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.

1.1 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.

1.2 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.

1.3MODULES

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.

2. 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.

3. 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.

4. 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:

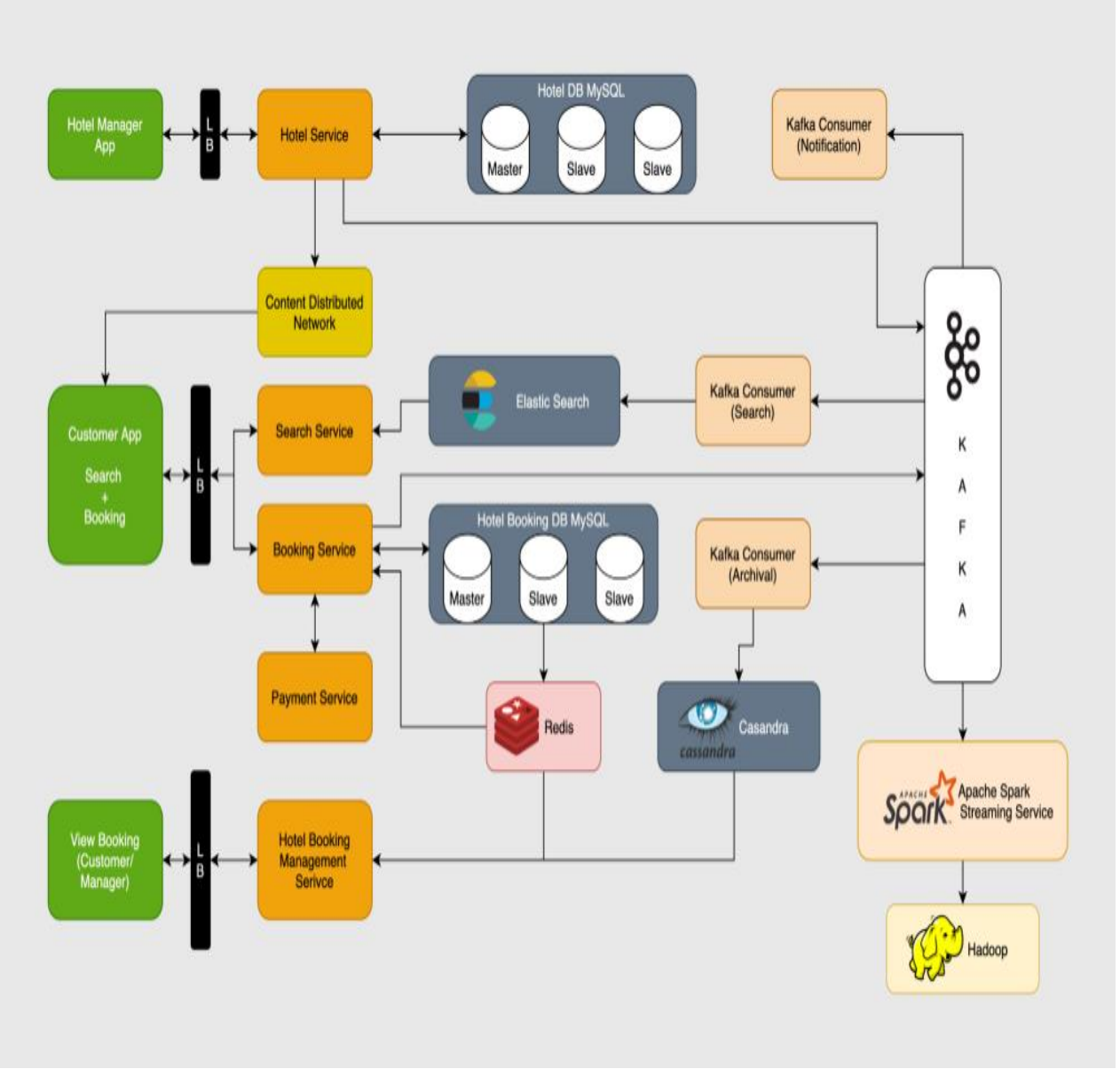
Firewall: UFW (for Linux), Windows Firewall

Antivirus: Bitdefender, Norton, or equivalent

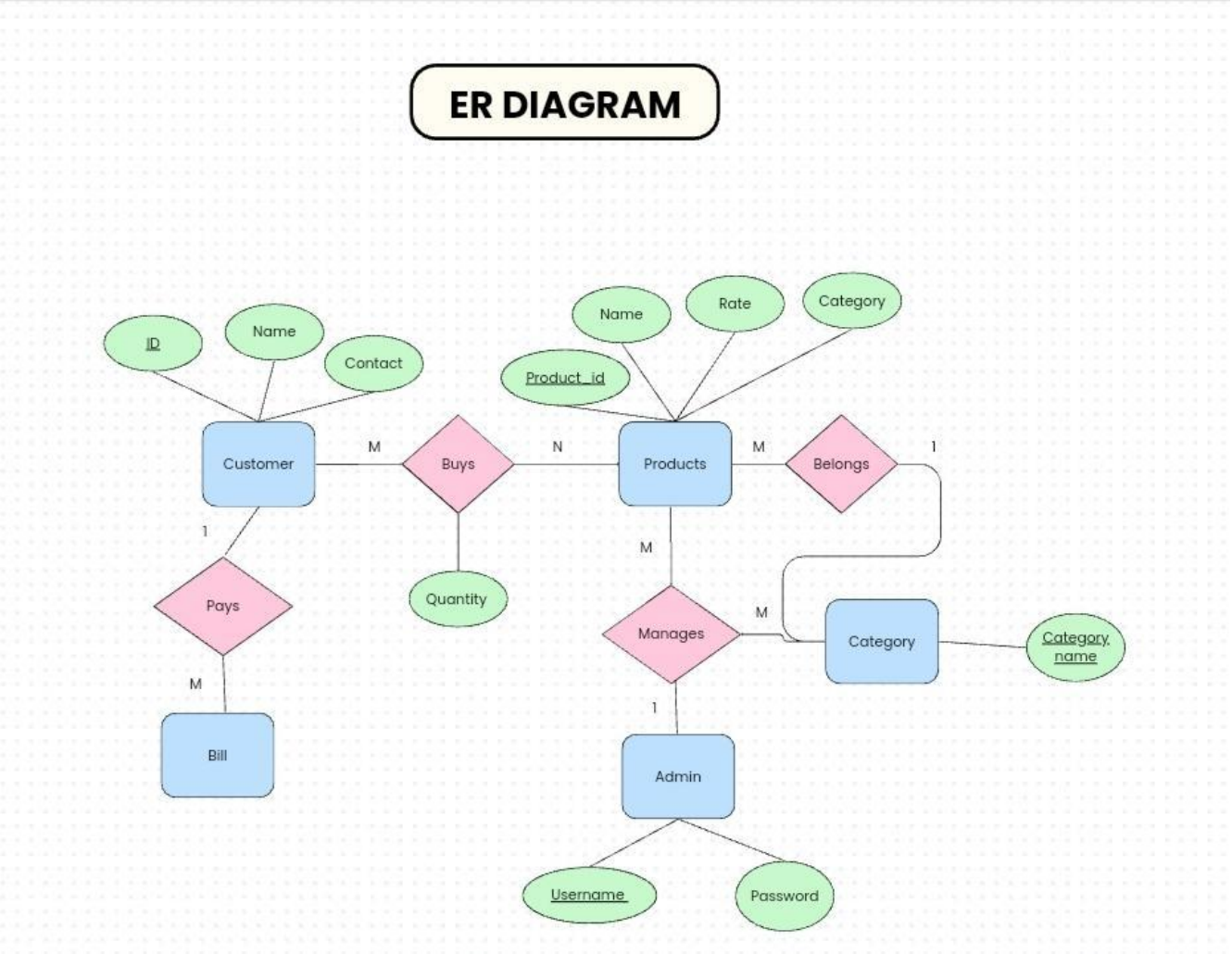
Encryption: SSL/TLS certificates for secure communication

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



3.4 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(),r
ate_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=cl
ear)
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")
```

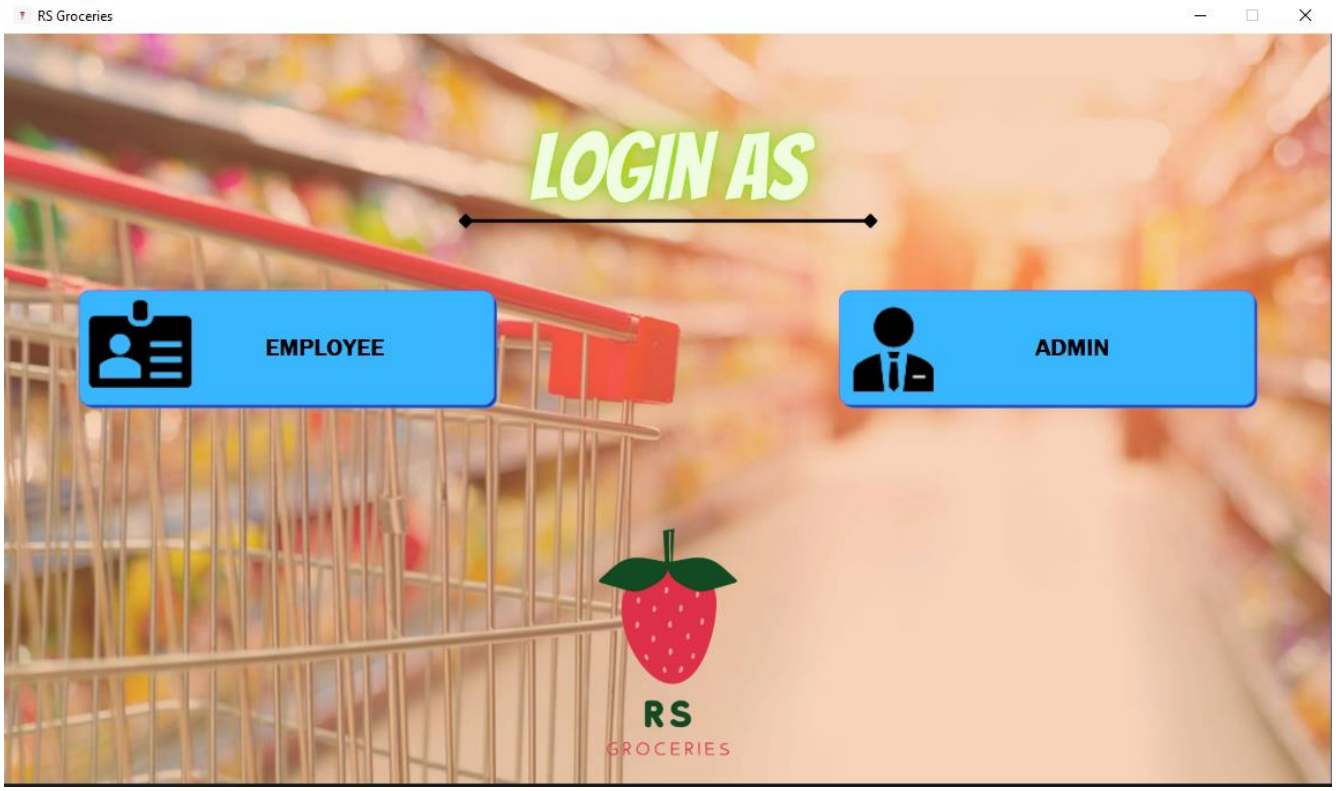
```
Button_1.place(relx=0.04325,rely=0.878,width=135,height=43)

# 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="blac
k",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()
```

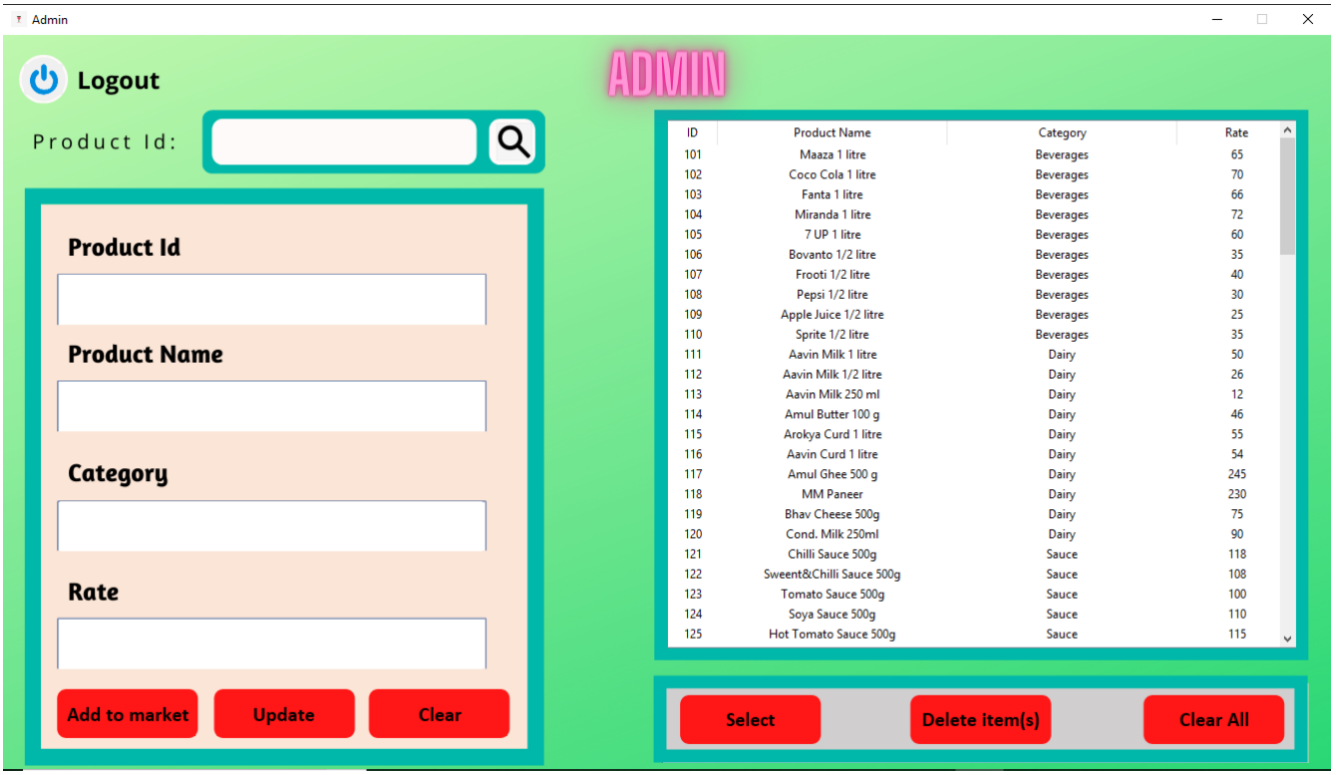
OUTPUT

Main Screen

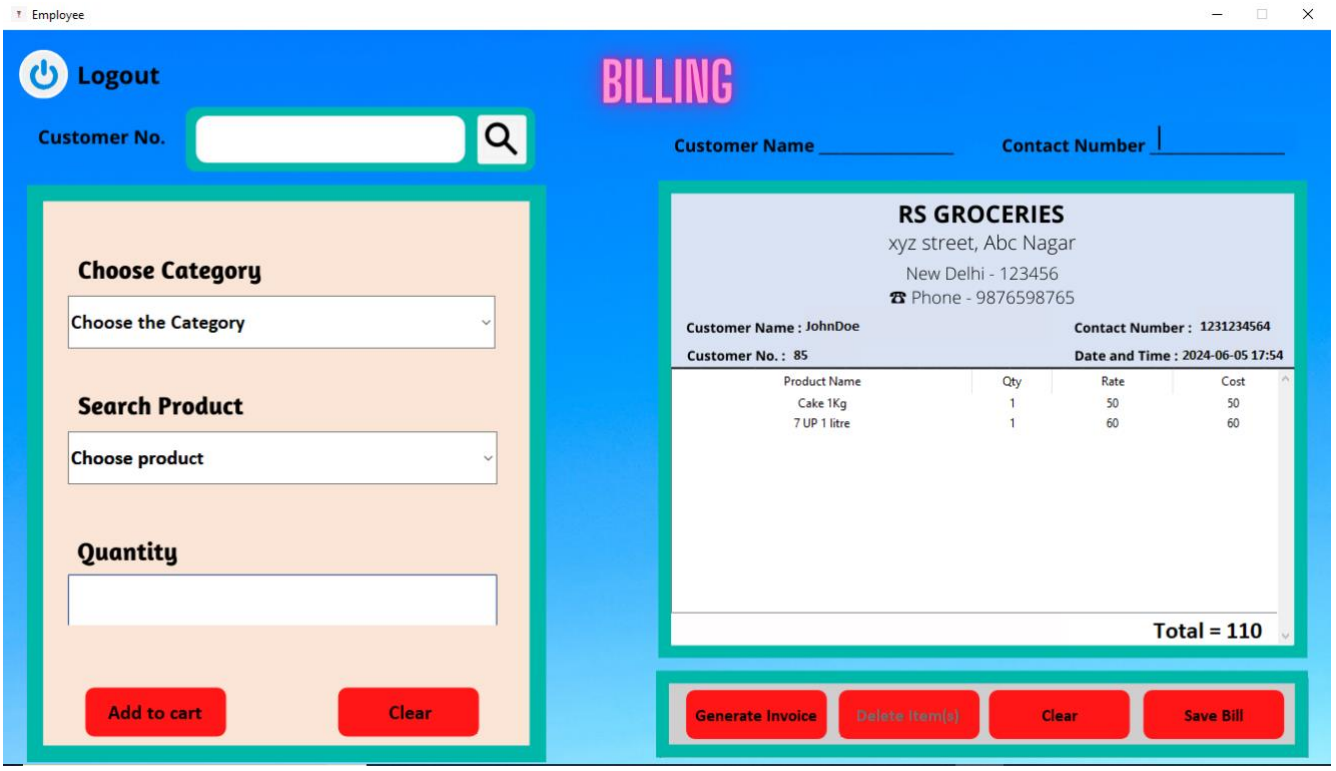


Admin Login

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.

2. **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.

3. **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.

4. **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.

5. **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.

2. **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.

3. **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.

4. **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.