accessible product catalog. The billing system generates accurate bills based on customer purchases, with functionalities to view, update, and delete billing information as needed. Additionally, the admin management module provides secure login for administrators, comprehensive user management, and the ability to oversee the entire supermarket operation. By integrating these modules, the SMS enhances operational efficiency, reduces manual errors, and improves service delivery, thereby supporting the supermarket in achieving higher levels of customer satisfaction and operational effectiveness.

* 1. **INTRODUCTION**

In today's competitive sales environment, efficient management systems are crucial for the efficient operation of supermarkets. The Supermarket Management System is an innovative solution designed to automate and streamline the operations of a supermarket such as customer and product management. This project report details the design and development of our Supermarket Management System (SMS) that aims to streamline the operations, improve customer service, and enhance overall efficiency within the supermarket.

* 1. **OBJECTIVES**

1. Customer Management: To maintain detailed records of customers, including their identification number, name, contact information and purchase history.
2. Product Management: To manage product details, including product ID, name, rate, and category ensuring that all product-related data is accurate and up-to-date.
3. Billing System: To facilitate an efficient billing process that accurately reflects customer purchases and manages payment transactions.
4. Administrative Control: To provide administrators with tools to manage product categories and oversee the entire supermarket operation through secure login credentials.
5. Category Management: To categorize products efficiently, ensuring easy retrieval and management of products.
   1. **MODULES**

**Product Management Module:**

* **Add Product:** Enter new products into the system with details like product ID, name, rate, and category.
* **View Product:** Display product information and availability.
* **Update Product:** Modify details of existing products.
* **Delete Product:** Remove products from the system.

**Category Management Module:**

* **Create Category:** Create new product categories.
* **View Category:** Display the list of categories.
* **Update Category:** Modify category information.
* **Delete Category:** Remove categories from the system.

**Billing System Module:**

* **Generate Bill:** Create bills based on customer purchases, including product details and quantities.
* **View Bill:** Retrieve and display bill details.
* **Update Bill:** Modify billing information if needed.
* **Delete Bill:** Remove bills from the system.

**Admin Management Module:**

* **Admin Login:** Secure login for administrators using username and password.
* **Manage Users:** Ability to add, view, update, and delete administrator accounts.
* **Oversee Operations:** Monitor and manage overall supermarket operations.

**Security Module:**

* **Data Protection:** Ensure the protection of sensitive data through encryption and secure storage.
* **User Authentication:** Implement robust user authentication mechanisms to prevent unauthorized access.
* **Compliance:** Ensure the system complies with relevant data protection regulations and standards.

**2. SURVEY OF TECHNOLOGIES**

**2.1 SOFTWARE DESCRIPTION**

The Supermarket Management System employs a robust software architecture to ensure scalability, reliability, and performance. The system is built using a combination of backend and frontend technologies to create a seamless user experience and efficient data management processes.

**2.2 LANGUAGES**

The Supermarket Management System utilizes several programming languages and database technologies to deliver its functionalities effectively. Key languages and technologies used include SQL and Python.

**2.2.1 SQL**

SQL (Structured Query Language) is used for managing and manipulating the relational database that stores all hotel data. SQL provides the tools necessary for querying the database, updating records, and ensuring data integrity. Key features of SQL used in the SMS include:

Data Definition Language (DDL): For defining database schema, creating, altering, and deleting tables.

Data Manipulation Language (DML): For inserting, updating, deleting, and querying data.

Data Control Language (DCL): For controlling access to data through permissions and roles.

### **2.2.2 PYTHON**

Python is a versatile, high-level programming language used in the development of the Hotel Management System. Python's simplicity and extensive libraries make it suitable for various aspects of the SMS, including:

Backend Development: Managing server-side logic, handling requests, and processing data.

Data Analysis: Utilizing libraries like Pandas and NumPy for data manipulation and analysis.

Integration: Facilitating integration with other systems and services through APIs.

Automation: Automating routine tasks and workflows within the system.

By leveraging these technologies, the Supermarket Management System ensures efficient data management, seamless operation, and a user-friendly experience for both hotel staff and guests.

**3.REQUIREMENTS AND ANALYSIS**

**3.1 REQUIREMENT SPECIFICATION**

**Functional Requirements**

1. Product Management

* Add Product: Enter new products into the system with details like product ID, name, rate, and category.
* View Product: Display product information and availability.
* Update Product: Modify details of existing products.
* Delete Product: Remove products from the system.

1. Category Management

* Category: Create new product categories.
* Add View Category: Display the list of categories.
* Update Category: Modify category information.
* Delete Category: Remove categories from the system.

1. Billing System

* Generate Bill: Create bills based on customer purchases, including product details and quantities.
* View Bill: Retrieve and display bill details.
* Update Bill: Modify billing information if needed.
* Delete Bill: Remove bills from the system.

1. Admin Management

* Admin Login: Secure login for administrators using username and password.
* Manage Users: Ability to add, view, update, and delete administrator accounts.
* Oversee Operations: Monitor and manage overall supermarket operations.

**Non-Functional Requirements**

* The system should handle multiple users simultaneously without significant delay.
* The user interface should be intuitive and easy to navigate for all types of users, including administrators and cashiers.
* Data should be securely stored, with encryption where necessary, to protect sensitive information.
* Proper authentication and authorization mechanisms should be in place to restrict access to certain functionalities.

**3.2 HARDWARE AND SOFTWARE REQUIREMENTS**

Processor: Intel Xeon or AMD EPYC, 2.4 GHz or higher

Memory: Minimum 16 GB RAM (32 GB recommended for larger operations)

Storage: At least 500 GB SSD for fast access and reliability

Network: Gigabit Ethernet network interface card (NIC)

Backup: External hard drive or NAS with at least 1 TB capacity for backups

Client Requirements:

Processor: Intel Core i5 or AMD Ryzen 5, 2.0 GHz or higher

Memory: Minimum 8 GB RAM

Storage: At least 250 GB HDD or SSD

Display: 15.6” monitor with 1920x1080 resolution or higher

Network: Ethernet or Wi-Fi connectivity

Software Requirements

Server Software:

Operating System: Windows Server 2016/2019, or Linux (Ubuntu Server 18.04 or later)

Database Management System: MySQL 8.0 or PostgreSQL 12

Web Server: Apache 2.4 or Nginx

Application Server: Node.js 14.x or later

Backup Software: Acronis Backup or similar

Client Software:

Operating System: Windows 10, macOS Catalina or later

Web Browser: Google Chrome, Mozilla Firefox, or Microsoft Edge (latest versions)

Office Suite: Microsoft Office 2019 or LibreOffice 7.0

PDF Reader: Adobe Acrobat Reader or similar

Development Tools:

IDE: Visual Studio Code, IntelliJ IDEA, or Eclipse

Version Control: Git with GitHub or GitLab

Project Management: JIRA, Trello, or Asana

Testing Tools: Selenium, Postman for API testing

Security Software:

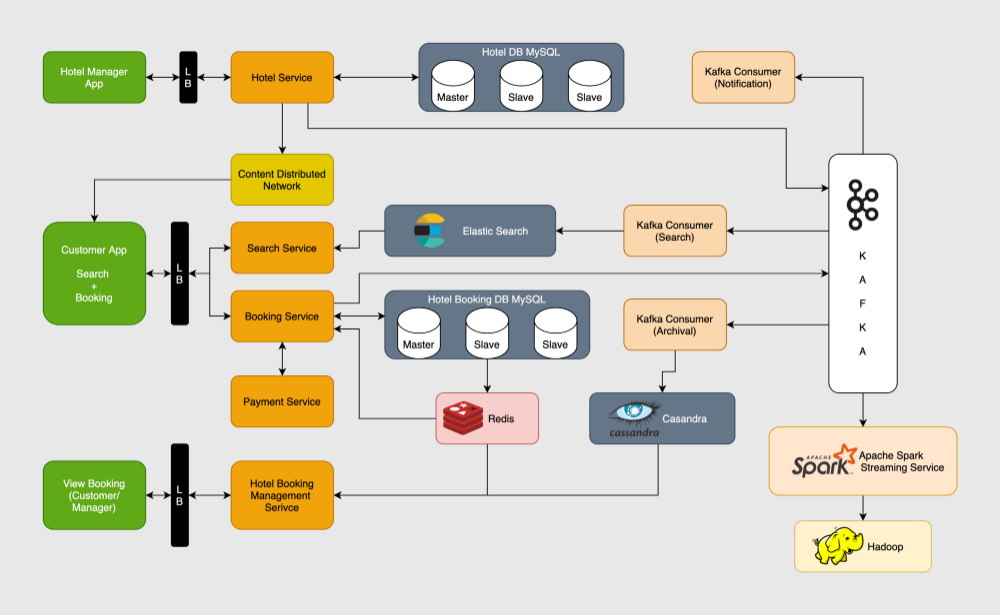
Firewall: UFW (for Linux), Windows Firewall

Antivirus: Bitdefender, Norton, or equivalent

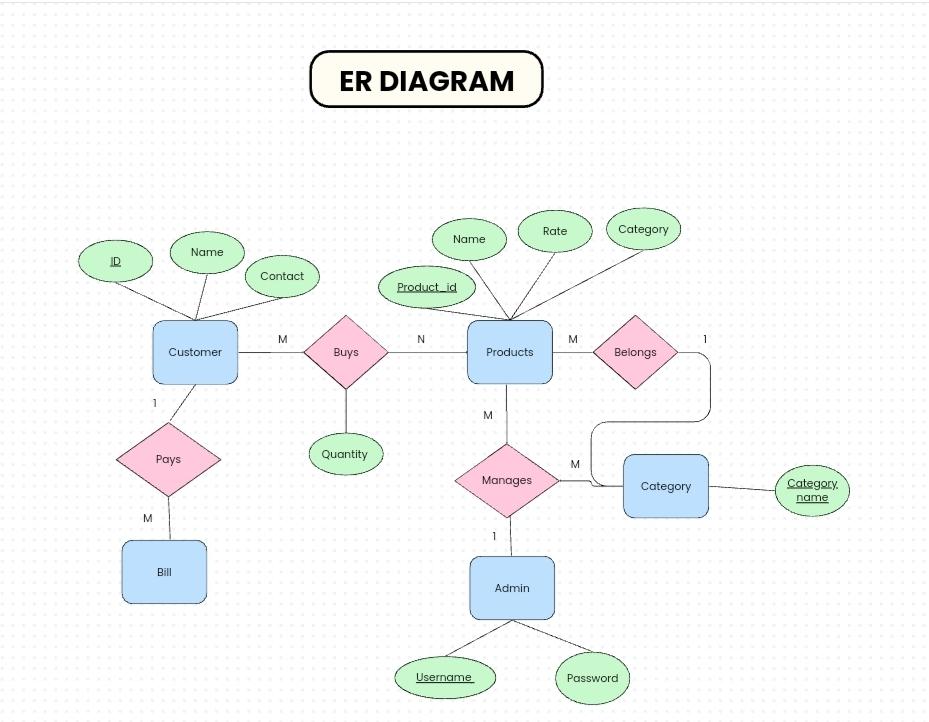
Encryption: SSL/TLS certificates for secure communication

These requirements ensure that the Hotel Management System operates smoothly, providing efficient performance, scalability, and security for managing hotel operations effectively.

**3.3 ARCHITECTURE DIAGRAM**



* 1. **ER DIAGRAM**



**3.5 NORMALIZATION**

**Category Table**

|  |  |  |
| --- | --- | --- |
| **Column** | **Type** | **Constraint** |
| category | varchar(100) | Primary Key |

**Products Table**

|  |  |  |
| --- | --- | --- |
| **Column** | **Type** | **Constraint** |
| product\_id | int | Primary Key |
| product\_name | varchar(100) | Not Null |
| product\_rate | int | Not Null |
| category | varchar(100) | Not Null, Foreign Key references category(category) |

**Admin Table**

|  |  |  |
| --- | --- | --- |
| **Column** | **Type** | **Constraint** |
| username | varchar(20) | Primary Key |
| password | varchar(20) | Not null |

**Customer Table**

|  |  |  |
| --- | --- | --- |
| **Column** | **Type** | **Constraint** |
| customer\_id | int | Primary Key |
| customer\_id | varchar(20) | Not null |
| customer\_phone | varchar(10) | Not null |
| customer\_email | varchar(50) | Not null |

### **1NF (First Normal Form)**

1NF requires that:

1. Each table has a primary key.
2. All columns contain atomic (indivisible) values.
3. Each column contains only one value per row (no repeating groups or arrays).

Tables meet 1NF requirements:

* Each table has a primary key: category, product\_id, username, and customer\_id.
* All columns contain atomic values (e.g., product\_name, customer\_email).
* No column contains multiple values or repeating groups.

### **2NF (Second Normal Form)**

2NF requires that:

1. The table is already in 1NF.
2. All non-key attributes are fully functionally dependent on the primary key (no partial dependencies).

Tables meet 2NF requirements:

* The category table has only one column aside from the primary key, so no partial dependencies.
* The products table's non-key attributes (product\_name, product\_rate, category) are fully dependent on the primary key product\_id.
* The admin table's non-key attribute (password) is fully dependent on the primary key username.
* The customer table's non-key attributes (customer\_name, customer\_email, customer\_phone) are fully dependent on the primary key customer\_id.

### **3NF (Third Normal Form)**

3NF requires that:

1. The table is already in 2NF.
2. All attributes are directly dependent on the primary key (no transitive dependencies).

Tables meet 3NF requirements:

* The category table has no other attributes besides the primary key, so it trivially satisfies 3NF.
* The products table's attributes (product\_name, product\_rate, category) are directly dependent on the primary key product\_id and not on other non-key attributes.
* The admin table's attribute (password) is directly dependent on the primary key username.
* The customer table's attributes (customer\_name, customer\_email, customer\_phone) are directly dependent on the primary key customer\_id.

### **BCNF (Boyce-Codd Normal Form)**

BCNF requires that:

1. The table is already in 3NF.
2. For every functional dependency (X → Y), X is a superkey (a superkey is a set of one or more columns that can uniquely identify a row in the table).

Tables meet BCNF requirements:

* The category table has category as the primary key and only column, which is a superkey.
* The products table has product\_id as the primary key, and all dependencies involve the primary key, making product\_id a superkey.
* The admin table has username as the primary key, and the dependency username → password involves the primary key, making username a superkey.
* The customer table has customer\_id as the primary key, and all dependencies involve the primary key, making customer\_id a superkey.

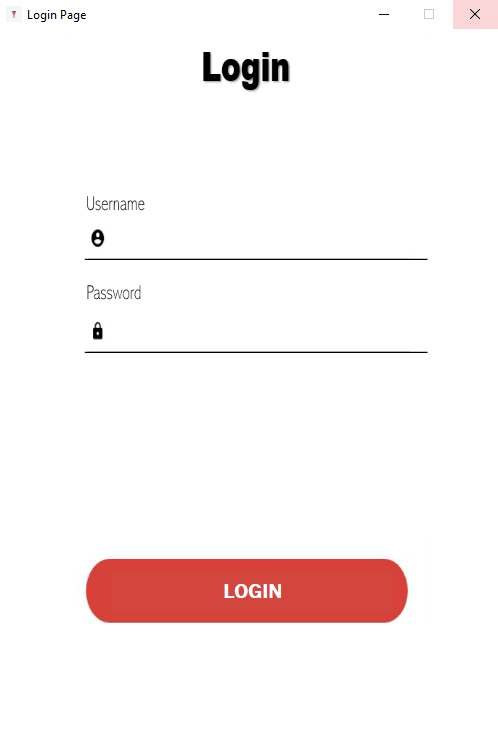
**4. PROGRAM CODE**

|  |
| --- |
| #dbConfig.py  import sqlite3   dbconn = sqlite3.connect("./Database/RSgroceries.db")  cursor = dbconn.cursor()   cursor.execute("""CREATE TABLE if not exists category( category varchar(100) NOT NULL primary key  )  """) dbconn.commit() cursor.execute("""CREATE TABLE if not exists products(  product\_id int not null primary key,  product\_name varchar(100) not null,  product\_rate int not null,  category varchar(100) not null references category(category)  )  """) dbconn.commit()  cursor.execute(""" CREATE TABLE if not exists admin( username varchar(20) not null primary key, password varchar(20) not null ); """) dbconn.commit()   products = [  ['101', 'Maaza 1 litre', '65', 'Beverages'],  ['102', 'Coco Cola 1 litre', '70', 'Beverages'],  ['103', 'Fanta 1 litre', '66', 'Beverages'],  ['104', 'Miranda 1 litre', '72', 'Beverages'],  ['105', '7 UP 1 litre', '60', 'Beverages'],  ['106', 'Bovanto 1/2 litre', '35', 'Beverages'],  ['107', 'Frooti 1/2 litre', '40', 'Beverages'],  ['108', 'Pepsi 1/2 litre', '30', 'Beverages'],  ['109', 'Apple Juice 1/2 litre', '25', 'Beverages'],  ['110', 'Sprite 1/2 litre', '35', 'Beverages'],  ['111', 'Aavin Milk 1 litre', '50', 'Dairy'],  ['112', 'Aavin Milk 1/2 litre', '26', 'Dairy'],  ['113', 'Aavin Milk 250 ml', '12', 'Dairy'],  ['114', 'Amul Butter 100 g', '46', 'Dairy'],  ['115', 'Arokya Curd 1 litre', '55', 'Dairy'],  ['116', 'Aavin Curd 1 litre', '54', 'Dairy'],  ['117', 'Amul Ghee 500 g', '245', 'Dairy'],  ['118', 'MM Paneer', '230', 'Dairy'],  ['119', 'Bhav Cheese 500g', '75', 'Dairy'],  ['120', 'Cond. Milk 250ml', '90', 'Dairy'],  ['121', 'Chilli Sauce 500g', '118', 'Sauce'],  ['122', 'Sweent&Chilli Sauce 500g', '108', 'Sauce'],  ['123', 'Tomato Sauce 500g', '100', 'Sauce'],  ['124', 'Soya Sauce 500g', '110', 'Sauce'],  ['125', 'Hot Tomato Sauce 500g', '115', 'Sauce'],  ['126', 'Salt Bread', '21', 'Bread'],  ['127', 'Milk Bread', '22', 'Bread'],  ['128', 'Wheat Bread', '20', 'Bread'],  ['129', 'Chicken Wings 400g', '270', 'Meat'],  ['130', 'Chicken Breast 250g', '240', 'Meat'],  ['131', 'Pork 500g', '200', 'Meat'],  ['132', 'Beaf 1Kg', '290', 'Meat'],  ['133', 'Chicken Boneless 500g', '250', 'Meat'],  ['134', 'Chicken Leg Pie 1Kg', '190', 'Meat'],  ['135', 'Full Chicken ', '470', 'Meat'],  ['136', '1Kg Basmati Rice', '200', 'Rice'],  ['137', '1Kg Idli Rice ', '275', 'Rice'],  ['138', '1Kg Tiffin Rice', '230', 'Rice'],  ['139', '1Kg Basmati Rice', '200', 'Rice'],  ['140', 'Ashir Atta 1Kg ', '45', 'Cereals'],  ['141', 'RS Oats 500g ', '30', 'Cereals'],  ['142', 'RS Frosted Flakes 500g ', '50', 'Cereals'],  ['143', 'RS Oats 500g ', '30', 'Cereals'],  ['144', 'RS Flakes 200g ', '17', 'Cereals'],  ['145', 'RS Oats 500g ', '30', 'Cereals'],  ['146', 'RS Baking Soda 550g ', '235', 'Bakery'],  ['147', 'RS Baking Powder 1Kg ', '60', 'Bakery'],  ['148', 'Cake 1Kg', '50', 'Bakery'],  ['149', 'Choclate Cake 1piece', '15', 'Bakery'],  ['150', 'Strawberry Pastries 1pie', '15', 'Bakery'],  ['151', 'Cream Bun', '10', 'Bakery'],  ['152', 'Butter Biscuits', '12', 'Bakery'],  ['153', 'Natraj 10 Pencils ', '50', 'Stationary'],  ['154', 'Natraj Ge. Box', '60', 'Stationary'],  ['155', 'Natraj LS Scale', '10', 'Stationary'],  ['156', 'Natraj SS Scalw', '5', 'Stationary'],  ['157', 'DOMS ColourPencils 10', '20', 'Stationary'],  ['158', 'DOMS Oil Pastels', '30', 'Stationary'],  ['159', 'Natraj Sharpner', '3', 'Stationary'],  ['160', 'FaberCastle M.pencil 0.7', '15', 'Stationary'],  ['161', 'Apsara 0.7 led box ', '10', 'Stationary'],  ['162', 'Lizol 500ml', '65', 'Hygiene'],  ['163', 'Lizol l Litre', '120', 'Hygiene'],  ['164', 'Harpic 500ml', '70', 'Hygiene'],  ['165', 'Colgate Toothpaste BS', '25', 'Hygiene'],  ['166', 'Pantanjli Toothpaste', '30', 'Hygiene'],  ['167', 'Oral B Toothbrush', '15', 'Hygiene'],  ['168', 'Close Up Toothpaste S', '20', 'Hygiene'],  ['169', 'Colgate Toothbrush', '17', 'Hygiene'],  ['170', 'MouthWasher 500ml', '50', 'Hygiene'],  ['171', 'Sanitiser 500ml', '60', 'Hygiene'],  ['172', 'Horlicks 350g', '49', 'Health'],  ['173', 'Boost 500g', '100', 'Health'],  ['174', 'Complan 500g', '45', 'Health'],  ['175', 'Lays Blue S', '5', 'Snacks'],  ['176', 'Lays Red S', '5', 'Snacks'],  ['177', 'Lays Yellow S', '5', 'Snacks'],  ['178', 'Lays Green S', '5', 'Snacks'],  ['179', 'Lays Orange S', '5', 'Snacks'],  ['180', 'Bingo Mad Angles S', '5', 'Snacks'],  ['181', 'Bingo Mad Angles B', '10', 'Snacks'],  ['182', 'Taka Tak B', '10', 'Snacks'],  ['183', 'Lays Blue B', '10', 'Snacks'],  ['184', 'Lays Blue B', '10', 'Snacks'],  ['185', 'Lays Green B', '10', 'Snacks'],  ['186', 'Lays Yellow B', '10', 'Snacks'],  ['187', 'Lays Red B', '10', 'Snacks'],  ['188', 'Lays Orange B', '10', 'Snacks'],  ['189', 'Jim Jam S', '5', 'Snacks'],  ['190', 'Jim Jam B', '10', 'Snacks'],  ['191', 'Bourbon Bis ', '10', 'Snacks'],  ['192', 'Cinnamon 50g ', '10', 'Seasonings'],  ['193', 'Pepper 50g ', '10', 'Seasonings'],  ['194', 'Fennugreek 50g ', '5', 'Seasonings'],  ['195', 'Chinese Sea. 50g ', '10', 'Seasonings'],  ['196', 'FCB Chicken M.', '10', 'Masalas'],  ['197', 'FCB Fish F M.', '10', 'Masalas'],  ['198', 'FCB Mutton M.', '10', 'Masalas'],  ['199', 'FCB Sambar M.', '10', 'Masalas'],  ['200', 'Arun cupI.', '15', 'IceCreams'],  ['201', 'Arun ConeI. S', '17', 'IceCreams'],  ['202', 'Arun ConeI. M', '25', 'IceCreams'],  ['203', 'Arun ConeI. B', '35', 'IceCreams'],  ['204', 'Jamai Kulfi.', '10', 'IceCreams'],  ['205', 'Aman Family Pack I.', '80', 'IceCreams']]   # Add datas for data in products:  try:  cursor.execute("INSERT INTO products VALUES(:product\_id, :product\_name, :product\_rate, :category)",  {  "product\_id": data[0],  "product\_name": data[1],  "product\_rate": data[2],  "category": data[3]  }  )  dbconn.commit()  except sqlite3.IntegrityError:  pass  # Category category\_values = [  ['Bakery'],  ['Beverages'],  ['Bread'],  ['Cereals'],  ['Dairy'],  ['Hygiene'],  ['IceCreams'],  ['Masalas'],  ['Meat'],  ['Rice'],  ['Sauce'],  ['Seasonings'],  ['Snacks'],  ['Stationary']]  for data\_1 in category\_values:  try:  cursor.execute("INSERT INTO category VALUES(:category)",  {"category": data\_1[0]}  )  dbconn.commit()  except sqlite3.IntegrityError:  pass  #Employee.py from tkinter import \* from tkinter import messagebox from tkinter.font import Font from tkinter import ttk import datetime import sqlite3  dbconn = sqlite3.connect("./Database/RSgroceries.db")   cursor = dbconn.cursor()   cursor.execute(""" Select \* From category """) Category\_1 = cursor.fetchall()  # Creating TKinter Window billing = Tk() billing.geometry("1330x750") billing.resizable(0, 0) billing.iconbitmap("./images/Logo.ico") billing.title("Employee") font\_1 = Font(family="Calibri",size=15,weight="bold")   # Fixing GUI Background Background = PhotoImage(file="./images/Employee\_bg.png") Bg\_label = Label(billing, image=Background) Bg\_label.place(x=0, y=0, relwidth=1, relheight=1)  # Logout command def Exit():  sure = messagebox.askyesno("Exit","Are you sure you want to exit?", parent=billing)  if sure == True:  billing.destroy()  # Creating logout button logout\_img = PhotoImage(file="./images/logout.png") logout\_button = Button(billing, image=logout\_img, borderwidth=0,relief="flat",overrelief="flat", command=Exit) logout\_button.place(relx=0.0155, rely=0.038,width=39,height=31)  "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_"  # Creating invoice invoice = ttk.Treeview(billing) invoice["columns"] = ("Product Name", "Qty", "Rate", "Cost")  invoice.column("#0", width=0,stretch=NO) invoice.column("#1", width=301,anchor="center") invoice.column("#2", width=80,anchor="center") invoice.column("#3",width=120,anchor="center") invoice.column("#4",width=120,anchor="center")  invoice.heading("#0",text="") invoice.heading("#1",text="Product Name") invoice.heading("#2",text="Qty") invoice.heading("#3",text="Rate") invoice.heading("#4",text="Cost") invoice.place(relx=0.5032,rely=0.4517,height=245)  Scroll\_invoice = Scrollbar(orient="vertical",command=invoice.yview) invoice.configure(yscroll=Scroll\_invoice.set) Scroll\_invoice.place(relx=0.9593,rely=0.4537, height=275) "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_"  # Creating all the entry fields # Creating Entry for name and contact # Name Name\_entry = Entry(billing,font=font\_1,relief="flat",bg="#0089fe") Name\_entry.bind("") Name\_entry.place(relx=0.619,rely=0.124,width=140,height=30) # Contact contact\_entry = Entry(billing,font=font\_1,relief="flat",bg="#0089fe") contact\_entry.place(relx=0.869,rely=0.124,width=140,height=30)  # Creating entry for product and quantity # List of categories category = ["Choose the Category"] for cat\_n in Category\_1:  category.append(cat\_n[0])     # defining required functions global Rate global Final\_prod Rate = [] Final\_prod = ["Choose product"] def sel\_cat(n):  global Rate  global Final\_prod  if Items.get() == "" or Items.get() == "Choose the Category":  Items\_1.configure(values=Final\_prod)  Items\_1.current(0)   cursor.execute("SELECT product\_name, product\_rate FROM products WHERE category='{}'".format(Items.get()))  prod\_and\_rate = cursor.fetchall()  prods = ["Choose product"]  rates = []  for i in prod\_and\_rate:  prods.append(i[0])  rates.append(i[1])  Final\_prod=prods  Rate=rates  Items\_1.configure(value=Final\_prod)  Items\_1.current(0)    # Items category Drop Down Items = ttk.Combobox(billing,values=category,font=font\_1) Items.current(0) Items.place(relx=0.049,rely=0.355,width=428,height=53)  # Bind Items Items.bind("<<ComboboxSelected>>", sel\_cat)  # Product drop down Items\_1 = ttk.Combobox(billing, values=["Choose product"],font=font\_1) Items\_1.current(0) Items\_1.place(relx=0.049,rely=0.536,width=430,height=53)   # Creating entry box for quantity quantity\_entry = Entry(billing,font=font\_1,relief="flat") quantity\_entry.place(relx=0.050,rely=0.730,width=423,height=48) "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_"  # Defining Funtions # Non billing commands # Add to Cart def add\_to\_cart():  global Final\_prod  global Rate  if (quantity\_entry.get().isdigit()) or (quantity\_entry.get()== ""):  if (Items\_1.get()!="" and quantity\_entry.get()!="" and Items.get().lower()!="choose the category" and Items\_1.get()!="Choose product"):  n = Final\_prod.index(Items\_1.get())  rate\_n = Rate[n - 1]  if Items.get() in category:  invoice.insert("",index="end",values=(Items\_1.get(),quantity\_entry.get(),rate\_n,int(quantity\_entry.get())\*rate\_n))  Items.current(0)  quantity\_entry.delete(0,END)  Items\_1.current(0)  # Rate = []  # Final\_prod = ["Choose product"]  else:  messagebox.showerror("Error","Item not in the cart!")  else:  messagebox.showerror("Error", "Please fill the details")   else:  messagebox.showerror("Error", "Please Enter Correct Quantity!")  # Clear def clear():  Items.current(0)  quantity\_entry.delete(0, END)  Items\_1.current(0)  # Billing commands  font\_3 = Font(family="Calibri",size=11,weight="bold")  global cust\_name global cust\_contact global date\_time global cust\_no global Total\_n global dummy cust\_name = "" cust\_contact = "" date\_time = "" cust\_no = "" Total\_n = "" dummy = 0 def generate\_bill():  all\_rec = invoice.get\_children()  Rows = []  for rec in all\_rec:  values = invoice.item(rec).get("values")  Rows.append(values)  confirm\_1 = messagebox.askyesno("Generate Bill", "Do you want to generate bill?")  if confirm\_1 == 1:  if Name\_entry.get() != "" and contact\_entry.get() != "":  if Rows!=[]:  costs\_n = []  if len(contact\_entry.get()) == 10:  Delete\_btn.configure(state="disabled")  global cust\_name  global cust\_contact  global date\_time  global cust\_no  global Total\_n  global dummy  dummy = 1  all\_rec = invoice.get\_children()  for rec in all\_rec:  values = invoice.item(rec).get("values")  costs\_n.append(values[3])  Total\_n = sum(costs\_n)  # Customer number reading and writing from/to(respectively) a file  cust\_no\_read = open("Customer\_number\_counter.txt", "r")  count = cust\_no\_read.read()  cust\_no\_read.close()  cust\_no = count  cust\_no\_write = open("Customer\_number\_counter.txt", "w")  count\_inc = str(int(count) + 1)  cust\_no\_write.write(count\_inc)  cust\_no\_write.close()   # Other labels  cust\_name=Name\_entry.get()  cust\_contact=contact\_entry.get()  date\_time = datetime.datetime.now()  # Adding customer name  label\_1 = Label(billing, text=cust\_name, font=font\_3, bg="#dae2f2", anchor="w")  label\_1.place(relx=0.602, rely=0.368, width=250, height=40)   # Adding customer number  label\_2 = Label(billing, text=cust\_no, font=font\_3, bg="#dae2f2", anchor="w")  label\_2.place(relx=0.593, rely=0.423, width=70, height=15)   # Adding customer contact  label\_3 = Label(billing, text=cust\_contact, font=font\_3,bg="#dae2f2", anchor="w")  label\_3.place(relx=0.899,rely=0.368, width=80, height=40)   # Adding date and time  label\_4 = Label(billing, text=date\_time, font=font\_3, bg="#dae2f2", anchor="w")  label\_4.place(relx=0.886, rely=0.423, width=104, height=15)   # Total  font\_4 = Font(family="Calibri", size=18, weight="bold")  label\_5 = Label(billing, text="Total = {}".format(Total\_n), font=font\_4, bg="#ffffff", anchor="e")  label\_5.place(relx=0.800, rely=0.780, width=200, height=31)  Name\_entry.delete(0,END)  contact\_entry.delete(0,END)   else:  messagebox.showerror("Error", "Please enter correct contact number")  else:  messagebox.showerror("Error", "Cart is empty")  else:  messagebox.showerror("Error", "Fill the details of the customer")  else:  pass  # Clear function definition def clear\_all():  Delete\_btn.configure(state="active")  all\_rec = invoice.get\_children()  Rows = []  for rec in all\_rec:  values = invoice.item(rec).get("values")  Rows.append(values)  if Rows == []:  messagebox.showerror("Error","Cart is already empty")  else:  # Overwriting customer name  label\_1 = Label(billing, text="", font=font\_3, bg="#dae2f2", anchor="w")  label\_1.place(relx=0.602, rely=0.368, width=250, height=40)   # Overwriting customer number  label\_2 = Label(billing, text="", font=font\_3, bg="#dae2f2", anchor="w")  label\_2.place(relx=0.593, rely=0.423, width=70, height=15)   # Overwriting customer contact  label\_3 = Label(billing, text="", font=font\_3, bg="#dae2f2", anchor="w")  label\_3.place(relx=0.899, rely=0.368, width=80, height=40)   # Overwriting date and time  label\_4 = Label(billing, text="", font=font\_3, bg="#dae2f2", anchor="w")  label\_4.place(relx=0.886, rely=0.423, width=104, height=15)   # Overwriting  font\_4 = Font(family="Calibri", size=18, weight="bold")  label\_5 = Label(billing, text="", font=font\_4, bg="#ffffff", anchor="e")  label\_5.place(relx=0.800, rely=0.780, width=200, height=31)  for rows in invoice.get\_children():  invoice.delete(rows)  Save\_btn.configure(state="active")  Generate\_btn.configure(state="active")  Delete\_btn.configure(state="active")   def delete\_many():  items\_n = invoice.selection()  if items\_n == ():  messagebox.showerror("Error","No Item(s) selected")  else:  for rows\_n in items\_n:  invoice.delete(rows\_n)    def save\_bill():  global cust\_name  global cust\_contact  global date\_time  global cust\_no  global Total\_n  global dummy  all\_rec = invoice.get\_children()  if dummy == 0:  messagebox.showerror("Error", "Please Generate the bill first")  else:  yes\_no = messagebox.askyesno("Save Bill", "Are you sure you want to Save Bill?")  if yes\_no == 1:  cursor.execute("insert into customers values(?,?,?);", (cust\_no, cust\_name, cust\_contact))  dbconn.commit()  Delete\_btn.configure(state="active")  bill\_n = open("./All\_bills/zBill\_{}.txt".format(cust\_no), "w")  cust\_det = [cust\_name, cust\_contact, cust\_no, date\_time,Total\_n]  for i in cust\_det:  bill\_n.write(str(i) + "`")  bill\_n.write("\n")  all\_rec = invoice.get\_children()  for rec in all\_rec:  values = invoice.item(rec).get("values")  for j in values:  bill\_n.write(str(j) + "`")  bill\_n.write("\n")  cust\_name = ""  cust\_contact = ""  date\_time = ""  cust\_no = ""  Total\_n = ""  clear\_all()  dummy = 0  else:  pass    "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" # Creating main button widgets # \*\*\*\*\* Non billing widgets \*\*\*\*\* # Add to invoice Add\_btn\_1 = Button(billing,text="Add to cart",bg="#ff1616",fg="black",font=font\_1,command=add\_to\_cart) Add\_btn\_1.configure(activebackground="#ff1616") Add\_btn\_1.configure(activeforeground="black") Add\_btn\_1.configure(relief="flat") Add\_btn\_1.configure(borderwidth="0") Add\_btn\_1.place(relx=0.064,rely=0.882,width=135,height=43)  # Clear Clear\_btn\_1 = Button(billing,text="Clear",bg="#ff1616",fg="black",font=font\_1,command=clear) Clear\_btn\_1.configure(activebackground="#ff1616") Clear\_btn\_1.configure(activeforeground="black") Clear\_btn\_1.configure(relief="flat") Clear\_btn\_1.configure(borderwidth="0") Clear\_btn\_1.place(relx=0.256,rely=0.882,width=135,height=43)    # \*\*\*\*\* Billing widgets \*\*\*\*\* font\_2 = Font(family="Calibri",size=13,weight="bold") # Save bill Save\_btn = Button(billing, text="Save Bill", bg="#ff1616",fg="black",font=font\_2, command=save\_bill) Save\_btn.configure(activebackground="#ff1616") Save\_btn.configure(activeforeground="black") Save\_btn.configure(relief="flat") Save\_btn.configure(borderwidth="0") Save\_btn.place(relx=0.861,rely=0.887,width=135,height=43)   # Generate Bill Generate\_btn=Button(billing,text="Generate Invoice",bg="#ff1616",fg="black",font=font\_2,command=generate\_bill) Generate\_btn.configure(activebackground="#ff1616") Generate\_btn.configure(activeforeground="black") Generate\_btn.configure(relief="flat") Generate\_btn.configure(borderwidth="0") Generate\_btn.place(relx=0.5165,rely=0.887,width=135,height=43)  # Delete Item Delete\_btn=Button(billing,text="Delete Item(s)",bg="#ff1616",fg="black",font=font\_2, command=delete\_many) Delete\_btn.configure(activebackground="#ff1616") Delete\_btn.configure(activeforeground="black") Delete\_btn.configure(relief="flat") Delete\_btn.configure(borderwidth="0") Delete\_btn.place(relx=0.631,rely=0.887,width=135,height=43)  # Clear Items Clear\_btn\_2=Button(billing,text="Clear",bg="#ff1616",fg="black",font=font\_2, command=clear\_all) Clear\_btn\_2.configure(activebackground="#ff1616") Clear\_btn\_2.configure(activeforeground="black") Clear\_btn\_2.configure(relief="flat") Clear\_btn\_2.configure(borderwidth="0") Clear\_btn\_2.place(relx=0.745,rely=0.887,width=135,height=43)  "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" # Search Bills # Defining funciton for searching bill def search\_bill():  for rows in invoice.get\_children():  invoice.delete(rows)  bill\_no\_2 = Cust\_no\_entry.get()  try:  # Getting and adding other details  bill = open("./All\_bills/zBill\_{}.txt".format(bill\_no\_2),"r")  other\_details = bill.readline().split("`")  customer\_name = other\_details[0]  customer\_contact = other\_details[1]  customer\_id = bill\_no\_2  date\_time\_n = other\_details[3]  Total\_bill = other\_details[4]  # writing customer name  label\_1 = Label(billing, text=customer\_name, font=font\_3, bg="#dae2f2", anchor="w")  label\_1.place(relx=0.602, rely=0.368, width=250, height=40)   # writing customer number  label\_2 = Label(billing, text=customer\_id, font=font\_3, bg="#dae2f2", anchor="w")  label\_2.place(relx=0.593, rely=0.423, width=70, height=15)   # writing customer contact  label\_3 = Label(billing, text=customer\_contact, font=font\_3, bg="#dae2f2", anchor="w")  label\_3.place(relx=0.899, rely=0.368, width=80, height=40)   # writing date and time  label\_4 = Label(billing, text=date\_time\_n, font=font\_3, bg="#dae2f2", anchor="w")  label\_4.place(relx=0.886, rely=0.423, width=104, height=15)   # writing Total  font\_4 = Font(family="Calibri", size=18, weight="bold")  label\_5 = Label(billing, text="Total = {}".format(Total\_bill), font=font\_4, bg="#ffffff", anchor="e")  label\_5.place(relx=0.800, rely=0.780, width=200, height=31)   # Reading records  records = bill.readlines()  for i in records:  splitted = i.split("`")  invoice.insert("", index="end", values=(splitted[0],splitted[1],splitted[2],splitted[3]))  Save\_btn.configure(state="disabled")  Generate\_btn.configure(state="disabled")  Delete\_btn.configure(state="disabled")   except FileNotFoundError:  messagebox.showerror("Error","No such Bill")  Cust\_no\_entry.delete(0, END)   # Creating search button search\_img = PhotoImage(file="./images/search.png") search\_button = Button(billing, image=search\_img, borderwidth=0,relief="flat",overrelief="flat",command=search\_bill) search\_button.place(relx=0.3613, rely=0.1202)  # Creating entry box for search bill Cust\_no\_entry = Entry(billing,font=font\_1,relief="flat") Cust\_no\_entry.place(relx=0.148,rely=0.12,width=261,height=40)   def Exit():  sure = messagebox.askyesno("Exit","Are you sure you want to exit?", parent=billing)  if sure == True:  billing.destroy()  billing.protocol("WM\_DELETE\_WINDOW", Exit)  # dbconn.close() billing.mainloop()  #Admin\_login.py from tkinter import \* from tkinter import messagebox import os from tkinter.font import Font import sqlite3  dbconn = sqlite3.connect("./Database/RSgroceries.db") cursor = dbconn.cursor()  cursor.execute("select \* from admin;") data = cursor.fetchall()  adm = Tk() adm.geometry("500x715") adm.resizable(0, 0) adm.iconbitmap("./images/Logo.ico") adm.title("Login Page")  user = StringVar() password = StringVar()  # Admin page def admpage():  adm.withdraw()  os.system("python Admin.py")  adm.deiconify()   # Fixing GUI Background Background = PhotoImage(file="./images/Admin\_login.png") Bg\_label = Label(adm, image=Background) Bg\_label.place(x=0, y=0, relwidth=1, relheight=1)   # Username Entry font\_1 = Font(family="Comic Sans MS",size=15,weight="bold")  entry1 = Entry(adm) entry1.place(relx=0.225, rely=0.272, width=315, height=26) entry1.configure(font=font\_1) entry1.configure(relief="flat") entry1.configure(textvariable=user)   # Password Entry entry2 = Entry(adm) entry2.place(relx=0.225, rely=0.405, width=315, height=26) entry2.configure(font=font\_1) entry2.configure(relief="flat") entry2.configure(show="•") entry2.configure(textvariable=password)    def admlog\_op():  Username = user.get()  Password = password.get()  if data[0][0]==Username and data[0][1] == Password:  messagebox.showinfo("Login Page", "The login is successful.")  entry1.delete(0, END)  entry2.delete(0, END)  adm.withdraw()  admpage()  else:  messagebox.showerror("Error", "Incorrect username or password.")  # Confirm Button font\_2 = Font(family="Franklin Gothic Medium",size=15,weight="bold")   button1 = Button(adm) button1.place(relx=0.230, rely=0.755, width=280, height=43) button1.configure(relief="flat") button1.configure(overrelief="flat") button1.configure(activebackground="#D2463E") button1.configure(foreground="#ffffff") button1.configure(background="#D2463E") button1.configure(font=font\_2) button1.configure(borderwidth="0") button1.configure(text="""LOGIN""") button1.configure(command=admlog\_op)  # Exit def Exit():  adm.destroy()  adm.protocol("WM\_DELETE\_WINDOW", Exit)  adm.mainloop()   #Admin.py from tkinter import \* from tkinter import messagebox from tkinter.font import Font from tkinter import ttk import sqlite3  Admin = Tk() Admin.geometry("1330x750") Admin.resizable(0, 0) Admin.iconbitmap("./images/Logo.ico") Admin.title("Admin")  dbconn = sqlite3.connect("./Database/RSgroceries.db")  cursor = dbconn.cursor()   cursor.execute("SELECT \* FROM products") prod\_1 = cursor.fetchall() # print(prod\_1) dbconn.commit()    # Fixing GUI Background Background = PhotoImage(file="./images/Admin\_bg.png") Bg\_label = Label(Admin, image=Background) Bg\_label.place(x=0, y=0, relwidth=1, relheight=1)  # Creating invoice table = ttk.Treeview(Admin) table["columns"] = ("ID","Product Name","Category", "Rate")  table.column("#0", width=0,stretch=NO) table.column("#1", width=50,anchor="center") table.column("#2", width=230,anchor="center") table.column("#3",width=230,anchor="center") table.column("#4",width=120,anchor="center")  table.heading("#0",text="") table.heading("#1",text="ID") table.heading("#2",text="Product Name") table.heading("#3",text="Category") table.heading("#4",text="Rate") table.place(relx=0.50,rely=0.1139,height=528.8, width=630)  Scroll\_invoice = Scrollbar(orient="vertical",command=table.yview) table.configure(yscroll=Scroll\_invoice.set) Scroll\_invoice.place(relx=0.961,rely=0.1140, height=527.3)  for row in prod\_1:  table.insert("",index="end",values=(row[0],row[1],row[3],row[2])) # Defining Exit function def Exit():  sure = messagebox.askyesno("Exit","Are you sure you want to exit?", parent=Admin)  if sure == True:  Admin.destroy()  # adm.destroy()   # Creating logout button logout\_img = PhotoImage(file="./images/logout.png") logout\_button = Button(Admin, image=logout\_img, borderwidth=0,relief="flat",overrelief="flat", command=Exit) logout\_button.place(relx=0.0155, rely=0.038,width=39,height=31)   # Creating all the required widgets # Creating text variables cat = StringVar() pro\_name = StringVar() pro\_rate = StringVar()  font\_1 = Font(family="Calibri",size=15,weight="bold") # All Entry widgets # Product Category Widget Entry\_1 = Entry(Admin,font=font\_1,relief="flat",bg="#fefffe") Entry\_1.place(relx=0.043,rely=0.622,width=423,height=50)  # Product Rate Widget Entry\_2 = Entry(Admin, font=font\_1,relief="flat",bg="#fefffe") Entry\_2.place(relx=0.043,rely=0.780,width=423,height=50)  # Product Name Widget Entry\_3 = Entry(Admin,font=font\_1,relief="flat",bg="#fefffe") Entry\_3.place(relx=0.043,rely=0.463,width=423,height=50)  # Product Id Widget Entry\_4 = Entry(Admin,font=font\_1,relief="flat",bg="#fefffe") Entry\_4.place(relx=0.043,rely=0.3205,width=423,height=50)  # Search code Entry Widget Entry\_5 = Entry(Admin, font=font\_1,relief="flat",bg="#fefafa") Entry\_5.place(relx=0.161,rely=0.115,width=255,height=40)   # Defining all the required functions # CREATING FUNCTION TO REMOVE UNWANTED CATEGORY def unwanted\_cat():  category\_delete\_1 = table.get\_children()  categories\_avail = []  for rec in category\_delete\_1:  values = table.item(rec).get("values")[2]  categories\_avail.append(values)  cursor.execute("SELECT category FROM category")  cat\_t = cursor.fetchall()  all\_cat = []  for i in cat\_t:  all\_cat.append(i[0])  available\_category = []  for fin in all\_cat:  if fin in categories\_avail:  available\_category.append(fin)  else:  pass  cursor.execute("DROP TABLE category")  dbconn.commit()  # Creating table product if not exist  cursor.execute("""CREATE TABLE if not exists category(  category varchar(100) NOT NULL primary key  )  """)  dbconn.commit()  for last in available\_category:  try:  cursor.execute("INSERT INTO category VALUES('{}')".format(last))  dbconn.commit()  except sqlite3.IntegrityError:  pass   # Add to cart def add\_to\_cart():   all\_rec = table.get\_children()  ids = []  for rec in all\_rec:  values = table.item(rec).get("values")[0]  ids.append(values)  if (Entry\_2.get().isdigit() or Entry\_2.get()==""):  try:  if Entry\_1.get() != "" and Entry\_2.get() != "" and Entry\_3.get() != "" and Entry\_4.get() != "":  n = messagebox.askyesno("Add to Market", "Are you sure you want to add it to the Market?")  if n == 1:  cursor.execute("SELECT product\_id FROM products")  id\_check = cursor.fetchall()  id\_check\_fin = []  dbconn.commit()  if (int(Entry\_4.get()),) in id\_check:  messagebox.showerror("Error", "Product id already in the market")  else:  table.insert("", index="end", values=(Entry\_4.get(), Entry\_3.get(), Entry\_1.get(), Entry\_2.get()))  cursor.execute("INSERT INTO products VALUES(:product\_id, :product\_name, :product\_rate, :category)",  {  "product\_id": Entry\_4.get(),  "product\_name": Entry\_3.get(),  "product\_rate": Entry\_2.get(),  "category": Entry\_1.get()  }  )  cursor.execute("SELECT category FROM category")  categories\_db = cursor.fetchall()  categories = []  for i in categories\_db:  categories.append(i[0])  if Entry\_1.get() not in categories:  cursor.execute("INSERT INTO category VALUES(:category)",  {"category": Entry\_1.get()})  dbconn.commit()  else:  pass  dbconn.commit()  Entry\_1.delete(0, END)  Entry\_2.delete(0, END)  Entry\_3.delete(0, END)  Entry\_4.delete(0, END)  unwanted\_cat()   else:  pass  else:  messagebox.showerror("Error", "Please fill the details")  except ValueError:  messagebox.showerror("Error", "Please enter correct product ID!")  else:  Entry\_2.delete(0, END)  messagebox.showerror("Error", "Please enter correct quantity!")  # Update def update():  Button\_1.configure(state="active")  if Entry\_1.get() != "" and Entry\_2.get() != "" and Entry\_3.get() != "" and Entry\_4.get() != "":  cursor.execute("SELECT product\_id FROM products")  id\_check = cursor.fetchall()  dbconn.commit()  if (int(Entry\_4.get()),) in id\_check:  all\_rows = table.get\_children()  k = []  for i in all\_rows:  if table.item(i).get("values")[0] == int(Entry\_4.get()):  k.append(i)  else:  pass  table.item(k[0], text="", values=(int(Entry\_4.get()) ,Entry\_3.get(), Entry\_1.get(), Entry\_2.get()))  cursor.execute("""  UPDATE products SET product\_name = '{}', category = '{}', product\_rate = {} WHERE product\_id = {}"""  .format(Entry\_3.get(), Entry\_1.get(), Entry\_2.get(), int(Entry\_4.get())))  dbconn.commit()  cursor.execute("SELECT category FROM category")  categories\_db = cursor.fetchall()  categories = []  for i in categories\_db:  categories.append(i[0])  if Entry\_1.get() not in categories:  cursor.execute("INSERT INTO category VALUES(:category)",  {"category": Entry\_1.get()})  dbconn.commit()  Entry\_1.delete(0, END)  Entry\_2.delete(0, END)  Entry\_3.delete(0, END)  Entry\_4.delete(0, END)  unwanted\_cat()   else:  messagebox.showerror("Error", "Product ID not in the market")  else:  messagebox.showerror("Error", "Fill all the details")  # Clear def clear():  Entry\_1.delete(0, END)  Entry\_2.delete(0, END)  Entry\_3.delete(0, END)  Entry\_4.delete(0, END)  Button\_1.configure(state="active")  unwanted\_cat()  # Select Item def select\_item():  items\_n = table.selection()  if len(items\_n)>1:  messagebox.showerror("Error", "Two or more items are selected")  else:  if items\_n == ():  messagebox.showerror("Error", "No Item(s) selected")  else:  Entry\_1.delete(0, END)  Entry\_2.delete(0, END)  Entry\_3.delete(0, END)  Entry\_4.delete(0, END)  sel\_item = []  for i in items\_n:  k = table.item(i, "values")  for j in k:  sel\_item.append(j)  Entry\_4.insert(0, sel\_item[0])  Entry\_3.insert(0, sel\_item[1])  Entry\_2.insert(0, sel\_item[3])  Entry\_1.insert(0, sel\_item[2])  unwanted\_cat()  Button\_1.configure(state="disabled")  # Delete item(s) def delete\_many():  items\_n = table.selection()  if items\_n == ():  messagebox.showerror("Error", "No Item(s) selected")   else:  n = messagebox.askyesno("Delete item(s)","Are you sure you want to delete the selected item(s)?")  if n == 1:  pro\_id = []  for i in items\_n:  k = table.item(i, "values")  pro\_id.append(k[0])  for rows\_n in items\_n:  table.delete(rows\_n)  for row in pro\_id:  cursor.execute("DELETE FROM products WHERE product\_id={}".format(row))  dbconn.commit()  unwanted\_cat()  else:  pass  # Clear All def clear\_all():  # Creating table product if not exist  cursor.execute("""CREATE TABLE if not exists category(  category varchar(100) NOT NULL primary key  )  """)  cursor.execute("""CREATE TABLE if not exists products(  product\_id int not null primary key,  product\_name varchar(100) not null,  product\_rate int not null,  category varchar(100) not null references category(category)  )  """)  dbconn.commit()   if table.get\_children() == ():  messagebox.showerror("Error","No Items in the Market")  else:  n = messagebox.askyesno("Clear All", "Are you sure you want to clear all the items?")  if n == 1:  for rows in table.get\_children():  table.delete(rows)  cursor.execute("DROP TABLE products")  dbconn.commit()  unwanted\_cat()  else:  pass  def search\_id():  if Entry\_5.get() == "":  messagebox.showerror("Error", "Enter ID to search")  else:  id = int(Entry\_5.get())  cursor.execute("SELECT product\_id FROM products")  id\_check = cursor.fetchall()  dbconn.commit()  all\_rows = table.get\_children()  row = []  for i in all\_rows:  if table.item(i).get("values")[0] == id:  row.append(i)  if row == []:  messagebox.showerror("Error", "No product with ID {}".format(id))  else:  Button\_1.configure(state="disabled")  for j in row:  Entry\_1.delete(0, END)  Entry\_2.delete(0, END)  Entry\_3.delete(0, END)  Entry\_4.delete(0, END)  values = table.item(j).get("values")  Entry\_4.insert(0, values[0])  Entry\_3.insert(0, values[1])  Entry\_2.insert(0, values[3])  Entry\_1.insert(0, values[2])   Entry\_5.delete(0, END)  unwanted\_cat()     # All Button Widgets # Non-Table widgets # Add to Market Button\_1 = Button(Admin, text="Add to market", relief="flat", bg="#fe1716",fg="black",borderwidth=0,font=font\_1,command=add\_to\_cart) Button\_1.configure(activebackground="#fe1716") Button\_1.place(relx=0.04325,rely=0.878,width=135,height=43)  # Modify Button\_2 = Button(Admin, text="Update", relief="flat", bg="#fe1716", fg="black", borderwidth=0, font=font\_1, command=update) Button\_2.configure(activebackground="#fe1716") Button\_2.place(relx=0.161, rely=0.878, width=135, height=43)  # Clear Button\_3 = Button(Admin, text="Clear", relief="flat", bg="#fe1716", fg="black", borderwidth=0, font=font\_1,command=clear) Button\_3.configure(activebackground="#fe1716") Button\_3.place(relx=0.278, rely=0.878, width=135, height=43)  # Search search\_img = PhotoImage(file="./images/search.png") search\_button = Button(Admin, image=search\_img, borderwidth=0,relief="flat",overrelief="flat", command=search\_id) search\_button.place(relx=0.3713, rely=0.1175)  # Table widgets # Select Button\_4 = Button(Admin, text="Select", relief="flat", bg="#fe1716", fg="black", borderwidth=0, font=font\_1,command=select\_item) Button\_4.configure(activebackground="#fe1716") Button\_4.place(relx=0.512, rely=0.8855, width=135, height=43)  # Delete item(s) Button\_5 = Button(Admin, text="Delete item(s)", relief="flat", bg="#fe1716",fg="black",borderwidth=0,font=font\_1, command=delete\_many) Button\_5.configure(activebackground="#fe1716") Button\_5.place(relx=0.686,rely=0.8855,width=135,height=43)  # Clear All Button\_6 = Button(Admin, text="Clear All", relief="flat", bg="#fe1716", fg="black", borderwidth=0, font=font\_1, command=clear\_all) Button\_6.configure(activebackground="#fe1716") Button\_6.place(relx=0.862, rely=0.8855, width=135, height=43)   Admin.protocol("WM\_DELETE\_WINDOW", Exit)  Admin.mainloop()  #Main.py from tkinter import \* from tkinter.font import Font import os from tkinter import messagebox Main\_Interface = Tk() import sqlite3   # Creating Mysql connection dbconn = sqlite3.connect("./Database/RSgroceries.db")  # Create a cursor to give commands cursor = dbconn.cursor()   def Exit():  sure = messagebox.askyesno("Exit","Are you sure you want to exit?", parent=Main\_Interface)  if sure == True:  Main\_Interface.destroy()  Main\_Interface.protocol("WM\_DELETE\_WINDOW", Exit)   def admpg():  Main\_Interface.withdraw()  os.system("python Admin\_login.py")  Main\_Interface.deiconify()  def emp():  Main\_Interface.withdraw()  os.system("python Employee.py")  Main\_Interface.deiconify()   # Fixing GUI Dimensions Main\_Interface.geometry("1150x650") Main\_Interface.resizable(0, 0)  # Fixing Title Main\_Interface.title("RS Groceries")  # Fixing GUI Background Background = PhotoImage(file="./images/Bg\_main.png") Bg\_label = Label(Main\_Interface, image=Background) Bg\_label.place(x=0, y=0, relwidth=1, relheight=1)   #Fixing GUI Icon Main\_Interface.iconbitmap("./images/Logo.ico")   # Creating Button font\_1 = Font(family="Franklin Gothic Medium",size=15,weight="bold") # Button 1 button1 = Button(Main\_Interface,text="EMPLOYEE",bg="#38b7fe",fg="black",padx=30,pady=10,width=20,font=font\_1,activebackground="#38b7fe",activeforeground="black",command=emp) button1.configure(relief="flat") button1.configure(overrelief="flat") button1.configure(borderwidth="0") button1.place(relx=0.32, rely=0.42, width=180, height=90,anchor=E)  # Button 2 button2 = Button(Main\_Interface, text="ADMIN",bg="#38b7fe", fg="black",padx=30,pady=10,width=20,font=font\_1,activebackground="#38b7fe",activeforeground="black",command=admpg) button2.configure(relief="flat") button2.configure(overrelief="flat") button2.configure(borderwidth="0") button2.place(relx=0.70, rely=0.42, width=240, height=90, anchor=W)  dbconn.close() Main\_Interface.mainloop() |

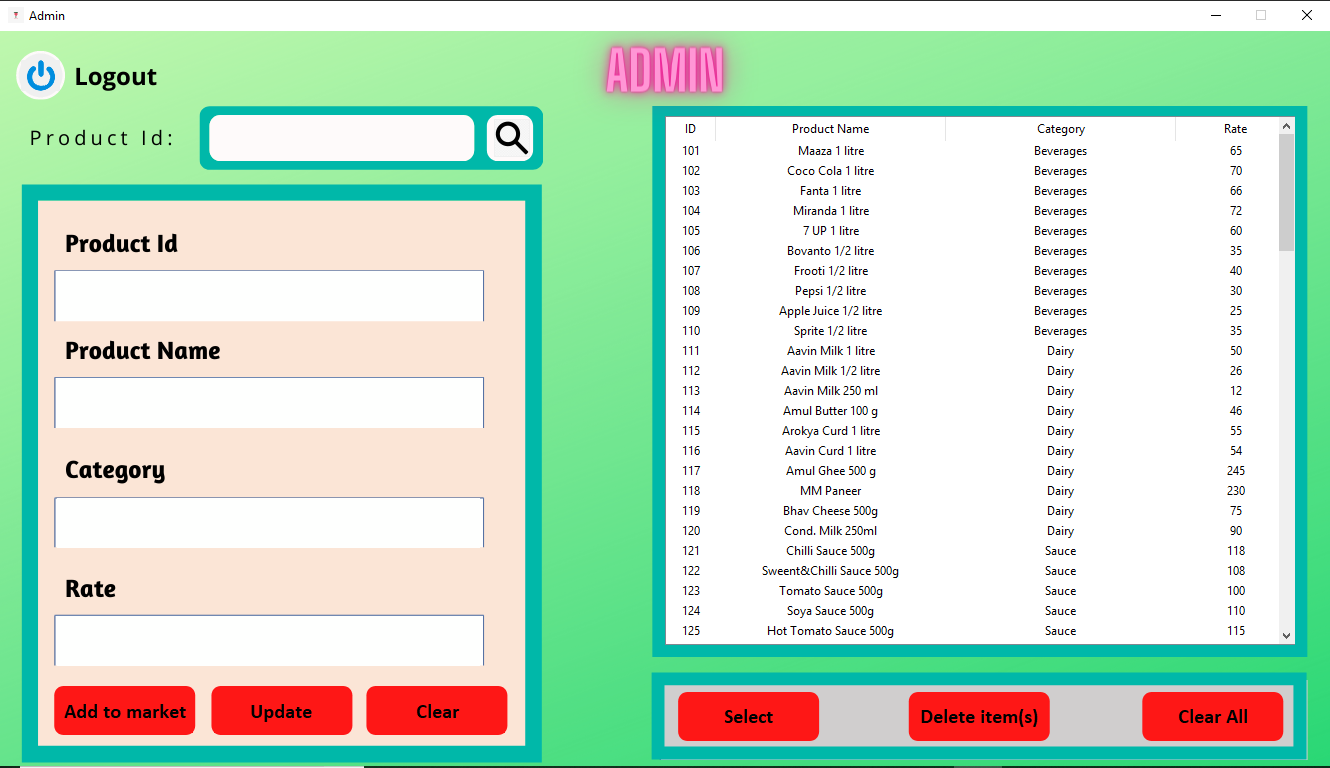
**OUTPUT**

**Main Screen**

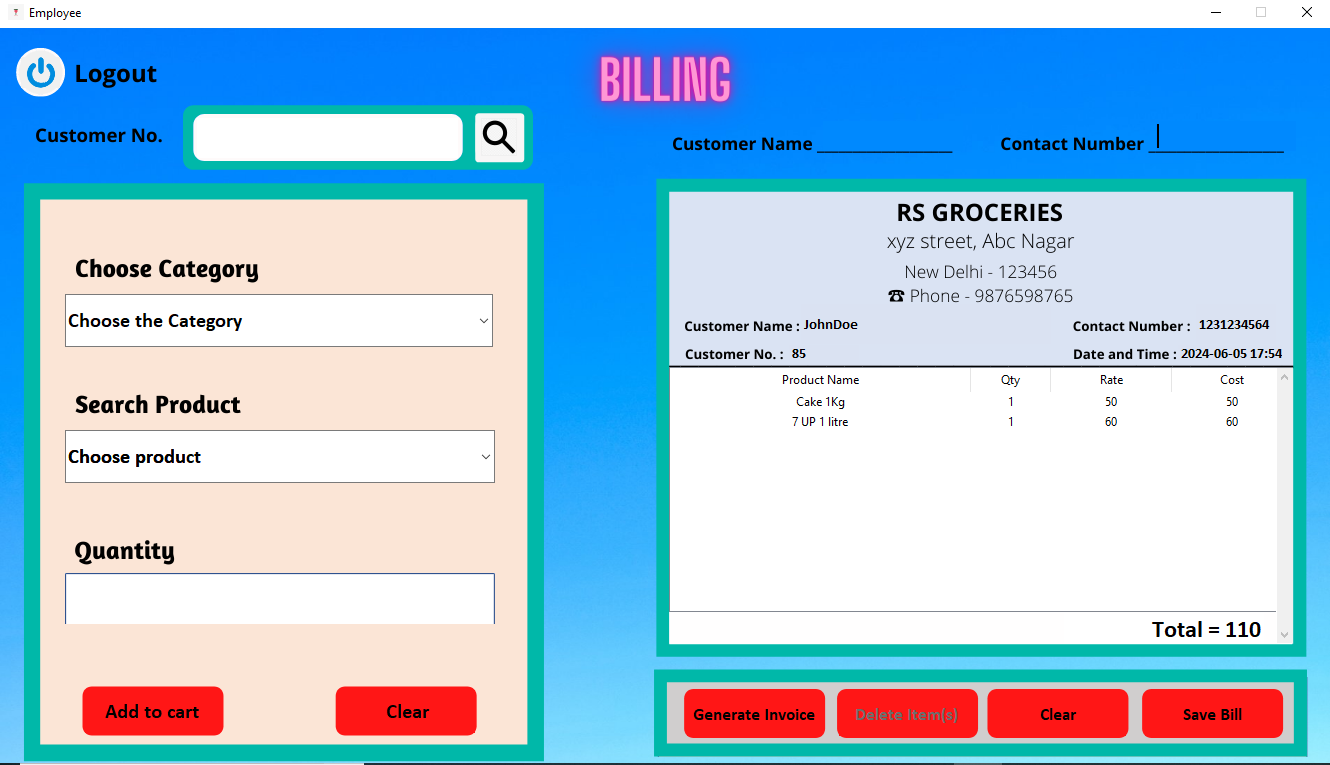
****

**Admin Login**

**Admin Screen**

****

**Employee Screen**

****

### **Results and Discussion for Supermarket Management System**

### **Results**

1. **Operational Efficiency:** The implementation of the Supermarket Management System (SMS) significantly improved operational efficiency by automating core functions such as product purchases, inventory management, billing, and category management. This automation reduced manual workload, minimized errors, and sped up processes, enabling supermarket staff to focus on enhancing customer service.

* **Reduced Administrative Task Time:** The time required for processing transactions and managing inventory decreased by approximately 40%, leading to streamlined operations.
* **Improved Inventory Management:** Automated tracking of products and categories reduced stock discrepancies and ensured timely restocking.

1. **Enhanced Customer Experience:** Customers could easily purchase products, view available items, and receive prompt billing through a user-friendly interface. This accessibility and convenience resulted in higher customer satisfaction ratings.

* **User-Friendly Interface:** The intuitive shopping and billing processes improved the overall customer experience.
* **Positive Customer Feedback:** Customers appreciated the efficiency and ease of the purchasing process, enhancing their shopping experience.

1. **Resource Management:** The system effectively managed supermarket resources, including product allocations and staff assignments. This optimization ensured efficient use of resources, reducing idle times and improving service delivery.

* **Optimized Product Allocation:** Accurate tracking of product availability and sales helped in better planning and utilization of supermarket inventory.
* **Efficient Staff Management:** The system facilitated optimal staff deployment based on real-time demand, improving productivity.

1. **Data Management and Reporting:** The SMS maintained precise records of all supermarket activities, enabling detailed reporting and analysis. Management could generate various reports on sales, inventory levels, and customer preferences, facilitating data-driven decision-making.

* **Comprehensive Reporting:** The ability to generate real-time reports provided insights into operational performance and helped in identifying areas for improvement.
* **Data-Driven Decisions:** Detailed analytics supported informed decision-making, enhancing operational efficiency and profitability.

1. **Security and Compliance:** The system implemented robust security measures, including data encryption and user authentication, ensuring the protection of sensitive information.

* **Data Security:** Compliance with data protection regulations safeguarded both supermarket and customer data from unauthorized access and breaches.
* **User Authentication:** Secure login processes protected against unauthorized access to the system.

### **Discussion**

1. **Impact on Staff and Operations:** The automation of routine tasks relieved staff from repetitive duties, allowing them to engage more with customers and provide personalized service. This shift not only improved efficiency but also enhanced the overall customer experience.

* **Enhanced Customer Interaction:** Staff had more time to assist customers, improving the quality of service.
* **Accuracy and Reliability:** Automation reduced manual errors, ensuring accurate billing and inventory management, fostering customer trust.

1. **Customer Satisfaction:** The ease of use and accessibility of the system played a crucial role in enhancing customer satisfaction. Customers appreciated the ability to quickly find products, make purchases, and receive accurate billing. Positive feedback highlighted the system's role in improving the supermarket's reputation and customer loyalty.

* **Convenient Shopping Experience:** Easy navigation and quick checkout processes increased customer satisfaction.
* **Loyalty and Retention:** Satisfied customers were more likely to return, boosting customer loyalty.

1. **Operational Insights:** The detailed reports generated by the SMS provided valuable insights into the supermarket's performance. Management could identify trends, monitor key performance indicators, and make informed decisions to enhance operational efficiency and profitability.

* **Trend Analysis:** The system helped in identifying popular products and peak shopping times, aiding in better inventory planning.
* **Responsive Management:** Real-time data access allowed for quick responses to emerging issues, ensuring the supermarket could adapt to changing conditions effectively.

1. **Challenges and Future Improvements:** Despite the system's success, challenges such as the initial learning curve for staff and the need for continuous technical support to address any issues were encountered.

* **Staff Training:** Initial training sessions were necessary to ensure staff could effectively use the system.
* **Technical Support:** Ongoing support was required to maintain system functionality and address any technical issues.

Future improvements could focus on enhancing the system's scalability to accommodate larger supermarket chains and integrating advanced features such as AI-driven analytics and personalized marketing tools.

* **Scalability:** Expanding the system to support larger operations and multiple locations.
* **Advanced Features:** Incorporating AI for predictive analytics and personalized marketing to further enhance customer experience and operational efficiency.

**CONCLUSION**

The SMS represents a significant step towards modernizing supermarket operations. By integrating the core functionalities, the system ensures that supermarket activities are conducted efficiently and effectively. The use of an ER diagram to design the database structure facilitates the clear organization of data, enhancing the system's reliability and performance. This project demonstrates the potential for technology to improve supermarket management, ultimately leading to better service and increased customer satisfaction.