**SUPERMARKET MANAGEMENT SYSTEM**

**A MINI PROJECT REPORT**

**Submitted by**

**SANTHOSH KUMAR R 220701253**

**SHARAN KUMAR D 220701264**

# 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 “**SUPERMARKET MANAGEMENT SYSTEM**” is the bonafide work of **“SANTHOSH KUMAR R(220701253)**, **SHARAN KUMAR(220701264)”**

who carried out the project work under my supervision.

**Submitted for the Practical Examination held on**

|  |  |
| --- | --- |
| **SIGNATURE** | **SIGNATURE** |
| **Dr.R.SABITHA** | **Ms.V.JANANEE** |

**Professor and II Year Academic Head Assistant Professor (SG),**

|  |  |
| --- | --- |
| **Computer Science and Engineering,** | **Computer Science and Engineering,** |
| **Rajalakshmi Engineering College** | **Rajalakshmi Engineering College,** |
| **(Autonomous),** | **(Autonomous),** |
| **Thandalam, Chennai - 602 105** | **Thandalam, Chennai - 602 105** |
| **INTERNAL EXAMINER** | **EXTERNAL EXAMINER** |

# ABSTRACT

The Supermarket Management System is a desktop application developed using Python and SQLite with a graphical user interface built using Tkinter. This application facilitates efficient management of supermarket operations, including inventory management, customer management, sales processing, and report generation. Users can perform essential operations such as adding, updating, and deleting inventory items, recording customer details, processing sales transactions, and viewing detailed reports of inventory, customers, and sales history. Each item in the inventory has a unique identification number, and customers can purchase items by entering the item ID and quantity. The system automatically updates the inventory quantities based on sales and ensures that accurate and up-to-date records are maintained. The application also provides functionality to view the purchase history of individual customers. This system aims to streamline supermarket management processes, improve data accuracy, and enhance overall operational efficiency.

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **S.No** | **CONTENT** | **Pg.No** |
| **1** | **INTRODUCTION** |  |
|  | 1.1 INTRODUCTION |  |
|  | 1.2 OBJECTIVES |  |
|  | 1.3 MODULES |  |
| **2** | **SURVEY OF TECHNOLOGIES** |  |
|  | 2.1 SOFTWARE DESCRIPTION |  |
|  | 2.2 LANGUAGES |  |
|  | 2.2.1 SQL |  |
|  | 2.2.2 PYTHON |  |
| **3** | **REQUIREMENTS AND ANALYSIS** |  |
|  | 3.1 REQUIREMENT SPECIFICATION |  |
|  | 3.2 HARDWARE AND SOFTWARE REQUIREMENTS |  |
|  | 3.3 ARCHITECTURE DIAGRAM |  |
|  | 3.4 ER DIAGRAM |  |
|  | 3.5 NORMALIZATION |  |
| **4** | **PROGRAM CODE** |  |
| **5** | **RESULTS AND DISCUSSION** |  |
| **6** | **CONCLUSION** |  |
| **7** | **REFERENCES** |  |

# INTRODUCTION

**1.1 Introduction**

The Supermarket Management System is designed to streamline the management processes within a supermarket. It integrates functionalities such as inventory management, customer management, sales processing, and report generation into a single, user-friendly application. The system utilizes a graphical user interface (GUI) built with Tkinter and leverages SQLite for data storage and retrieval.

**1.2 Objectives**

To provide a user-friendly interface for managing supermarket operations.

To maintain accurate and up-to-date inventory records.

To facilitate efficient customer management and sales processing.

To generate detailed reports on inventory, customers, and sales.

To ensure data integrity and ease of data retrieval.

**1.3 Modules**

Inventory Management: Adding, updating, and deleting items.

Customer Management: Adding new customers and viewing customer information.

Sales Processing: Recording sales transactions and updating inventory accordingly.

Reporting: Viewing inventory, customer lists, sales history, and customer purchase history.

**2. SURVEY OF TECHNOLOGIES**

**2.1 Software Description**

The system is built using Python, a versatile and widely-used programming language, and SQLite, a lightweight, disk-based database that doesn't require a separate server process. Tkinter is used for creating the graphical user interface, providing a simple yet powerful way to develop desktop applications.

**2.2 Languages**

**2.2.1 SQL**

SQL (Structured Query Language) is used to interact with the SQLite database. It allows for the creation, manipulation, and querying of data tables within the database. Key SQL operations in this project include creating tables, inserting records, updating records, and querying data.

**2.2.2 Python**

Python is the primary programming language used for this project. It is known for its readability and ease of use. Python, combined with its standard libraries and additional modules like Tkinter for GUI and sqlite3 for database management, provides a robust platform for developing the Supermarket Management System.

**3. REQUIREMENTS AND ANALYSIS**

**3.1 Requirement Specification**

**Functional Requirements:**

The system shall allow the user to add, update, and delete inventory items.

The system shall allow the user to add new customers.

The system shall allow the user to record sales transactions.

The system shall update inventory quantities based on sales.

The system shall provide views for inventory, customers, and sales history.

The system shall provide a view for the purchase history of a specific customer.

**Non-Functional Requirements:**

The system shall have a user-friendly interface.

The system shall ensure data accuracy and integrity.

The system shall be reliable and handle concurrent access appropriately.

The system shall perform operations efficiently.

**3.2 Hardware and Software Requirement**

**Hardware Requirements:**

A personal computer or laptop with at least 2 GB RAM and 1 GHz processor.

Sufficient storage space for the database and application files.

**Software Requirements:**

Operating System: Windows, macOS, or Linux.

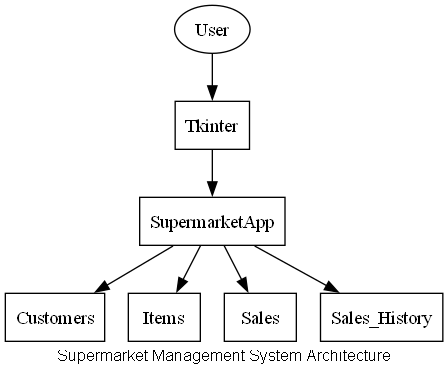
Python 3.x installed on the system.

Tkinter library for Python (usually included with standard Python installation).

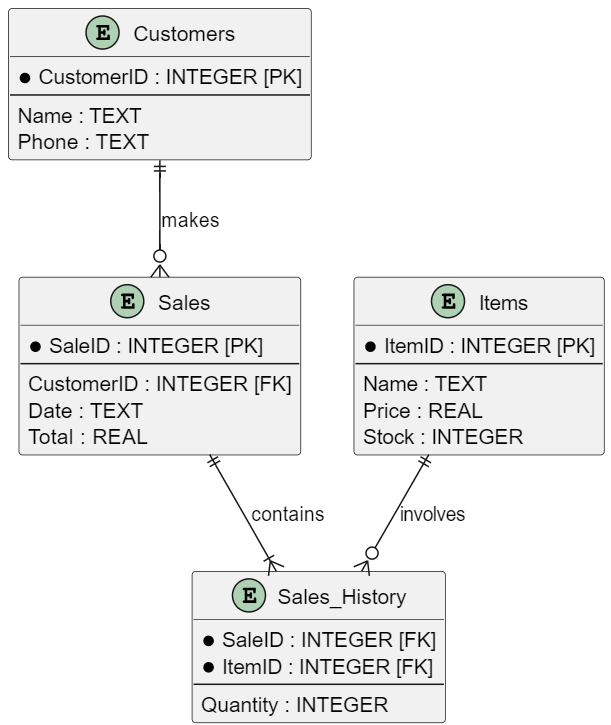
SQLite3 library for Python (also typically included with standard Python installation).

By addressing these requirements and utilizing the specified technologies, the Supermarket Management System aims to enhance the efficiency and accuracy of supermarket operations, providing a comprehensive solution for inventory, customer, and sales management.

**3.3ARCHITECTURE DIAGRAM**

****

**3.4 ER DIAGRAM**

****

**PROGRAM CODES:**

* **PYTHON WITH SQLINTE FOR THE CREATION OF CUSTOMER,SALES AND INVENTORY TABLES:**

import sqlite3

from datetime import datetime

def initialize\_db():

conn = sqlite3.connect('supermarket.db')

cursor = conn.cursor()

# Drop existing tables (for testing purposes)

cursor.execute('DROP TABLE IF EXISTS inventory')

cursor.execute('DROP TABLE IF EXISTS customers')

cursor.execute('DROP TABLE IF EXISTS sales')

# Create tables

cursor.execute('''

CREATE TABLE IF NOT EXISTS inventory (

item\_id INTEGER PRIMARY KEY AUTOINCREMENT,

item\_name TEXT NOT NULL,

quantity INTEGER NOT NULL,

price REAL NOT NULL

)

''')

cursor.execute('''

CREATE TABLE IF NOT EXISTS customers (

customer\_id INTEGER PRIMARY KEY AUTOINCREMENT,

customer\_name TEXT NOT NULL,

customer\_phone TEXT NOT NULL

)

''')

cursor.execute('''

CREATE TABLE IF NOT EXISTS sales (

sale\_id INTEGER PRIMARY KEY AUTOINCREMENT,

customer\_id INTEGER,

item\_id INTEGER,

quantity INTEGER NOT NULL,

sale\_date TEXT NOT NULL,

FOREIGN KEY(customer\_id) REFERENCES customers(customer\_id),

FOREIGN KEY(item\_id) REFERENCES inventory(item\_id)

)

''')

conn.commit()

conn.close()

def execute\_db\_query(query, params=()):

conn = sqlite3.connect('supermarket.db')

cursor = conn.cursor()

cursor.execute(query, params)

conn.commit()

conn.close()

def fetch\_db\_query(query, params=()):

conn = sqlite3.connect('supermarket.db')

cursor = conn.cursor()

cursor.execute(query, params)

records = cursor.fetchall()

conn.close()

return records

* **PYTHON CODE TO HANDLE GUI INTERFACE AND TASKS:**

import tkinter as tk

from tkinter import messagebox, simpledialog

from database import initialize\_db, execute\_db\_query, fetch\_db\_query

class SupermarketApp:

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

self.root = root

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

self.root.geometry("600x400")

self.create\_widgets()

def create\_widgets(self):

# Create Frames

self.frame\_title = tk.Frame(self.root, pady=20)

self.frame\_title.pack()

self.frame\_buttons\_top = tk.Frame(self.root, pady=10)

self.frame\_buttons\_top.pack()

self.frame\_buttons\_middle = tk.Frame(self.root, pady=10)

self.frame\_buttons\_middle.pack()

self.frame\_buttons\_bottom = tk.Frame(self.root, pady=10)

self.frame\_buttons\_bottom.pack()

# Title Label

self.label\_title = tk.Label(self.frame\_title, text="Supermarket Management System", font=("Helvetica", 16, "bold"))

self.label\_title.pack()

# Create Buttons with Styling

button\_font = ("Helvetica", 12)

button\_bg = "#4CAF50"

button\_fg = "#FFFFFF"

button\_padx = 15

button\_pady = 10

self.btn\_add\_item = tk.Button(self.frame\_buttons\_top, text="Add Item", font=button\_font, bg=button\_bg, fg=button\_fg, command=self.add\_item, padx=button\_padx, pady=button\_pady)

self.btn\_update\_item = tk.Button(self.frame\_buttons\_top, text="Update Item", font=button\_font, bg=button\_bg, fg=button\_fg, command=self.update\_item, padx=button\_padx, pady=button\_pady)

self.btn\_delete\_item = tk.Button(self.frame\_buttons\_top, text="Delete Item", font=button\_font, bg=button\_bg, fg=button\_fg, command=self.delete\_item, padx=button\_padx, pady=button\_pady)

self.btn\_add\_customer = tk.Button(self.frame\_buttons\_middle, text="Add Customer", font=button\_font, bg=button\_bg, fg=button\_fg, command=self.add\_customer, padx=button\_padx, pady=button\_pady)

self.btn\_make\_sale = tk.Button(self.frame\_buttons\_middle, text="Make Sale", font=button\_font, bg=button\_bg, fg=button\_fg, command=self.make\_sale, padx=button\_padx, pady=button\_pady)

self.btn\_view\_inventory = tk.Button(self.frame\_buttons\_bottom, text="View Inventory", font=button\_font, bg=button\_bg, fg=button\_fg, command=self.view\_inventory, padx=button\_padx, pady=button\_pady)

self.btn\_view\_customers = tk.Button(self.frame\_buttons\_bottom, text="View Customers", font=button\_font, bg=button\_bg, fg=button\_fg, command=self.view\_customers, padx=button\_padx, pady=button\_pady)

self.btn\_view\_sales = tk.Button(self.frame\_buttons\_bottom, text="View Sales History", font=button\_font, bg=button\_bg, fg=button\_fg, command=self.view\_sales, padx=button\_padx, pady=button\_pady)

self.btn\_customer\_purchase\_history = tk.Button(self.frame\_buttons\_bottom, text="Customer Purchase History", font=button\_font, bg=button\_bg, fg=button\_fg, command=self.view\_customer\_purchase\_history, padx=button\_padx, pady=button\_pady)

# Layout the buttons

self.btn\_add\_item.grid(row=0, column=0, padx=10, pady=10)

self.btn\_update\_item.grid(row=0, column=1, padx=10, pady=10)

self.btn\_delete\_item.grid(row=0, column=2, padx=10, pady=10)

self.btn\_add\_customer.grid(row=1, column=0, padx=10, pady=10)

self.btn\_make\_sale.grid(row=1, column=1, padx=10, pady=10)

self.btn\_view\_inventory.grid(row=2, column=0, padx=10, pady=10)

self.btn\_view\_customers.grid(row=2, column=1, padx=10, pady=10)

self.btn\_view\_sales.grid(row=2, column=2, padx=10, pady=10)

self.btn\_customer\_purchase\_history.grid(row=3, column=1, padx=10, pady=10)

def add\_item(self):

name = simpledialog.askstring("Input", "Enter item name:")

quantity = simpledialog.askinteger("Input", "Enter item quantity:")

price = simpledialog.askfloat("Input", "Enter item price:")

if name and quantity is not None and price is not None:

execute\_db\_query("INSERT INTO inventory (item\_name, quantity, price) VALUES (?, ?, ?)", (name, quantity, price))

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

def update\_item(self):

item\_id = simpledialog.askinteger("Input", "Enter item ID to update:")

name = simpledialog.askstring("Input", "Enter new item name:")

quantity = simpledialog.askinteger("Input", "Enter new item quantity:")

price = simpledialog.askfloat("Input", "Enter new item price:")

if item\_id and name and quantity is not None and price is not None:

execute\_db\_query("UPDATE inventory SET item\_name = ?, quantity = ?, price = ? WHERE item\_id = ?", (name, quantity, price, item\_id))

messagebox.showinfo("Success", "Item updated successfully")

def delete\_item(self):

item\_id = simpledialog.askinteger("Input", "Enter item ID to delete:")

if item\_id:

execute\_db\_query("DELETE FROM inventory WHERE item\_id = ?", (item\_id,))

messagebox.showinfo("Success", "Item deleted successfully")

def add\_customer(self):

name = simpledialog.askstring("Input", "Enter customer name:")

phone = simpledialog.askstring("Input", "Enter customer phone:")

if name and phone:

execute\_db\_query("INSERT INTO customers (customer\_name, customer\_phone) VALUES (?, ?)", (name, phone))

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

def make\_sale(self):

customer\_id = simpledialog.askinteger("Input", "Enter customer ID:")

item\_id = simpledialog.askinteger("Input", "Enter item ID:")

quantity = simpledialog.askinteger("Input", "Enter quantity sold:")

if customer\_id and item\_id and quantity:

execute\_db\_query("INSERT INTO sales (customer\_id, item\_id, quantity, sale\_date) VALUES (?, ?, ?, ?)", (customer\_id, item\_id, quantity, datetime.now().strftime("%Y-%m-%d %H:%M:%S")))

execute\_db\_query("UPDATE inventory SET quantity = quantity - ? WHERE item\_id = ?", (quantity, item\_id))

messagebox.showinfo("Success", "Sale recorded successfully")

def view\_inventory(self):

records = fetch\_db\_query("SELECT \* FROM inventory")

self.display\_records(records, ["Item ID", "Item Name", "Quantity", "Price"])

def view\_customers(self):

records = fetch\_db\_query("SELECT \* FROM customers")

self.display\_records(records, ["Customer ID", "Customer Name", "Customer Phone"])

def view\_sales(self):

records = fetch\_db\_query('''

SELECT sales.sale\_id, customers.customer\_name, inventory.item\_name, sales.quantity, sales.sale\_date

FROM sales

JOIN customers ON sales.customer\_id = customers.customer\_id

JOIN inventory ON sales.item\_id = inventory.item\_id

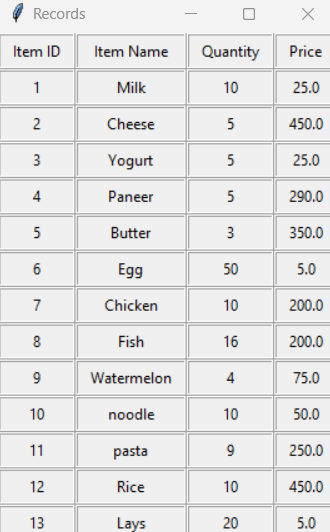
''')

self.display\_records(records, ["Sale ID", "Customer Name", "Item Name", "Quantity", "Sale Date"])

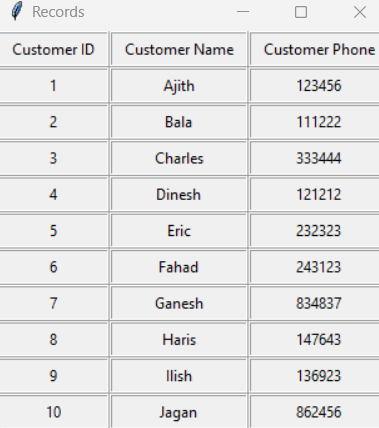
def view\_customer\_purchase\_history(self):

customer\_id = simpledialog.askinteger("Input", "Enter customer

**TABLE FOR INVENTORY**

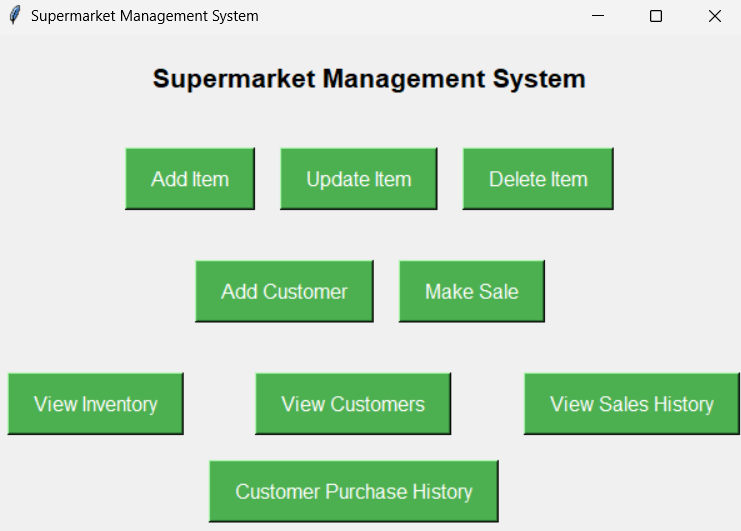
****

**TABLE FOR CUSTOMERS**

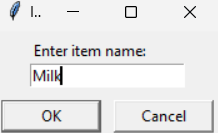
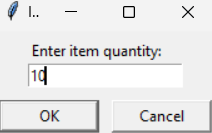
****

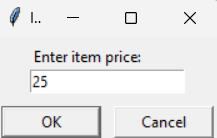
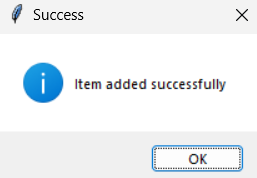
**OUTPUT**

* **STARTING INTERFACE:**

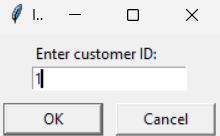
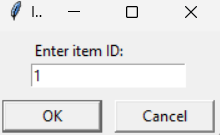
****

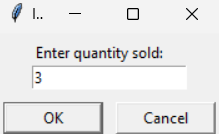
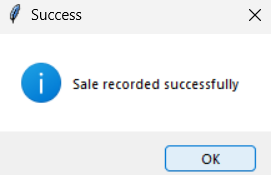
* **ADD ITEM INTERFACE:**

** **

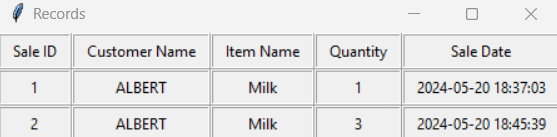
** **

* **MAKE SALE:**

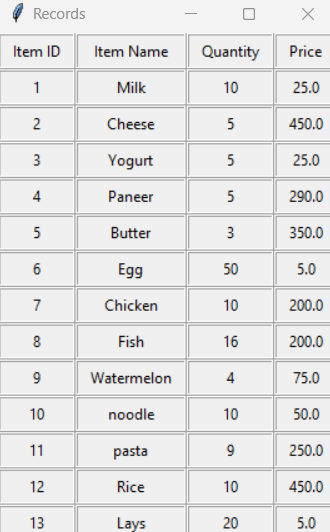
** **

** **

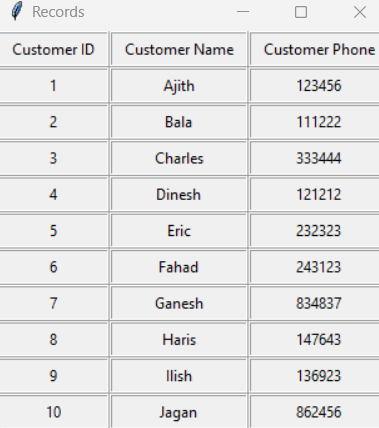
* **SALES HISTORY INSTERFACE:**

****

* **WHEN CLICKING THE VIEW INVENTORY :**

****

* **WHEN CLICKING THE VIEW CUSTOMERS:**

****

**RESULTS AND DISCUSSION**

The program above implements a basic supermarket billing system using SQLite for the database and Tkinter for the graphical user interface (GUI). Here are the key features and results of the program:

**Key Features**

1.Database Management:

The program creates four tables in SQLite: Customers, Items, Sales, and Sales\_History.

Functions for adding customers and items to the database.

Functions to retrieve inventory and sales history.

2.GUI Interface:

A main window with buttons for viewing inventory, sales history, adding customers, adding items, and processing sales.

Separate windows for each feature to add items/customers and to view inventory/sales history.

A sale processing window where items can be added to a sale, and the sale can be finalized.

Results

The system successfully manages customer and item data and processes sales, updating the inventory accordingly.

It provides a user-friendly interface to perform different operations related to supermarket management.

The sales processing functionality ensures that the inventory is updated correctly, and sales records are maintained.

**Discussion**

**Advantages:**

The program provides a straightforward solution for small-scale supermarket management.

The use of Tkinter ensures that the GUI is simple and easy to use.

**Limitations**:

The program could benefit from additional error handling and validation (e.g., checking for non-numeric input where numeric input is expected).

More complex features like report generation, user authentication, and role-based access control are not included.

**Future Improvements:**

Adding features such as data visualization for sales trends, advanced inventory management, and customer loyalty programs.

Implementing a more sophisticated user interface with modern GUI frameworks.

**6. CONCLUSION**

The supermarket billing system developed in this project demonstrates how SQLite and Tkinter can be combined to create a functional application for managing supermarket operations. The program successfully handles customer and item data, processes sales, and provides a user-friendly interface for interacting with the system. Future improvements can further enhance its functionality and usability, making it suitable for more extensive applications.

**7. REFERENCE**